T UAB/07995

Universidad Autónoma de Barcelona
Escuela Técnica Superior de Ingeniería
Departamento de Arquitectura de Computadores y
Sistemas Operativos

¿Podemos Predecir en Algoritmos Paralelos No-Deterministas?

Memoria presentada por **Paula Cecilia Fritzsche**
para optar al grado de **Doctora** por la Universidad
Autónoma de Barcelona.

Universitat Autònoma de Barcelona
Servei de Biblioteques
1501138308

¿Podemos Predecir en Algoritmos Paralelos No-Deterministas?

Memoria presentada por **Paula Cecilia Fritzsche** para optar al grado de Doctora por la Universidad Autónoma de Barcelona. Trabajo realizado en el Departamento de Arquitectura de Computadores y Sistemas Operativos de la Universidad Autónoma de Barcelona dentro del programa de Doctorado en Informática, opción A: "Arquitectura de Computadores y Procesamiento Paralelo", bajo la dirección del Dr. **Emilio Luque Fadón**.

Bellaterra, Julio 2007

Emilio Luque Fadón

Agradecimientos

En primer lugar, quiero expresar mi más sincera gratitud a la invalorable guía, paciencia y tiempo que me dedicara mi Director de Tesis, Doctor Emilio Luque Fadón durante esta investigación. Sus recomendaciones y sugerencias me orientaron en el desarrollo de este proyecto, haciendo que mi atención en procesamiento paralelo y distribuido se volcara particularmente a la predicción de rendimiento no determinista. Asimismo, sus aportes me permitieron entender la importancia de mi Tesis a nivel de grupo general de trabajo.

A la Doctora Dolores Rexachs del Rosario por su inestimable supervisión, conocimiento y orientación en el día a día. Su sensibilidad y estímulo constante son cualidades dignas de destacarse.

También deseo extender mi agradecimiento a la Doctora Ana Ripoll por sus constantes enseñanzas y directivas a lo largo de mis estudios. Sus continuos consejos, enfoques y desafíos han marcado positivamente mi trabajo.

Deseo agradecer al Doctor Remo Suppi por sus explicaciones y su ayuda en diversos temas relacionados con el ámbito académico. A esto debo sumar sus consejos a nivel humano. También deseo reconocer el compromiso de asistencia ante cualquier inconveniente de su equipo técnico, integrado por Daniel Ruiz Molina y Jordi Valls.

Hago extensiva mi gratitud a todos los docentes que me han formado en principio en mi Licenciatura de Ciencias de la Computación en la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires (Argentina) y luego en mis estudios de Postgrado en la Escuela Técnica Superior de Ingeniería de la Universidad Autónoma de Barcelona (España).

Quiero destacar la colaboración de mis compañeros de doctorado y miembros de CAOS: Aiman Omary, Alexander Guevara Quintero, Alexandre Strube, Alvaro Wong, Anna Cortés Fite, Angelo Duarte, Anna Morajko, Antonio Hervás Vílchez, Bahjat Qazzaz, Baker Abdalhaq, Carlos Moreno Losada, Crisos López, Christianne Orrico, Daniel Franco, Daniel Ruiz Molina, Darío Rodríguez Aseretto, Diego Lugones, Diego Mostaccio, Eduardo Argollo de Olivera, Eduardo César, Elisa Heymann, Enol Fernández del Castillo, Fernando Cores, Gemma Roqué, Genaro Costa, Germán Bianchini, Guna Santos, Gustavo Martínez, Hisham Ihshaish, Ihab Salawdeh, Javier Balladini, Joan Sorribes, John Corredor, Jordi Valls Pérez, Josemar Souza, José Ramón García, Josep Jorba, Juan Carlos Moure, Leandro Coelho, Manuel Brugnoli Rivera, María del Mar López, Mauricio Hanzich, Miquel

Angel Senar, Mónica Denham, Oriol Díez, Paola Caymes Scutari, Pilar Gómez Sánchez, Porfidio Hernández, Remo Suppi, Rodrigo Godoi, Tomás Margalef, Vicente Ivars, Xiao Yuan Yang, por sus constantes muestras de interés, paciencia y apoyo.

Un eterno agradecimiento a Dolores Rexachs y a mi MADRE quienes corrigieron y revisaron gramaticalmente este documento.

Doy las gracias a mis amigos / conocidos de la vida. Mis compañeros del primario, del secundario, de la universidad, de mis distintos trabajos, de maratones y de aventuras deportivas, de entrenamientos: Andrea Bugallo, Anna Esteve Poms, Ariel Stancato, Carmen Jimenez, Cata Rodhius, Cristina Gioveni, Dani Álvarez, Dany Conejero, David Serra, David Toro, Eric Martin, Fernando Beunza (junto con todos los 'Polenta'), Fernando Di Stefano, Gladys Ruffini (mi mejor amiga por siempre), mi 'hermana' Gloria Urbano, Héctor Montañés, Inés Martins, José Ángel Rivas, Laura Costa, Luli Pastoriza (mi pequeña gran amiga, en representación de todos los Ursuleños), Manel Masachs, Mar, Marcelo Gallo, Marcelo Szlifman, Maru Morales, Pato Giofre, Pep Guil, Ramón González, Toni Anglaril, Xavi Mimo y algunos que sin querer estoy dejando a un lado.

Quiero resaltar las pruebas de amistad que me dieran Diego Mostaccio, Gloria Urbano, Marcelo Gallo y Pilar Gómez Sánchez en ciertos momentos difíciles. También a mi hermana María Inés por preocuparse, ocuparse y ser mi sostén a nivel familiar.

No puedo olvidarme de quienes me han dado de almorzar / cenar en esta última etapa. Gracias Maru Morales, Mónica Denham y Xavi Mimo... por cierto, la comida de las peques es mucho mejor.

Xavi, Sara y Laia, todo mi cariño por los buenos momentos compartidos.

Por último, el mayor reconocimiento a mis padres (Ricardo y María Teresita), mis hermanos (Ricardo, Guillermo y María Inés) y mi familia en general, por su paciencia y amor; especialmente a mis hermosos sobrinos Melanie, Madelaine, Cecilia y Agustín. Cada uno tiene reservado un lugar especial de mi corazón. También en memoria de mi abuela Luisa de quien tanto aprendí.

GRACIAS A TODOS.

Paula.

Índice general

Agradecimientos I

Prólogo XI

1. Introducción 1

1.1. Predicción de rendimiento 3

1.2. Enfoque 4

1.3. Organización de la Tesis 8

2. Algoritmos 11

2.1. Problemática algorítmica 11

2.1.1. Definición de algoritmo 11

2.1.2. Corrección 11

2.1.3. Comparación de algoritmos 12

2.1.3.1. Modelo RAM de computación 12

2.1.3.2. Modelo PRAM de computación 12

2.1.3.3. Complejidad del mejor y del peor caso 14

2.1.4. La notación O 14

2.1.5. Razón de crecimiento 15

2.1.6. Eficiencia 16

2.1.7. Determinismo vs. no determinismo 17

2.1.8. Clasificación de problemas de decisión 17

2.1.8.1. La cuestión de si P = NP 18

2.2. Un problema no determinista: el TSP 18

6. Conclusiones y principales aportes **81**

6.1. Problemas en los que sería factible la aplicación de la metodología expuesta 82

6.2. Contribuciones 84

6.3. Líneas abiertas 84

A. Descubrir conocimiento en bases de datos **87**

A.1. Modelos de minería de datos 89

A.2. Técnicas de minería de datos 90

A.2.1. Clustering 91

A.2.1.1. Métodos jerárquicos 93

A.2.1.2. Métodos particionados 93

B. Ambiente Cluster-Frame **97**

B.1. Requerimientos del sistema 97

B.2. Instalación 97

B.3. Ejecución del ambiente Cluster-Frame 98

C. Experimentación **103**

C.1. Resumen del volumen de información procesado 108

Bibliografía **111**

Índice de figuras

2.1. Máquina de acceso al azar (RAM) 13

2.2. Máquina paralela de acceso al azar (PRAM) 13

2.3. Complejidad del mejor y del peor caso 14

2.4. Notaciones O y Ω 15

2.5. Ordenes de complejidad 17

2.6. Clasificación de un problema de decisión 17

2.7. Tour óptimo de 42 y 13509 ciudades en los Estados Unidos. Anuncio publicitario de una competencia para resolver el TSP para 33 ciudades (1962) 21

3.1. La ciencia computacional que complementa la ciencia teórica y la ciencia experimental 26

3.2. Arquitecturas MIMDs. 27

3.3. Comunicación entre procesadores 28

3.4. Paradigmas de programación 29

3.5. Fases de la metodología de trabajo 32

3.6. Extracción del conocimiento 33

3.7. Módulo de comparación de datos 36

3.8. Metodología de trabajo 37

4.1. Paradigma maestro / trabajador seguido por los algoritmos TSP 40

4.2. Caminos posibles para el viajante considerando 5 ciudades 40

4.3. Proceso de poda factible en los algoritmos TSP 41

4.4. Proceso de ramificación y poda 42

4.5. Nueva metodología de trabajo 54

4.6. Posibles cinco patrones gráficos para 15 ciudades 55
4.7. Ambiente Cluster-Frame 58
4.8. Correspondencia entre ciudades distantes y tiempos de ejecución 60
4.9. Correspondencia entre ciudades distantes y suma de distancias 62

5.1. Un patrón histórico, el mismo patrón histórico trasladado más rotado y el mismo patrón histórico escalado, consistiendo de 15 ciudades cada uno . . 66
5.2. Ciudades uniformemente distribuidas en una circunferencia. 68
5.3. Tendencia de los tiempos de ejecución (en seg.) para ciudades alineadas . 70
5.4. Ciudades alineadas separadas entre sí por un valor constante. 70
5.5. Tiempos de ejecución (en seg.) para el algoritmo de poda global consistente en 20 ciudades alineadas usando 32 nodos 72
5.6. Patrón gráfico para las 16 ciudades estudiadas. 73
5.7. Escalabilidad en cuento al número de procesadores para el algoritmo de poda global manteniendo constante 16 ciudades 73
5.8. Patrón gráfico para las 15 y 16 ciudades estudiadas. 74
5.9. Aumento del tamaño del problema (C) para el algoritmo de poda global usando 8, 16 y 32 nodos (P) 75

A.1. Proceso KDD 87
A.2. Ejemplo gráfico de clustering 91
A.3. Funcionamiento de k-means para k=2 95

B.1. Cluster-Frame: instalación 98
B.2. Cluster-Frame: pantalla principal 99
B.3. Cluster-Frame: selección de la muestra de casos 99
B.4. Cluster-Frame: configuración de distancias 100
B.5. Cluster-Frame: calidad aceptable 100
B.6. Cluster-Frame: antes de ejecutar el método 101
B.7. Cluster-Frame: evolución de la compactación 101
B.8. Cluster-Frame: casos por grupo 102
B.9. Cluster-Frame: información de un grupo 102

C.1. Experimentos ejecutados por algoritmo y número de nodos 109
C.2. Experimentos ejecutados por número de ciudades y número de nodos . . . 110

Índice de tablas

2.1. Razón de crecimiento de funciones conocidas medidas en nanosegundos. . 16

2.2. Historia de la evolución del TSP. 21

4.1. Características de los algoritmos TSP presentados. 42

4.2. Media y desviación media de los tiempos de ejecución (en seg.) por patrón gráfico para el algoritmo exhaustivo usando 8 nodos. 55

4.3. Tiempos de ejecución (en seg.) para cada patrón gráfico aplicando el algoritmo de poda local usando 8 nodos y cluster asignado según k-means (Cl). 56

4.4. Tiempos de ejecución (en seg.) para cada patrón gráfico (PD) aplicando el algoritmo de poda global usando 8 nodos, cluster asignado según k-means (Cl) y suma de distancias (SD). 57

5.1. Comparativa de los tiempos de ejecución (en seg.) del patrón histórico estudiado [PHist], del mismo patrón histórico rotado más trasladado [TrasRot] y del mismo patrón histórico escalado [Esc] usando 32 nodos. También se exhibe la desviación media (MD). 67

5.2. Media y desviación media de los tiempos de ejecución (en seg.) para el algoritmo exhaustivo de acuerdo con el número de ciudades trabajadas usando 32 nodos. 69

5.3. Media y desviación media de los tiempos de ejecución (en seg.) para el algoritmo de poda global de acuerdo con el número de ciudades trabajadas usando 32 nodos. 69

5.4. Coordenadas de la ciudad inicial, suma de las distancias y tiempo de ejecución (en seg.) del algoritmo de poda global para (a) 18 (b) 19 ciudades usando 32 nodos. 71

C.1. Experimentos analizados. 106

C.2. Experimentos analizados (continuación). 107
C.3. Experimentos analizados (continuación). 108

Prólogo

¿Qué es lo que determina que los objetos caigan sobre la tierra? ¿Por qué los planetas giran alrededor del sol? ¿Qué circunstancia mantiene a las galaxias juntas? Si una persona viajara a otro planeta, ¿por qué cambiaría su peso? Todas estas preguntas están relacionadas con un aspecto de la física: la gravedad.

Pero... ¿cómo empezó a gestarse la Ley de Gravedad Universal de Newton?

Galileo Galilei demostró que todos los objetos caen sobre la superficie de la tierra con la misma aceleración y que esta aceleración es independiente de la masa del objeto que cae.

Sin duda, Isaac Newton conocía este concepto, de ahí que formulara una teoría de la gravedad más amplia y extensa. A pesar de que Newton identificó correctamente la relación entre fuerza, masa y distancia, sólo fue capaz de estimar el valor de la constante gravitacional entre dichas cantidades.

El mundo debió esperar más de un siglo para que Henry Cavendish re-confirmara la teoría de Newton y estableciera la constante de la proporcionalidad con una exactitud de aproximadamente el 1 %.

Mucho tiempo después Albert Einstein demostró que la ley era sólo aproximadamente correcta y que no funcionaba cuando la gravedad se convertía en demasiado fuerte.

Desde entonces la constante gravitacional es un tema de estudio y debate que los científicos aún intentan determinar con exactitud.

En ciencias de la computación, ¿qué es evaluar el rendimiento en aplicaciones no deterministas? En las aplicaciones dependientes de los datos, ¿por qué los tiempos de ejecución pueden variar tanto según el conjunto de datos de entrada usado, incluso manteniendo constante su tamaño? ¿Es el problema del viajante un buen representante de esta clase de problemas?

Estas cuestiones, entre otras, marcaron nuestro desafío de intentar predecir el rendimiento en programas paralelos no deterministas. El ejemplo de Galileo, Newton, Cavendish, Einstein y el camino que ellos crearon para *hacer ciencia*, lo hemos aplicado a nuestro problema de predicción, en un intento de hacer ciencia según ese modelo.

Todos usamos un conjunto de métodos y técnicas para la adquisición, refinado y organización de conocimientos sobre la estructura de un conjunto de hechos objetivos y

accesibles a varios observadores. La aplicación de estos métodos y conocimientos conllevó la generación de más conocimiento objetivo en forma de predicciones concretas, cuantitativas y comprobables referida a hechos observables. Esto no es ni más ni menos lo que hoy en día se conoce como método científico; el proceso mediante el cual una teoría científica es aceptada por la comunidad científica o bien descartada.

Concretamente, nuestra propuesta partió de ciertas premisas y utilizó la ciencia computacional para extraer conocimiento de los experimentos diseñados. A medida que se fueron corroborando ciertas conjeturas, los experimentos fueron guiados hacia nuevos objetivos, los cuales a su vez dieron fruto a otros nuevos planteamientos. Las predicciones se fueron cumpliendo una a una.

Cualquier aporte en el área no determinista por mínima que sea representa un gran avance ante la falta de conocimiento general. Desde ya, todavía queda mucho por hacer. Sólo esperamos haber sumado un granito de arena en una gran montaña.

Capítulo 1

Introducción

¿Podemos predecir en algoritmos paralelos no deterministas? Sí, es posible. A lo largo de este documento se demostrará esta afirmación.

La evaluación científica de algoritmos para solucionar todo tipo de problemas es uno de los puntos claves en ciencias de la computación. En el terreno de ciencia computacional (una disciplina emergente) continuamente surgen nuevos desafíos.

Dentro del método científico, con frecuencia se menciona la ciencia computacional como el tercer paradigma científico, que complementa la ciencia teórica y la ciencia experimental (de laboratorio). La ciencia computacional como paradigma implica la cooperación entre los conocimientos en una disciplina científica o tecnológica (biología, medio ambiente, ingeniería) con el soporte de la matemática (modelado, análisis numérico, simulación) y la informática (la capacidad de cómputo) para la resolución de problemas en esa disciplina.

Por otra parte, la ciencia computacional logra ventaja no sólo a partir de las mejoras de hardware sino, lo que es más importante aún, en función de las mejoras de los algoritmos y en las técnicas matemáticas. También permite construir un modelo para hacer predicciones sobre qué puede pasar en un determinado momento, de modo tal de estar mejor preparados para hacer observaciones convenientes o entender en forma más adecuada lo que se está viendo. Asimismo, se puede usar la técnica computacional para satisfacer requerimientos que son demasiado costosos [1]. Por consiguiente, todos estos trabajos multi disciplinarios necesitan algoritmos más rápidos y eficientes que se ejecuten en computadores (por lo general paralelos) con el objeto de generar los modelos de cómputo y/o las simulaciones.

La computación paralela es una alternativa que permite alcanzar una solución en una fracción del tiempo original que emplearía un único computador. Es decir, más de un computador, o procesador, interconectados por una red trabajan en conjunto para resolver un problema común. Los objetivos de la computación paralela se pueden cumplir de diversas maneras y éstos difieren básicamente en la estrategia de comunicación de los procesos (memoria compartida, memoria distribuida), en el modelo de programación

paralelo (paso de mensajes, procesos ligeros -threads-, variables compartidas) y /o en el paradigma de programación (pipeline, maestro / trabajador, SPMD -Single Program Multiple Data-, divide y vencerás).

Como consecuencia de todo lo expresado, el incremento del uso de computadores paralelos y distribuidos para solucionar problemas científicos de gran escala o críticos en tiempo, ha generado la necesidad de predecir el rendimiento de aplicaciones tanto deterministas como no deterministas con el fin de aprovechar mejor los recursos disponibles. El rendimiento, también llamado prestaciones, es el grado de satisfacción que una aplicación encuentra de acuerdo con las expectativas de la persona involucrada en ésta [2]. Una manera sencilla de comparar sin subterfugios el rendimiento de diversos sistemas informáticos es utilizar el tiempo de ejecución de un programa o, más habitualmente, un conjunto de programas [3].

La predicción de rendimiento plantea cuestiones de tipo cuantitativo sobre el funcionamiento de una aplicación: ¿cuán rápidamente realiza las tareas?, ¿con qué aprovechamiento utiliza los recursos?, o, en el caso en que el sistema no funciona, ¿con qué probabilidad falla?, ¿cuánto tiempo tarda en fallar? Estas cuestiones acerca de la aplicación deben concretarse en el cálculo de ciertos índices de rendimiento. Los índices de rendimiento son indicadores cuantificables usados para representar algún aspecto del rendimiento del sistema. Pueden representar medidas de ejecución, de eficiencia, de utilización, de fiabilidad, de disponibilidad. Los diseñadores de computadores, los investigadores en general y los usuarios finales son los principales interesados en obtener valores reales del rendimiento esperado [2].

En este trabajo, el índice de rendimiento que se desea evaluar es el tiempo de ejecución, por lo cual en lo que sigue del texto se hablará de manera indistinta de rendimiento y tiempo de ejecución.

En particular, la predicción de rendimiento de aplicaciones dependientes de los datos es un problema extremadamente desafiante porque para un evento específico, conjuntos de datos de entrada similares (en apariencia) pueden causar variabilidad en los tiempos de ejecución. Por ende, el rendimiento en estos casos no depende únicamente del tamaño del problema (N) y del número de procesadores (P). Otros parámetros, como el valor que mide la dependencia de los datos, deben tenerse en cuenta.

Entre las aplicaciones dependientes de los datos de entrada más conocidas se pueden mencionar los algoritmos de ordenación, los algoritmos de búsqueda, el problema de satisfacibilidad booleana (*satisfiability problem* - SAT) [4], el problema de la partición de grafos (*graph partition*) [4], el problema de la mochila (*knapsack problem*) [5], el problema de empaquetar un conjunto de objetos (*bin packing*) [5], el problema de la planificación de movimiento (*motion planning*) [6], el problema del viajante (*traveling salesman problem* - TSP) [7] y los problemas prácticos que pueden ser formulados como alguno de los problemas mencionados.

Algunos ejemplos de problemas prácticos que requieren predicción del rendimiento, imprescindibles para ciencias de la computación, son la planificación de trabajos en un

computador [8], la asignación física de recursos [9] y la minimización de la cantidad de tiempo empleado por un trazador gráfico para dibujar una figura [4].

1.1. Predicción de rendimiento

El rendimiento de las aplicaciones puede estudiarse a muchos niveles de detalle. Cualquier análisis de una aplicación es sólo un análisis de un cierto modelo de ella. Un modelo es una abstracción que contiene sólo las variables significativas y sus relaciones, permitiendo al analista concentrarse en los puntos que él considera de interés [10]. En este trabajo, se estudia la predicción del tiempo de ejecución de aplicaciones dependientes de los datos de entrada en un computador paralelo.

La ejecución de toda aplicación paralela está influida por un número de factores no deterministas. Para estimar el rendimiento de tales aplicaciones existen dos líneas, la predicción estática y la predicción dinámica. La elección entre una u otra representa un compromiso fundamental entre el costo y la exactitud de la predicción que se desea [11].

Las técnicas que existen para la predicción dinámica, tal como la simulación, pueden proveer valores de predicción pero están limitadas a trabajar con parámetros y/o datos de entrada muy específicos.

Las técnicas que existen para la predicción estática producen información analítica e información de diagnóstico sobre cómo los parámetros simbólicos del programa / computador (el tamaño del problema, el número de procesadores, el ancho de banda cómputo / comunicación, entre otros) afectan el rendimiento, sin requerir ejecuciones costosas o múltiples simulaciones para cada parámetro o conjunto de datos de entrada diferente. Sin embargo, estas técnicas sólo pueden modelar el comportamiento dinámico del programa en un grado limitado, por lo cual la predicción puede resultar algo inexacta.

Algunas fuentes de comportamiento variable de un programa son el no determinismo introducido por [12]:

- La planificación de tareas en forma dinámica sobre un número limitado de procesadores.
- La comunicación entre procesos.
- La contención de recursos hardware y software (enlaces de comunicación, discos, memorias).
- La dependencia de un programa sobre el conjunto de datos de entrada. Especialmente en las aplicaciones de procesamiento de datos, tales como ordenación y búsqueda, los tiempos de ejecución pueden variar demasiado según el conjunto de datos de entrada usado, incluso manteniendo constante el parámetro que representa el tamaño del conjunto de datos.

A su vez, las técnicas estáticas se extienden desde las técnicas simbólicas de bajo costo a técnicas numéricas analíticas basadas en las cadenas de Markov [13], en las redes de Petri estocásticas [14, 15, 16], en las redes de cola [17, 18], en las álgebras de proceso [19, 20], en los grafos de tareas [21] o en híbridos (una combinación de grafos de tareas y redes de cola) [22, 23].

Brevemente, las *cadenas de Markov* son un caso particular de los procesos estocásticos y resultan adecuadas para el modelado y análisis de sistemas con espacio de estados discreto y que evolucionan en el tiempo. Sin embargo, su pobre capacidad de abstracción hace que desde un punto de vista práctico resulten inadecuadas. Las *redes de cola* se usan para el modelado y el análisis de sistemas informáticos. Su mayor nivel de abstracción, en comparación con las cadenas de Markov, las hicieron preferibles frente a ellas. Su principal problema es la carencia de una primitiva para modelar la sincronización. Las *álgebras de proceso* trabajan con un lenguaje textual mientras que las *redes de Petri* usan una notación gráfica. Ambos formalismos son enfoques que prevalecen en el modelado de sistemas de eventos discretos concurrentes, tanto en los aspectos de validación de propiedades lógicas como en los de análisis de prestaciones.

Mientras que en las técnicas numéricas analíticas la complejidad del tiempo de resolución puede ser exponencial debido a la explosión del espacio de estados, en las técnicas simbólicas el tiempo de ejecución se estima en términos de una expresión que conserva todos los parámetros de interés del algoritmo analizado. Por ese motivo, las técnicas simbólicas proveen máxima información de diagnóstico sobre el rendimiento de un programa.

En la literatura, hasta donde se ha podido relevar, no existen muchos trabajos de predicción de tiempo de ejecución de aplicaciones paralelas dependientes de los datos de entrada. Una manera en que se ha abordado este tema, es la descomposición de un algoritmo en secuencia de bloques donde un bloque es una operación de cómputo, de comunicación o de sincronización [24]. Otra, es el uso del modelado de rendimiento simbólico donde la carga de trabajo se representa en términos de un número de momentos estadísticos [11].

1.2. Enfoque

Esta Tesis intenta dar un paso adelante en un ámbito poco explorado. La misma presenta una nueva metodología práctica para la predicción de rendimiento de aplicaciones dependientes de los datos en un computador paralelo, lo cual refleja un comportamiento no determinista.

Dichas aplicaciones deben cumplir con determinadas características tales como requerir el cálculo masivo y estar preparadas para trabajar en paralelo con todo lo que esto involucra (sincronización, comunicación). La idea de trabajar en este sentido surge de la necesidad de dar respuesta a un número de aplicaciones paralelas frecuentes que normalmente son dejadas a un lado. Cualquier aporte en el área no determinista por mínima que sea representa un gran avance ante la falta de conocimiento general. No sólo hay que pensar

en el problema que se resuelve sino en el beneficio que representa para la familia de problemas que contiene al problema resuelto.

La propuesta parte de ciertas premisas y utiliza la ciencia computacional para extraer conocimiento de los experimentos diseñados (datos de entrada), en una primera fase, y luego proporciona un mecanismo de predicción que resulta bastante prometedor, en una segunda fase. El desarrollo se centra en aplicaciones paralelas (P procesadores), pero naturalmente todo lo expresado puede emplearse en aplicaciones series ($P = 1$).

La formulación de las hipótesis, la composición de los experimentos, la ejecución de la aplicación estudiada, el proceso de descubrir conocimiento, la interpretación del modelo y la evaluación de calidad son las etapas que forman parte de la fase de extracción del conocimiento, mientras que la etapa de comparación de datos pertenece a la fase de predicción.

Si bien las etapas más importantes, una vez que se conoce a fondo la aplicación estudiada, son la formulación de las hipótesis y la comparación de nuevos datos con datos históricos para dar una predicción del orden de rendimiento, no por ello las otras etapas son menos relevantes para la satisfacción general de la metodología. Es más, sin el proceso de descubrir conocimiento, involucrando la técnica y el algoritmo de minería de datos como apoyo fundamental, difícilmente se puedan plantear las suficientes y adecuadas hipótesis.

Una aplicación representativa dependiente de los datos de entrada es el problema del viajante (TSP), donde es difícil predecir el tiempo de ejecución por depender fuertemente del conjunto de ciudades de entrada. Este problema es de considerable significación no sólo desde el punto de vista teórico. Hay muchos casos importantes de problemas prácticos que pueden ser formulados como problemas TSP y muchos otros problemas son generalizaciones de este problema.

Además de utilizarse en la soldadura de conexiones en los circuitos impresos [25], en las áreas del transporte y en la logística [7], problemas que tienen la estructura TSP ocurren en el análisis de la estructura de cristales [26], en el manejo del material en un almacén [27], en la planificación de trabajos en un máquina [8], en los problemas de asignación física en computación [9], en la re-disposición del genoma [28], en la construcción del árbol filogenético [29], en la minimización de la cantidad de tiempo empleado por un trazador gráfico para dibujar una figura [4], entre otros tópicos.

Una variante TSP relacionada incluye el problema del viajante con un tiempo límite [30]. A su vez, el problema de orientación [31] es un caso especial del anterior. En MAX TSP el objetivo es encontrar el camino más largo [32]. El problema TSP de máximo envío (maximum scatter TSP problem) se usa en algunos procesos de manufactura donde es importante tener una separación sustancial entre consecutivas operaciones sobre una pieza y en aplicaciones que trabajan con imágenes médicas [33]. También existen problemas TSP más relevantes que surgen como sub- problemas de problemas combinatorios complejos. De éstos el más conocido e importante es el problema de rutas de vehículos; es decir, el problema de determinar para una flota de vehículos qué clientes deben ser servidos por cada vehículo y en qué orden cada vehículo debe visitar a los clientes asignados [34]. Por

todo lo antes expuesto, se puede decir que hay una tremenda necesidad de contar con algoritmos TSP.

Teniendo en cuenta lo expresado anteriormente y sumado a la aparente simplicidad de su formulación se eligió al TSP como problema de estudio. Tres algoritmos paralelos de TSP euclideano (el algoritmo exhaustivo, el algoritmo de poda local y el algoritmo de poda global) son presentados y luego estudiados minuciosamente para mostrar la utilidad y el beneficio de la metodología presentada.

Los algoritmos TSP tienen como entrada el número de ciudades y sus coordenadas, la ciudad de inicio y el número de procesadores entre otros datos, procesan los mismos y generan un gran volumen de información entre los cuales están el camino mínimo y su distancia (solución óptima). Los algoritmos fueron desarrollados con el objeto de analizar la influencia de distintos grados de indeterminismo en la predicción de rendimiento. Todos siguen el paradigma de programación maestro / trabajador, uno de los paradigmas más extensamente usado en el ámbito del paralelismo.

Asimismo, es importante mencionar que los algoritmos TSP buscan recursivamente todos los posibles caminos (permutaciones) a través de un proceso de ramificación y sólo difieren en la estrategia de poda usada. Específicamente, en el algoritmo exhaustivo no hay poda y el tour mínimo se calcula entre todas las posibles permutaciones, por lo cual la predicción resulta muy exacta. En el algoritmo de poda local, cada trabajador poda según una distancia mínima que va encontrando a medida que va procesando una tarea. Por otro lado, en el algoritmo de poda global, los trabajadores podan siempre que sea posible de acuerdo con un valor de distancia global mínima [35, 36].

Es indudable, que el orden del tiempo de ejecución de los algoritmos paralelos TSP de poda local y de poda global depende del número de ciudades (C), del número de procesadores (P) y de otros parámetros relacionados con los datos de entrada, $O(C, P, ?)$. Encontrar estos otros parámetros es la clave para obtener una buena o aceptable predicción de rendimiento. Actualmente, las hipótesis que se trabajan son que la suma de las distancias (SD) de una ciudad a las restantes ciudades y la desviación media de estas sumas ($MDSD$), juegan un papel preponderante en los tiempos de ejecución de los algoritmos mencionados, $O(C, P, SD, MDSD)$. En cambio, ni la suma de las distancias de una ciudad a las restantes ciudades ni la desviación media de estas sumas, afectan el tiempo final de ejecución en el algoritmo exhaustivo, $O(C, P)$.

En un principio, los experimentos para el problema TSP fueron paulatinamente creados de forma tal de extraer conocimiento intrínseco con un determinado fin. Durante el desarrollo del trabajo, al irse corroborando ciertas conjeturas, los experimentos fueron guiados hacia nuevos objetivos, los cuales a su vez dieron fruto a otros nuevos planteamientos. Esto significó, en una primera aproximación, agrupar las ciudades siguiendo patrones gráficos para finalmente terminar agrupando las ciudades según la combinación de dos variables, la suma de las distancias de una ciudad a las restantes más la desviación media de esta suma. La idea es que si una ciudad en particular está muy alejada de las otras ciudades, su suma de distancias a las otras ciudades será considerablemente mayor a la suma de

las distancias de las otras ciudades y, en consecuencia, su tiempo de ejecución también será bastante mayor al resto si se comienza el tour por esta ciudad. Es decir, la ubicación de la ciudad distante en el proceso de ramificación es determinante de la cantidad que se poda.

Como ya se expresó, los algoritmos TSP reciben el número de ciudades y sus coordenadas, la ciudad de inicio y el número de procesadores entre otros datos, procesan los mismos y, en contrapartida, generan un gran volumen de información que debe examinarse por medio de un proceso de descubrir conocimiento en bases de datos (knowledge discovery in databases - KDD).

Para el problema TSP la técnica de minería de datos llamada clustering (agrupamiento) es la elegida para obtener una partición del conjunto de datos en grupos de datos distintivos, de forma tal que los datos en cada grupo compartan unas ciertas características comunes que a la larga significan tiempos de ejecución similares. En particular, se aplica el algoritmo matemático particionado de aprendizaje no supervisado conocido como k-means [37], embebido en un ambiente dinámico y abierto de agrupamiento llamado Cluster-Frame [38]. En Cluster-Frame los resultados alcanzados aplicando diferentes parámetros pueden estudiarse desde distintos ángulos y compararse.

Por otro lado, las herramientas de visualización resultan de gran ayuda en esta instancia ya que dan soporte a la toma de decisiones. La acumulación de experimentos, el conocimiento y la experiencia analizada por medio de instrumentos gráficos permiten llevar a cabo investigaciones y proyectos de análisis de datos de forma más óptima en un menor tiempo.

La evaluación de calidad envuelve la validación de las hipótesis realizadas para el problema TSP. En la evaluación se observa si los experimentos en los clusters se reasociaron según las hipótesis planteadas. El porcentaje de aciertos en este sentido expresa la capacidad de predicción.

Los resultados experimentales para los algoritmos TSP son bastante prometedores; la capacidad de predicción fue mayor al 75 % para todos los casos desde el comienzo de la investigación. Luego este valor fue mejorando.

En general, de cada evaluación surgen nuevas conjeturas / ideas que, en una primera instancia, son analizadas con los datos existentes en la base de datos histórica y, más adelante si persisten como promesas, son procesadas del principio al fin por toda la metodología para su corroboración. Algunos experimentos sintéticos y ciertos experimentos que involucran transformaciones geométricas han servido para confirmar unas cuantas suposiciones.

En cuanto a la fase de predicción de la metodología para el TSP, se puede decir que para un nuevo conjunto de ciudades es posible dar un orden de predicción de rendimiento midiendo las diferencias entre este conjunto de ciudades (su patrón) y los patrones gráficos históricos.

Para esta operación de comparación se usan conceptos de geometría de reconocimiento de patrones bajo movimiento euclideano. Si el mayor valor de similitud de todos los

calculados resulta ser menor al número aceptable, con el nuevo conjunto de ciudades se compone un nuevo experimento que retroalimentará la base histórica una vez aplicada la fase de extracción de conocimiento. En caso contrario, el tiempo de ejecución estimado para el nuevo conjunto es el tiempo de ejecución registrado para el patrón histórico con el que se obtuvo el mayor valor de similitud.

Las estimaciones del orden de rendimiento para nuevos conjuntos de datos fueron bastante buenas (en el orden del 90 %). Estas fueron comparadas con los tiempos reales de ejecución.

1.3. Organización de la Tesis

En las secciones previas se ha expuesto el contexto de la investigación realizada como así también se han especificado los desafíos. Esta Tesis está organizada como sigue:

En el capítulo 2 se presenta el marco teórico necesario para entender ciertos conceptos que se irán usando a lo largo de todo este documento. Dentro de este marco, se introducen ciertas nociones generales de algoritmos, se exhibe la razón de crecimiento de diversas funciones con el fin de observar el orden de sus tiempos y se muestra la clasificación de los problemas de decisión. Asimismo, se plantea un problema no determinista concreto: el problema del viajante. De este problema, se da a conocer la historia de su evolución desde sus comienzos hasta el día de hoy y su complejidad computacional.

En el capítulo 3 se ofrece una introducción a las ciencias computacionales y a la evaluación de rendimiento en programas paralelos para luego presentar la metodología práctica que permite predecir el orden del tiempo de ejecución de aplicaciones paralelas dependientes de los datos. Se enuncian las fases y dentro de éstas las etapas que componen la metodología al mismo tiempo que se explican concienzudamente.

En el capítulo 4 se plantea como caso de estudio el problema del viajante. En una primera parte, se introducen los algoritmos paralelos TSP desarrollados y se brindan detalles de cada una de las implementaciones. En una segunda parte, se aborda el proceso de experimentación llevado a cabo que permitió definir la complejidad asintótica en tiempo para cada uno de los algoritmos TSP.

En el capítulo 5 se exhiben las validaciones experimentales del TSP realizadas con el objeto de mostrar la robustez de la metodología. Mediante experimentos sintéticos se respaldan las hipótesis definidas. Otros experimentos procuran demostrar la escalabilidad del problema tanto a nivel de procesadores como tamaño del problema. También, se investiga la predicción del tiempo de ejecución para nuevos conjuntos.

En el capítulo 6 se resumen y se delinean en forma general las conclusiones de la Tesis junto con las contribuciones y los caminos a seguir. Además para terminar, se exponen diversos problemas donde sería factible la aplicación de la metodología expuesta.

Finalmente, en el apéndice A se muestra un estudio del proceso de descubrir conocimiento en base de datos. En particular, se hace hincapié en la técnica de minería de datos conocida

como clustering y en el algoritmo particionado de aprendizaje no supervisado k-means por ser los utilizados.

En el apéndice B se enseña el ambiente de clustering dinámico y abierto llamado ClusterFrame. El mismo permite la evaluación de diversos algoritmos como k-means, fuzzy k-means, k-prototypes, k-modes, k-medoid, k-means^{+} y k-means^{++}, entre otros.

En el apéndice C se exponen todos los experimentos llevados a cabo a lo largo de la investigación y se resume gráficamente el volumen de información procesado.

La Tesis desarrollada en esta memoria es el resultado de un estudio científico realizado en el Departamento de Arquitectura de Computadores y Sistemas Operativos (DACSO-CAOS) de la Universidad Autónoma de Barcelona. El trabajo se enmarca en una de las líneas de investigación de nuestro grupo "Procesamiento Paralelo y Distribuido: desde la Especificación y Sintonización de Aplicaciones hasta la Arquitectura" [1].

[1]Referencia: MEyC-España bajo contrato TIN 2004-03388

Capítulo 2

Algoritmos

Este capítulo tiene por objeto brindar el marco teórico necesario para entender ciertos conceptos que se irán usando a lo largo del trabajo. Es imprescindible definir qué es un algoritmo y qué significa el término complejidad computacional, al mismo tiempo que se deben tener en cuenta las razones de crecimiento de ciertas funciones. Asimismo, se plantea un problema no determinista representativo que depende de los datos de entrada: el problema del viajante. La importancia del problema escogido se refleja no sólo desde el punto de vista teórico sino que también desde la perspectiva de los problemas prácticos que se presentan.

2.1. Problemática algorítmica

A continuación se brinda una introducción breve a las nociones generales de algoritmos.

2.1.1. Definición de algoritmo

Un algoritmo es un proceso que efectúa una tarea específica para resolver un problema general. Un problema algorítmico se especifica describiendo el conjunto completo de instancias que éste debe trabajar y las propiedades que la salida debe tener como resultado de ejecutar alguna de estas instancias. Formalmente, un algoritmo es una secuencia de instrucciones que determina completamente el comportamiento de una máquina de Turing (MT / modelo de computación).

2.1.2. Corrección

Los objetivos deseables en todo algoritmo son la corrección, la eficiencia (como se verá más adelante, sección 2.1.6) y la facilidad de implementación. Un algoritmo es correcto si resuelve un problema dado. Generalmente, todo algoritmo correcto tiene una demostración

de corrección, que es una explicación del por qué se sabe que el algoritmo para todas las instancias de entrada produce el resultado deseado [4].

2.1.3. Comparación de algoritmos

Los algoritmos son la parte más importante, durable y original de las ciencias de la computación porque pueden ser estudiados independientemente de un lenguaje o de una máquina.

Esto significa que se necesitan técnicas capaces de comparar algoritmos sin estar éstos implementados. Las herramientas más importantes son el modelo RAM de computación para algoritmos series [4], el modelo PRAM para algoritmos paralelos [39] y el análisis asintótico de la complejidad del peor caso tanto para algoritmos series como para algoritmos paralelos [4].

2.1.3.1. Modelo RAM de computación

El diseño de un algoritmo depende de un computador hipotético llamado máquina de acceso al azar (random access machine - RAM), que es el tradicional modelo secuencial de computación, también conocido como el modelo de von Neumann, Figura 2.1.

La RAM tiene M posiciones de memoria (direcciones), donde M es un número grande finito. En una posición de memoria se puede almacenar un dato y el acceso a ésta consume una unidad de tiempo. Además la RAM, contiene un único procesador, que opera bajo el control de un algoritmo secuencial. Cada instrucción es ejecutada antes de que el procesador continúe con la próxima instrucción. La unidad de acceso a memoria se usa para crear un camino (una conexión directa) entre el procesador y una posición de memoria. Bajo este modelo:

- cada operación simple (+, -, *, /, =, if, ...) consume exactamente una unidad de tiempo.
- los ciclos y subrutinas no son considerados operaciones simples sino la composición de varias operaciones simples.
- cada acceso a memoria invierte exactamente una unidad de tiempo y se tiene cuanta memoria se necesita.

En consecuencia, en este modelo se mide el tiempo de ejecución de un algoritmo contando el número de unidades de tiempo que toma una instancia del problema.

2.1.3.2. Modelo PRAM de computación

La máquina paralela de acceso al azar (parallel random access machine - PRAM) es el modelo paralelo de computación más extensamente utilizado. La PRAM fue desarrollada para proveer una plataforma de diseño de algoritmos teóricos que predeciría

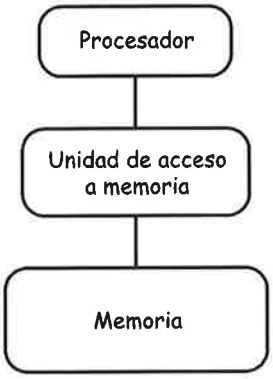


Figura 2.1: Máquina de acceso al azar (RAM).

el funcionamiento de dichos algoritmos en computadores paralelos reales. La ventaja (o desventaja según como se mire) de PRAM es que ignora las comunicaciones para permitir a los usuarios focalizarse en el paralelismo potencial disponible en el diseño de una solución a un problema dado.

La Figura 2.2 muestra las características de una PRAM donde mantiene n procesadores $P_1, ..., P_n$ cada uno idéntico a un procesador RAM. Se tiene una memoria común (global). La unidad de acceso a memoria de la PRAM es similar a la de RAM en el sentido que ésta asume que cada procesador consume una unidad de tiempo en acceder a cualquier posición de memoria.

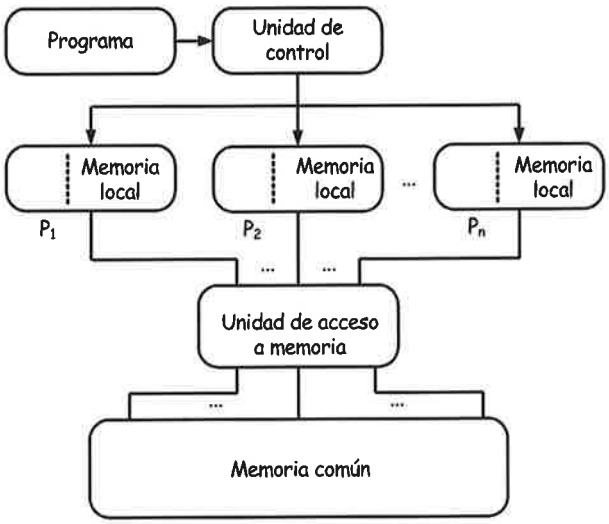


Figura 2.2: Máquina paralela de acceso al azar (PRAM).

Hay que observar que los procesadores no están directamente conectados entre sí, por lo cual, si un procesador desea comunicar algo, lo debe hacer a través de la memoria común.

Al acceder a memoria, tanto para una operación de lectura como para una operación de escritura, se deben resolver posibles conflictos mediante alguna política.

2.1.3.3. Complejidad del mejor y del peor caso

La complejidad del peor caso de un algoritmo es la función definida por el máximo número de unidades de tiempo que toma una instancia de tamaño N. En la Figura 2.3, la función mencionada está representada por la curva que pasa por los puntos más altos en cada columna. En la práctica, la complejidad del peor caso es la medida más útil.

La complejidad del mejor caso de un algoritmo es la función definida por el mínimo número de unidades de tiempo que toma una instancia de tamaño N. En la Figura 2.3, la función mencionada está representada por la curva que pasa por los puntos más bajos en cada columna.

Finalmente, la complejidad del caso promedio de un algoritmo es la función definida por el promedio de unidades de tiempo que toma una instancia de tamaño N.

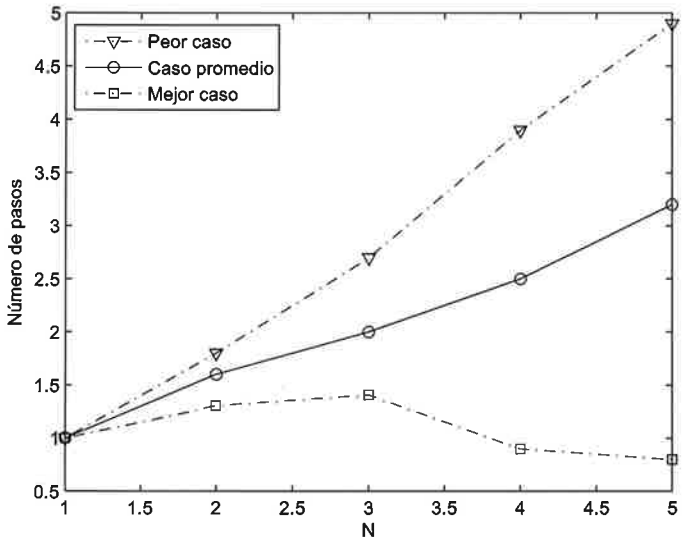
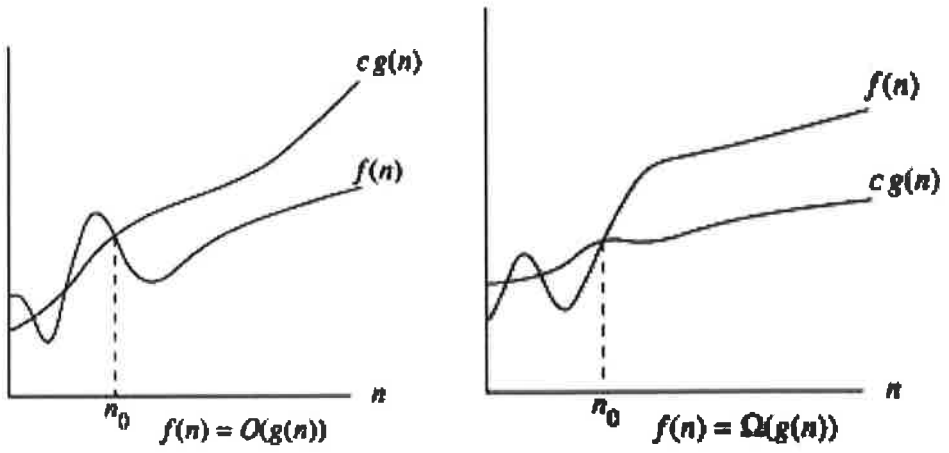


Figura 2.3: Complejidad del mejor y del peor caso; también se grafica el caso promedio.

2.1.4. La notación O

Las complejidades peor caso, mejor caso y caso promedio para un algoritmo son funciones numéricas del tamaño de las instancias. Como es difícil trabajar con ellas, se habla de límite superior e inferior de tales funciones. La Figura 2.4(a) y la Figura 2.4(b) ilustran el límite superior O y el límite inferior Ω, respectivamente [4].



(a) Límite superior (O) (b) Límite inferior (Ω)

Figura 2.4: Notaciones O y Ω.

- $f(n) = O(g(n))$ significa que $c * g(n)$ es un límite superior de $f(n)$. Así, existe una constante c tal que $f(n)$ es siempre menor o igual que $c * g(n)$, para n suficientemente grande.
- $f(n) = \Omega(g(n))$ significa que $c * g(n)$ es un límite inferior de $f(n)$. Así, existe una constante c tal que $f(n)$ es siempre mayor o igual que $c * g(n)$, para n suficientemente grande.
- $f(n) = \Theta(g(n))$ significa que $c_1 * g(n)$ es un límite superior de $f(n)$ y $c_2 * g(n)$ es un límite inferior de $f(n)$, para n suficientemente grande. Así, existen las constantes c_1 y c_2 tal que $f(n)$ es menor o igual que $c_1 * g(n)$ y $f(n)$ es mayor o igual que $c_2 * g(n)$. Esto significa que $g(n)$ es un límite hermético de $f(n)$.

2.1.5. Razón de crecimiento

La razón de crecimiento de funciones conocidas que se utilizan en distintos algoritmos medidas en nanosegundos se puede observar en la Tabla 2.1, significando s: segundos, d: días, m: meses y a: años.

Estudiando la Tabla 2.1, se pueden sacar unas cuantas conclusiones:

- Casi todos estos algoritmos insumen la misma cantidad de tiempo para n igual a 10.
- El algoritmo cuyo tiempo de ejecución es $n!$ no resulta útil cuando n se acerca a 20.
- El algoritmo cuyo tiempo de ejecución es 2^n tiene un mayor rango de operación pero no resulta práctico cuando n es mayor a 40.

Tabla 2.1: Razón de crecimiento de funciones conocidas medidas en nanosegundos.

n	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 a
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 s	$8{,}4 * 10^{15}$ a
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 m	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 d	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	$4 * 10^{13}$ a	
1000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10000	0.013 μs	10 μs	130 μs	100 ms		
100000	0.017 μs	0.10 ms	1.67 ms	10 s		
1000000	0.020 μs	1 ms	19.93 ms	16.7 m		
10000000	0.023 μs	0.01 s	0.23 s	1.16 d		
100000000	0.027 μs	0.10 s	2.66 s	115.7 d		
1000000000	0.030 μs	1 s	29.90 s	31.7 a		

- El algoritmo cuyo tiempo de ejecución es n^2 es perfectamente razonable hasta n igual a 100, pero éste rápidamente se va deteriorando con entradas más grandes, tal que si n es mayor a 1000000 deja de tener sentido.
- Los algoritmos n y $n \lg n$ son útiles con entradas de hasta un 1000000 de datos.
- Cuando se trabaja con algoritmos $O(\lg n)$ no se puede esperar encontrar un problema.
- Es un error pensar que basta esperar algunos años para que ciertos algoritmos costosos se puedan ejecutar en un tiempo razonable.

La Figura 2.5 muestra los órdenes de complejidad para las funciones $\lg n, n, n \lg n, n^2, n^3$ y 2^n.

2.1.6. Eficiencia

Un algoritmo es eficiente si su complejidad es polinómica, es decir, $O(n^k)$, donde $k \in N$. También se puede decir que es tratable.

$$\underbrace{\underbrace{O(1) \subset O(\lg n) \subset O(n) \subset O(n \lg n) \subset O(n^2)}_{\textit{Razonables en la práctica}} \subset \ldots \subset O(n^k)}_{\textit{Tratables}} \underbrace{\subset \ldots \subset O(2^n) \subset O(n!)}_{\textit{Intratables}}$$

Un buen diseñador de algoritmos es un experto de eficiencia; el diseñador siempre debe tener en mente los beneficios de un algoritmo eficiente.

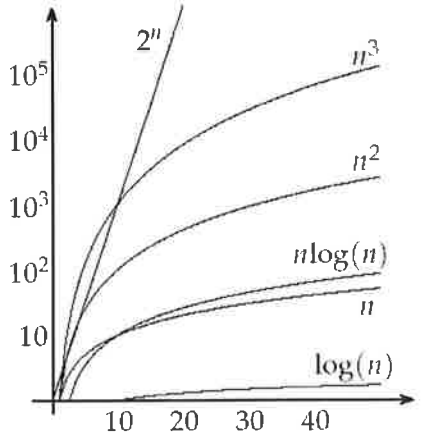


Figura 2.5: Ordenes de complejidad.

2.1.7. Determinismo vs. no determinismo

La palabra *determinista* denota la ausencia de elección. En un evento determinista se conoce precisamente qué pasará para cada conjunto de entrada. En contraste, el término *no determinista* se usa para expresar un evento cuyo resultado depende de los datos de entrada o de las elecciones realizadas por sus participantes.

El término no determinista se usó por primera vez en ciencias de la computación en conexión con la teoría de autómatas. En este campo, el término se refiere a un cierto tipo de máquina abstracta que obedece a un conjunto de instrucciones ambiguas en el sentido de que dos o más instrucciones pueden aplicarse en un determinado momento. Por otro lado, la expresión algoritmos no deterministas fue usada por Floyd, en primera instancia, para indicar aquellos programas que contenían instrucciones de elección [40].

2.1.8. Clasificación de problemas de decisión

Un problema de decisión es un problema en donde las respuestas posibles son *sí* o *no*. Un problema de decisión se puede clasificar como de clase P, clase NP, clase NP-completo o clase NP-duro [41].

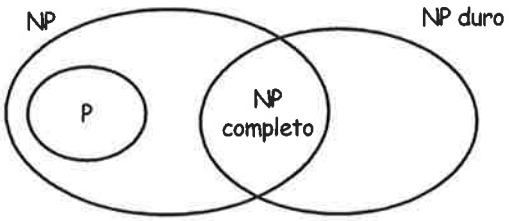


Figura 2.6: Clasificación de problemas de decisión.

Clase P: La clase P contiene aquellos problemas que pueden ser resueltos en tiempo polinómico por una máquina de Turing (MT) determinista. Esto indica a aquellos en los que, para cada par, estado / símbolo, existe a lo sumo una posibilidad de ejecución. Los problemas de complejidad polinómica resultan tratables, es decir, en la práctica se pueden resolver en un tiempo razonable.

Clase NP: La clase NP incluye los problemas que son resueltos por una MT no determinista en tiempo polinomial (de ahí su nombre, non-deterministic polynomial).

Clase NP-completo: La clase NP-completo abarca los peores problemas posibles de la clase NP y éstos se caracterizan por ser todos iguales en el sentido de que si se descubriera una solución P para alguno de ellos, esta solución sería fácilmente aplicable a todos ellos. Gráficamente se puede decir que estos problemas están en la frontera externa de la clase NP.

Clase NP-duro: La clase NP-duro involucra los problemas de decisión que son tan difíciles como un problema NP. Además, si se encontrara un algoritmo que resolviera uno de los problemas NP-duro en tiempo polinomial, entonces sería posible construir un algoritmo que trabajara en tiempo polinómico para cualquier problema de NP.

La clase NP-completo puede definirse alternativamente como la intersección entre la clase NP y la clase NP-duro.

2.1.8.1. La cuestión de si P = NP

La cuestión de si $P = NP$ es una de las más misteriosas e importantes en informática teórica [42].

Naturalmente, cualquier problema de P también se encuentra en NP. Si una MT determinista resuelve un problema, una no determinista también lo hará y en igual tiempo. Ahora, para mostrar que $P \neq NP$ se tendría que encontrar un problema en NP que no esté en P, mientras que para probar que $P = NP$, en principio, se tendría que encontrar un algoritmo polinómico para cada problema en NP.

Esta tarea en realidad se puede simplificar considerablemente. Es suficiente encontrar un algoritmo de tiempo polinómico para uno sólo de los problemas en la clase de los problemas llamados NP-completos.

2.2. Un problema no determinista: el TSP

El problema del viajante (TSP) es uno de los más famosos problemas (y quizás uno de los más estudiados) en el campo de la optimización combinatoria. A pesar de la aparente simplicidad de su formulación, el TSP es un problema de resolución compleja y la complejidad de su solución ha sido un continuo desafío para los matemáticos durante siglos [43, 44, 7].

No sólo el estudio de este problema ha atrapado a personas del área de matemáticas sino también a muchos investigadores de otros campos como investigación operativa, física, biología e inteligencia artificial que se han visto eclipsados por el mismo. En consecuencia, acerca de este problema se puede encontrar una gran cantidad de literatura. Por otro lado resulta llamativo que, aun para el caso general, no se conozca todavía un algoritmo efectivo que halle la solución en tiempo polinomial. Deben considerarse aún muchos aspectos del problema y algunas cuestiones que plantea están lejos de ser contestadas satisfactoriamente. Ciertamente, la resolución del TSP daría solución al problema P versus NP [1], el cual tiene un premio de 1000000 del Clay Mathematics Institute [45].

2.2.1. Planteamiento del problema

El TSP para C ciudades consiste en encontrar un tour que visite todas las ciudades exactamente una vez y que retorne a la ciudad inicial, de modo tal que sea mínima la suma total de las distancias entre las ciudades consecutivas. El requerimiento de retornar a la ciudad inicial no cambia la complejidad computacional del problema.

Una formulación equivalente en términos de teoría de grafo es: dado un grafo completo con pesos (los vértices son las ciudades, los ejes son los caminos y los pesos son los distintos costos) encontrar un circuito Hamiltoneano con el menor peso [46].

En el TSP euclideano, el peso de un camino entre dos ciudades es la distancia euclideana. Aunque se ha demostrado teóricamente que el TSP euclideano es igual de riguroso comparado con la forma general de TSP [47, 48], se sabe que existe un algoritmo de tiempo sub-exponencial para éste y además que muchas heurísticas trabajan mejor. De ahí que se exponga mayoritariamente en investigaciones de diversos ámbitos, además de ser *benchmark* para muchas áreas.

2.2.2. Problemas prácticos

El problema TSP es de considerable significación no sólo desde el punto de vista teórico. Hay muchos casos importantes de problemas prácticos que pueden ser formulados como problemas TSP y muchos otros problemas son generalizaciones de este problema. Además de utilizarse en la soldadura de conexiones en los circuitos impresos [25], en las áreas del transporte y en la logística [7], problemas que tienen la estructura TSP ocurren en el análisis de la estructura de cristales [26], en el manejo del material en un almacén [27], en la planificación de trabajos en una máquina [8], en los problemas de asignación física en computación [9], en la re-disposición del genoma [28], en la construcción del árbol filogenético [29], en la minimización de la cantidad de tiempo empleado por un trazador gráfico para dibujar una figura [4], entre otros tópicos.

Existen una cuantas variantes del problema TSP. Una de éstas incluye el problema del viajante con un tiempo límite [30]. A su vez, el problema de orientación [31] es un caso

[1] Sobre el problema P versus NP referirse a la sección 2.1.8 página 17

especial del anterior. En MAX TSP el objetivo es encontrar el camino más largo [32]. El problema TSP de máximo envío (maximum scatter TSP problem) se usa en algunos procesos de manufactura donde es importante tener una separación sustancial entre consecutivas operaciones sobre una pieza y en aplicaciones que trabajan con imágenes médicas [33]. También existen problemas TSP más relevantes que surgen como sub-problemas de problemas combinatorios complejos. De éstos el más conocido e importante es el problema de rutas de vehículos, es decir, el problema de determinar para una flota de vehículos qué clientes deben ser servidos por cada vehículo y en qué orden cada vehículo debe visitar a los clientes asignados [34]. Por todo lo antes expuesto, se puede decir que hay una tremenda necesidad de contar con algoritmos TSP.

2.2.3. Historia

Los primeros problemas matemáticos relacionados con TSP fueron tratados en el 1800 por el matemático irlandés William Rowan Hamilton y por el matemático británico Thomas Penyngton Kirkman [43].

El primer estudio sobre la forma general de TSP fue realizado por el matemático Karl Menger en Viena y luego en Harvard hacia 1930. Más tarde, el problema fue promovido por Hassler Whitney y Merrill Flood en Princeton [44].

La Figura 2.7(a) exhibe el primer *gran TSP*, un tour óptimo de 42 ciudades en los Estados Unidos resuelto en 1954. George Dantzig, Ray Fulkerson y Selmer Johnson publicaron la descripción de su método TSP e ilustraron la potencia de éste resolviendo una instancia para 49 ciudades, un tamaño imprevisible en aquel momento. La instancia consistía en elegir una ciudad de cada uno de los 48 estados de Estados Unidos (Alaska y Hawai fueron declarados estados solamente en 1959) más Washington D.C.; el costo de viajar entre estas ciudades fue definido por la distancia de las carreteras. Pero, en lugar de resolver este problema, ellos solucionaron el problema de 42 ciudades eliminando Baltimore, Wilmington, Philadelphia, Newark, New York, Hartford, y Providence. La ruta más corta entre Washington D.C. y Boston pasaba a través de las siete ciudades descartadas; en consecuencia, la solución para 42 ciudades derivaba en la resolución del problema para 49 ciudades [7].

En 1962, Proctor y Gamble originan una competencia cuyo anuncio publicitario se puede ver en la Figura 2.7(c). La competencia requería resolver un TSP para 33 ciudades específicas. Existió un empate entre las varias personas que encontraron el tour óptimo. El profesor Gerald Thompson de Carnegie Mellon University fue uno de los tantos ganadores [7].

Applegate, Bixby, Chvátal, y Cook (1998) encontraron el tour óptimo para 13509 ciudades en los Estados Unidos cuya población superaba los 500 habitantes, Figura 2.7(b).

En el 2005 se dio el último salto en la evolución de la historia del TSP. La visita de 33810 puntos en un circuito impreso fue resuelto usando Concorde [7].

Una completa y breve historia de la evolución del TSP se presenta en la Tabla 2.2.

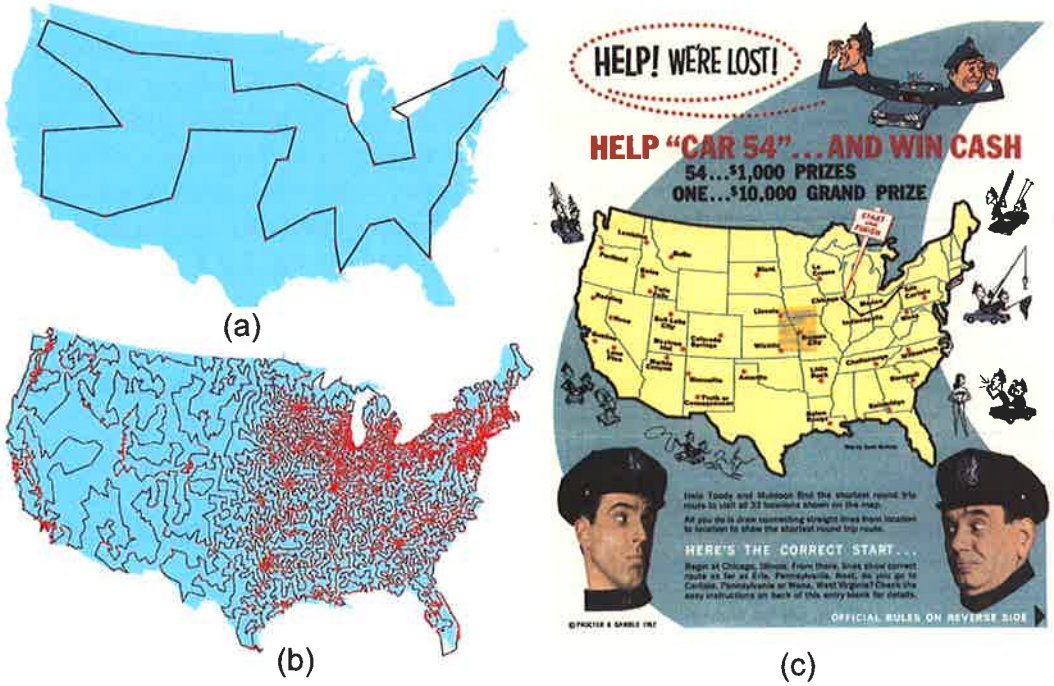


Figura 2.7: Tour óptimo de 42(a) & 13509(b) ciudades en Estados Unidos. (c) Anuncio publicitario de una competencia para resolver el TSP para 33 ciudades (1962).

Tabla 2.2: Historia de la evolución del TSP.

Año	Máx	Investigador/es	Ciudades / puntos / etc.
1954	49	Dantzig, Fulkerson y Johnson	Ciudades de los Estados Unidos
1962	33	Gerald Thompson	Una ruta de 33 localidades
1977	120	Groetschel	Ciudades alemanas del oeste
1987	532	Padberg y Rinaldi	Ciudades de los Estados Unidos
1987	666	Groetschel y Holland	Ciudades alrededor del mundo
1987	2392	Padberg y Rinaldi	Puntos de Tektronics Inc.
1994	7397	Applegate, Bixby, Chvátal y Cook	Ciudades varias
1998	13509	Applegate, Bixby, Chvátal y Cook	Ciudades de los Estados Unidos
2001	15112	Applegate, Bixby, Chvátal y Cook	Ciudades de Alemania
2004	24978	Applegate, Bixby, Chvátal y Cook	Ciudades de Suecia
2005	33810	Concorde	Puntos en una tarjeta de circuitos

2.2.4. Complejidad computacional

Se ha demostrado que el TSP es NP-duro [49]. Más precisamente, éste es completo para la clase complejidad (FP^{NP}) [2], y el problema de decisión (dado el costo y un número x, decidir si hay una ruta de ida y de vuelta más barata que x) es NP-completo. Específicamente, si se encuentra un algoritmo eficiente (por ejemplo un algoritmo que garantice encontrar una solución óptima en un número de pasos polinomial) para el problema del viajante, esto implicará que también podrían encontrarse algoritmos eficientes para todos los problemas pertenecientes a la clase NP-completo (ver sección 2.1.8).

La solución más directa para el problema TSP es calcular el número de tours diferentes a través de las C ciudades. Dada una ciudad inicial, esta tiene $C-1$ elecciones para la segunda ciudad, $C-2$ elecciones para la tercera ciudad, etc. Multiplicando estos valores se tiene $(C-1)! = (C-1)*(C-2)*...*2*1$ para una ciudad y $(C)!$ para las C ciudades. Otra solución sencilla es intentar todas las permutaciones (combinaciones ordenadas) y ver cuál es la menos costosa. Al final, el orden es también factorial del número de ciudades. Las dos soluciones presentadas son bastante similares y ellas resultan imprácticas porque el número de operaciones crece rápidamente.

La complejidad factorial del algoritmo motiva la investigación en dos líneas para atacar el problema: el algoritmo exacto (cuyo seudocódigo se exhibe en Algoritmo 1) y el algoritmo con la aplicación de heurísticas. Los algoritmos exactos buscan la solución óptima a través del uso de la técnica de ramificación y poda (branch-and-bound, 40 a 60 ciudades), de la programación lineal (120 a 200 ciudades) o ramificación y poda más cortes tomados de la programación lineal (5000 ciudades) [50].

Algoritmo 1 : TSP óptimo.

```
TSP_Optimo (C_1, ..., C_C) {
    d = ∞
    Para cada una de las permutaciones C_i de (C_1, ..., C_C)
        Si costo(C_i) ≤ d entonces d = costo(C_i) y C_min = C_i
    Retornar C_min
}
```

Las soluciones heurísticas para TSP son algoritmos de aproximación que en una fracción del tiempo del algoritmo exacto alcanzan una solución aproximada, cercana a la óptima. Los algoritmos heurísticos para resolver el TSP están basados originalmente en algoritmos genéticos o evolutivos[3] [51, 52], en temple simulado (*simulated annealing*)[4] [53], en búsqueda tabú (*tabu search*)[5] [54], en redes neuronales [55] o en colonias de hormigas [56].

[2] FP significa problemas de función (function problems)

[3] Los algoritmos genéticos son técnicas de búsqueda inspirados en los mecanismos de selección y genética natural, que combinan la capacidad de supervivencia de individuos con operadores genéticos.

[4] El temple simulado es una técnica basada en las observaciones del proceso de enfriado de metales; se puede decir que este algoritmo enfría la solución lentamente hasta alcanzar el objetivo más bajo posible.

[5] El procedimiento búsqueda tabú guía una heurística de escalamiento descendente con el objetivo de

2.3. Sumario

En este capítulo se ha cubierto una introducción a las nociones generales de algoritmos, desde su definición hasta las técnicas que permiten comparar tanto algoritmos series como algoritmos paralelos. Con el objeto de mostrar la racionabilidad de trabajar con cierto tipo de funciones se ha expuesto la razón de crecimiento de cada una de las funciones enunciadas. En particular, se ha planteado un problema no determinista representativo: el problema del viajante. La importancia del algoritmo escogido se ha reflejado no sólo desde el punto de vista teórico sino que también desde el punto de vista práctico. Asimismo, se definió la complejidad del TSP dándose a conocer previamente las distintas clases de problemas de decisión que existen y se expuso la historia de su evolución.

---

continuar la exploración evitando un retroceso a un óptimo local desde el cual ya se ha salido. En cada iteración se aplica un movimiento admisible a la solución actual, transformándola de esta manera en una solución vecina con menor costo. Se permiten las soluciones que incrementan la función de costo, mientras que el movimiento inverso está prohibido para algunas iteraciones con el objetivo de evitar la ocurrencia de ciclos.

Capítulo 3

Evaluación de rendimiento

En este capítulo se presenta la ciencia computacional, un paradigma emergente, donde continuamente surgen nuevos desafíos. También se introducen una serie de nociones para la evaluación de rendimiento en computación paralela. Finalmente, se expone la metodología de predicción de rendimiento de aplicaciones dependientes de los datos compuesta por las fases de extracción de conocimiento y de predicción.

3.1. Ciencia computacional

Dentro del método científico, con frecuencia se menciona la ciencia computacional como el tercer paradigma científico, que complementa la ciencia teórica y la ciencia experimental (de laboratorio); la Figura 3.1 esquematiza esta complementación.

La ciencia computacional como paradigma implica la cooperación entre los conocimientos en una disciplina científica o tecnológica (biología, medio ambiente, ingeniería) con el soporte de la matemática (modelado, análisis numérico, simulación) y la informática (la capacidad de cómputo) para la resolución de problemas en esa disciplina. En este trabajo, la ciencia computacional implica la colaboración de la disciplina informática, del soporte matemático representado por el proceso de descubrir conocimiento (involucrando la minería de datos) y por el modelo, y de la informática a través de la capacidad de cómputo paralelo.

Por otra parte, la ciencia computacional logra ventaja no sólo a partir de las mejoras de hardware sino, lo que es más importante aún, en función de las mejoras de los algoritmos y en las técnicas matemáticas. También permite construir un modelo para hacer predicciones acerca de lo que puede pasar en un determinado momento, de modo tal de encontrarnos preparados para hacer observaciones convenientes o entender de manera más adecuada lo que se está viendo. Asimismo, se puede usar la técnica computacional para satisfacer requerimientos que son demasiado costosos [1]. Por consiguiente, todos estos trabajos multi disciplinarios necesitan algoritmos más rápidos y eficientes que se

rendimiento del sistema. Pueden representar medidas de ejecución, de eficiencia, de utilización, de fiabilidad, de disponibilidad [2].

3.2.2.1. Algunos índices de rendimiento

Si se quiere medir la capacidad de respuesta de la aplicación, el índice *tiempo de respuesta* o *de ejecución* (tiempo transcurrido desde que se efectúa una entrada de datos al sistema hasta que se produce la respuesta correspondiente) es el que se debe utilizar, Ecuación 3.2.

$$T_{Ejecución} = T_{Final} - T_{Inicial} = T_{Cómputo} + T_{Comunicación} \tag{3.2}$$

Para estimar el *tiempo de cómputo* paralelo se debe tener en cuenta si los procesadores operan a una misma velocidad; la heterogeneidad trae aparejado el balanceo de carga (distribución del trabajo a realizar). El *tiempo de comunicación* depende entre otras variables del tamaño del mensaje, de la estructura de interconexión y del modo de transferencia. La Ecuación 3.3 puede representar lo anteriormente expresado:

$$T_{Comunicación} = T_{Inicialización} + n \cdot T_{Datos} \tag{3.3}$$

Asimismo, si se quiere ser estricto hay que considerar si el tiempo de comunicación se puede o no ocultar con el tiempo de cómputo total o parcialmente.

Si se quiere hablar de *rendimiento de un computador* se puede definir como el inverso del tiempo que tarda en ejecutar una aplicación, Ecuación 3.4. De esta manera, cuanto más rápido ejecute la aplicación, más alto será su rendimiento [3].

$$Rendimiento\ de\ un\ computador = \frac{1}{T_{Ejecución}} \tag{3.4}$$

Un índice de productividad sería el *throughput*, definido como el volumen de información procesado por la aplicación por unidad de tiempo.

Si la faceta de interés es la utilización de un componente concreto de la aplicación, el *índice de utilización* es aquel al que se debe hacer referencia. El índice de utilización es el porcentaje de tiempo en que dicho componente se usa.

Una medida de rendimiento entre un procesador y varios multiprocesadores es el factor *speedup* definido por el cociente entre el tiempo de ejecución en un procesador y el tiempo de ejecución en P procesadores, Ecuación 3.5.

$$Speedup = \frac{T_{Ejecución_1}}{T_{Ejecución_P}} \tag{3.5}$$

Otra métrica interesante que permite medir la efectividad con que un algoritmo usa los recursos computacionales de un computador paralelo, independientemente del tamaño del

problema, es la *eficiencia* definida por el cociente entre speedup y P procesadores [59], Ecuación 3.6:

$$Eficiencia = \frac{Speedup}{P} = \frac{T_{Ejecución_1}}{P * T_{Ejecución_P}} \tag{3.6}$$

El valor de la eficiencia puede estar comprendido entre 0 y 1.

Una vez precisados cuáles son los índices de rendimiento que se desean evaluar, una técnica de evaluación tiene por objeto el cálculo de dichos índices a partir de la observación / monitorización de la aplicación real (herramientas de medida) o bien mediante la utilización de un modelo simplificado del sistema que reproduzca el comportamiento y una cantidad de carga del mismo cercana a la prevista (técnicas de simulación o técnicas de modelado analítico) o bien por punto de referencia (benchmarking) con aplicaciones reales o modeladas.

Cabe enfatizar que, el índice de rendimiento que se trabaja a lo largo de esta Tesis es el tiempo de respuesta o de ejecución que, valga la redundancia, se obtiene a partir de la ejecución de la aplicación real. La utilidad de este índice está presente en un sinfín de tareas tales como la planificación y / o reserva de recursos, la administración del *timeout*, el escalamiento tanto a nivel de procesadores como a nivel de tamaño del problema, entre otras. Por lo tanto, la predicción del tiempo de ejecución resulta ser sumamente útil y a la vez complicada cuando se trata con aplicaciones no deterministas, como por ejemplo las aplicaciones dependientes de los datos.

Los diseñadores de computadores, los investigadores en general y los usuarios finales son los principales interesados en obtener valores reales del rendimiento de una aplicación en general.

En particular, la predicción de rendimiento de aplicaciones dependientes de los datos es un problema extremadamente desafiante porque para un evento específico, conjuntos similares (en apariencia) de datos de entrada pueden causar tiempos de ejecución muy variables. En consecuencia, el rendimiento en estos casos no depende únicamente del tamaño del problema (N) y del número de procesadores (P). Otros parámetros, que midan la dependencia de los datos deben tenerse en cuenta.

Entre las aplicaciones dependientes de los datos de entrada imprescindibles para ciencias de la computación están la planificación de trabajos en un computador [8], la asignación física de recursos [9] y la minimización de la cantidad de tiempo empleado por un trazador gráfico para dibujar una figura [4]. Por otro lado, como ya se mencionó, existen otras aplicaciones generales dependientes de los datos de entrada: los algoritmos de ordenación, los algoritmos de búsqueda, el problema de satisfacibilidad booleana (*satisfiability problem* - SAT) [4], el problema de la partición de grafos (*graph partition*) [4], el problema de la mochila (*knapsack problem*) [5], el problema de empaquetar un conjunto de objetos (*bin packing*) [5], el problema de la planificación de movimiento (*motion planning*) [6], el problema del viajante (*traveling salesman problem* - TSP) [7] y los problemas prácticos que pueden ser formulados como alguno de los problemas mencionados.

3.3. Metodología de predicción

La metodología de predicción propuesta en esta Tesis estima el tiempo de ejecución de aplicaciones paralelas dependientes de los datos. Dichas aplicaciones deben cumplir con determinadas características tales como ser complejas computacionalmente requiriendo el cálculo masivo, es decir deben llevar aparejado un elevado tiempo de cómputo en su resolución. La metodología es un proceso diseñado para explorar conjuntos de datos de entrada en búsqueda de patrones y / o relaciones y para predecir luego el orden de rendimiento destinado a nuevos conjuntos aplicando el conocimiento que se tiene de patrones almacenados en la base histórica.

De ahí que se hable de dos fases bien marcadas, la **extracción del conocimiento** desde los mismos datos mediante el uso de técnicas computacionales y la **predicción** en sí misma. La Figura 3.5 muestra las fases de la metodología de trabajo con sus entradas y salidas.

Figura 3.5: Fases de la metodología de trabajo.

3.3.1. Extracción del conocimiento

La fase de extracción del conocimiento de la metodología, Figura 3.6, se compone de los siguientes pasos:

- La formulación de hipótesis.
- La composición de experimentos.
- La ejecución de la aplicación estudiada.
- El proceso de descubrir conocimiento en bases de datos.
- La interpretación del modelo.
- La evaluación de calidad.

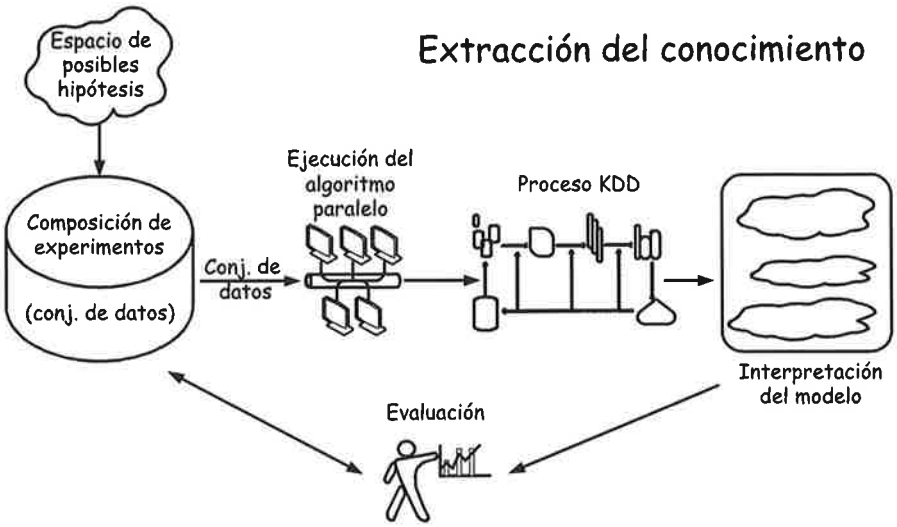


Figura 3.6: Extracción del conocimiento.

3.3.1.1. Formulación de las hipótesis

La formulación de hipótesis consiste en definir los parámetros claves que pueden afectar el tiempo de ejecución de la aplicación estudiada. En un principio, es importante entender el dominio de la aplicación y el conocimiento relevante a priori, para luego analizar su funcionamiento paso a paso, de una manera profunda. Es un trabajo arduo donde los especialistas deben identificar manualmente o automáticamente los puntos importantes.

Encontrar las hipótesis correctas es la base fundamental para construir grupos representativos de datos de entrada ya que, una vez ejecutados todos los pasos de la metodología, los resultados alimentan la base histórica con el objeto de tener en el futuro una buena capacidad de predicción. Esta predicción surgirá del cotejo de los nuevos datos con datos ya conocidos.

Es necesario aclarar que, la inclusión de demasiados parámetros puede derivar en un modelo más preciso pero al mismo tiempo complicado de entender o, más aún, irresoluble por la diversidad de variables. En consecuencia, se debe tener sumo cuidado en la selección de parámetros realizándose un compromiso razonable entre cantidad y precisión deseada.

3.3.1.2. Composición de experimentos

Definir un experimento implica la articulación de un objetivo, la elección de una salida que caracterice un aspecto del objetivo y la especificación de los campos que serán usados en el estudio (parámetros de la aplicación).

La creación de los experimentos debe hacerse con suma atención, teniendo en cuenta las hipótesis formuladas en la etapa previa. La cantidad de experimentos debe tratar de abarcar cada una de las clases representativas que surgen de combinar las distintas

hipótesis con las variables conocidas que pesan en el tiempo de resolución (tamaño del problema, cantidad de procesadores).

Se debe advertir que la necesidad de comprobar nuevas ideas irá estableciendo nuevos experimentos como así también el surgimiento espontáneo de experimentos singulares.

3.3.1.3. Ejecución de la aplicación estudiada

Antes de tenerse la aplicación paralela, se deben realizar una serie de elecciones entre las cuales se encuentran la manera de compartir la información, el modelo de programación a trabajarse y el paradigma de programación a usarse. Hay que tener en cuenta que cada decisión adoptada, puede impactar en la forma en que deben recibirse los experimentos, pudiendo ser necesario un preprocesamiento de los mismos.

Asimismo, para un adecuado examen del rendimiento entre el uso de un procesador y varios multiprocesadores destinados a una aplicación, se puede calcular el factor *speedup*.

También, es interesante llevar un control de la escalabilidad en cuanto al número de procesadores para saber hasta qué punto se mejora el rendimiento agregando máquinas. En este sentido, una métrica interesante es la *eficiencia*.

En general, existen diversas variantes para resolver un problema. Una vez que el estudio ha sido definido y tanto el tamaño del problema (N) como el número de procesadores (P) se han establecido, la aplicación paralela lee un experimento, procesa éste y obtiene los resultados. Estas acciones son repetidas con cada uno de los experimentos creados.

La obtención del tiempo de ejecución es el objetivo primordial de esta etapa. El fijar valores de N y de P permite analizar la influencia de otras variables que pesan en el tiempo de ejecución, las que están relacionadas con la dependencia de datos.

3.3.1.4. Proceso KDD

El proceso de descubrir conocimiento en bases de datos (knowledge discovery in databases - KDD) analiza los datos desde diferentes perspectivas y sumariza éstos en información útil. Esta tarea se desarrolla en diversas etapas entre las que se debe destacar la relacionada con la definición de un estudio y con la construcción de un modelo de conocimiento.

En la definición de un estudio se selecciona la técnica de minería de datos (también llamada exploración de datos) a aplicar. Una vez escogida la técnica, se debe seleccionar el algoritmo / método con el cual se trabajará para la búsqueda de estructuras.

La construcción de un modelo que sea realmente útil al problema requiere de muchas consideraciones y supone un importante trabajo objetivo. El modelo representa patrones de comportamiento observados en los valores de las variables del problema o en las relaciones de asociación entre dichas variables. Las variables del problema son las entradas de la aplicación dependiente de los datos estudiada.

El apéndice A contiene una descripción más amplia de todas las etapas que involucra un proceso KDD.

3.3.1.5. Interpretación del modelo

Una vez que los datos han sido minados, el proceso de interpretación del modelo creado desde los experimentos involucra distintos aspectos. El modelo resultante muestra principalmente las conjunciones que se consideran relevantes para describir el objetivo. Las herramientas de visualización constituyen una gran ayuda en esta instancia ya que otorgan soporte a la toma de decisiones. La acumulación de experimentos, el conocimiento y la experiencia analizadas por medio de instrumentos gráficos permiten llevar a cabo investigaciones y proyectos de análisis de datos de una forma más óptima en un menor tiempo.

3.3.1.6. Evaluación de calidad

En general, la precisión de los resultados obtenidos varía para cada uno de los algoritmos dependiente de los datos que se estudian. Esto se debiera tener presente en todo momento ya que lo esperado para un caso puede ser considerado no adecuado en otra circunstancia.

La validación de las hipótesis planteadas en un comienzo implica el cotejo de los resultados obtenidos con suposiciones conocidas, verificándose que las conclusiones que se arrojan sean válidas y suficientemente satisfactorias.

La evaluación de calidad, también debe proporcionar medidas de utilidad, de novedad y de interés para un análisis profundo de los datos. Es importante medir cualitativamente y cuantitativamente las características de los grupos obtenidos. Para esto se debe tener en mente el objetivo a alcanzar ya que las medidas cualitativas y cuantitativas no son mutuamente excluyentes. Cuando el objetivo reside en identificar cualitativamente tipos distintos, entonces lo importante es determinar el número correcto de tales tipos. En cambio si el objetivo es cuantitativo, se debe medir cuán buena es una partición (agrupamiento) y luego juzgar si la misma es suficientemente buena. Existen medidas internas y externas que permiten determinar la calidad de un grupo.

Esta etapa puede determinar un retorno a pasos previos con el fin de obtener información extra o más precisa.

3.3.2. Predicción

La fase de predicción se compone de la etapa de comparación de datos.

3.3.2.1. Comparación de datos

De la comparación entre un nuevo conjunto de datos con otros conjuntos de datos ya procesados, es posible predecir el orden del tiempo de ejecución para este nuevo conjunto. Si de las comparaciones surge que no existe un representante histórico para el nuevo

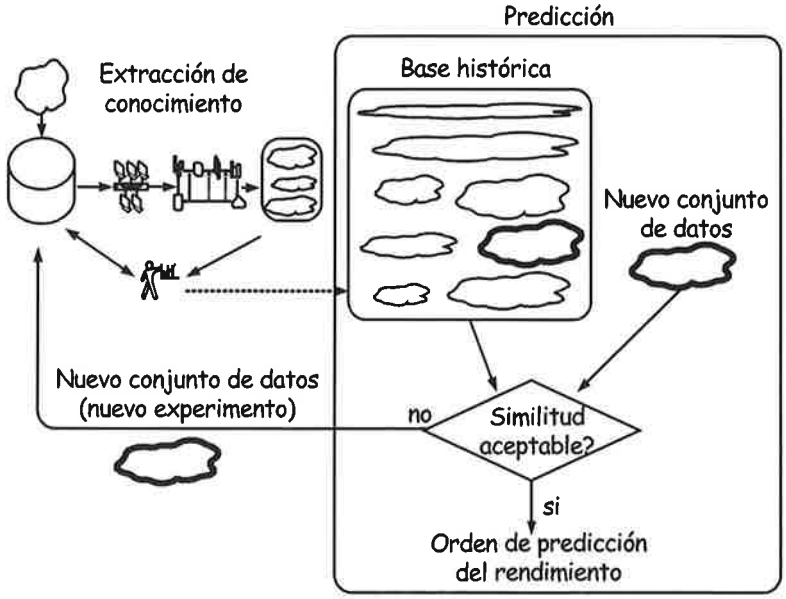


Figura 3.7: Módulo de comparación de datos.

conjunto, será necesario volver a la etapa de composición de experimentos para ampliar el modelo.

El nuevo conjunto de datos se coteja con cada conjunto histórico obteniéndose una medida de semejanza (habrá tantos valores de semejanza como conjuntos históricos). El orden del tiempo de ejecución del conjunto histórico perteneciente al par asociado a la mayor medida de semejanza, resulta ser la predicción candidata del orden de rendimiento para el nuevo conjunto. Se habla de candidata porque el valor de semejanza puede no ser aceptable y, en consecuencia, el nuevo conjunto pasa a formar parte de un nuevo lote de experimentos al que se le deberán aplicar todos los pasos de la metodología. La Figura 3.7 exhibe el proceso completo del módulo de comparación y la interacción con la fase de extracción de conocimiento.

El Algoritmo 2 describe el procedimiento que se sigue en el módulo de comparación; donde SA es la similitud aceptable, $D_1, ..., D_n$ son los datos particulares del nuevo conjunto, d va almacenando la máxima medida de similitud encontrada o en su defecto se inicializa con la similitud aceptable, BDH es la base de datos histórica, BDH_i es la instancia i de la base de datos histórica y max es el índice de la instancia en BHD que tiene la máxima medida de similitud de la base de datos histórica.

Algoritmo 2 : Similitud aceptable.

$Similitud_Aceptable\ (D_1, ..., D_n, BDH, SA)$ {
  $d = SA$
  $max = -1$
  Para cada uno de los datos historicos BDH_i *de* BHD {
    $aux = CalcularMedidaSimilitud(D_1, ..., D_n, BDH_i)$
    Si $aux > d$ {
      $d = aux$
      $max = i$
    }
  }
  Si $max = -1$
    Retornar -1
  Sino
    Retornar $Orden(BDH_{max})$
}

La Figura 3.8 grafica (resume) todo el circuito completo de la metodología. La clave está en la interacción entre la fase de extracción del conocimiento y la fase de predicción.

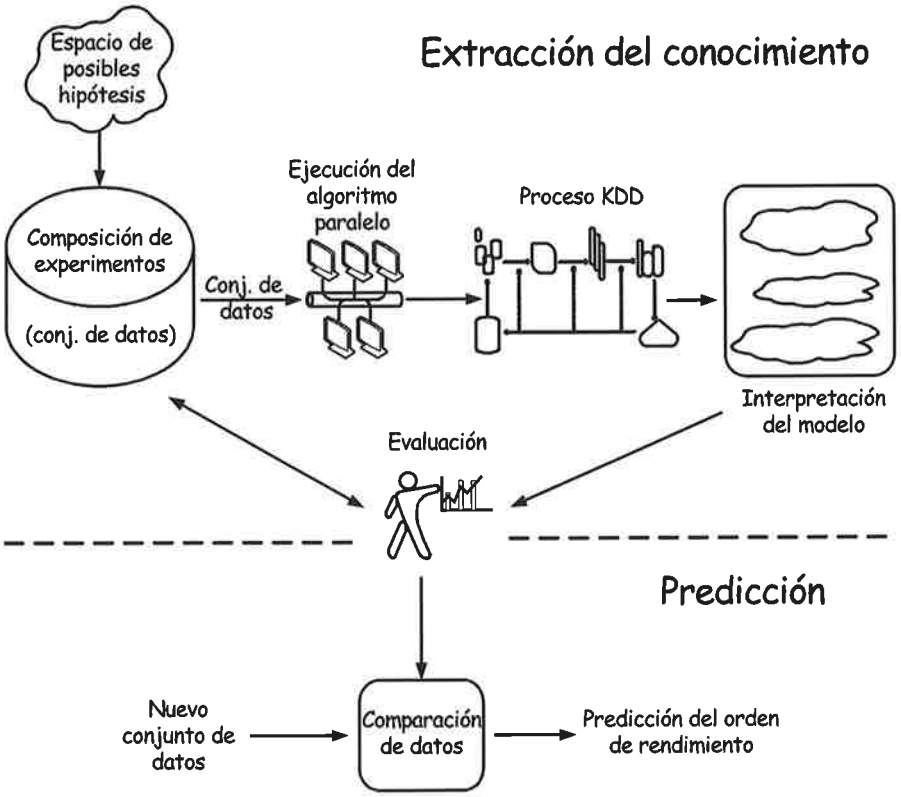


Figura 3.8: Metodología de trabajo.

3.4. Sumario

En este capítulo se ha ofrecido una introducción a las ciencias computacionales, ya que es el paradigma científico utilizado, y a los conceptos generales de computación paralela. La colaboración de la disciplina informática, del soporte matemático (representado por el proceso KDD y por el modelo) y de la informática a través de su capacidad de cómputo han permitido definir una metodología de predicción de rendimiento para aplicaciones paralelas dependientes de los datos. La metodología propuesta es un proceso diseñado para explorar conjuntos de datos de entrada en búsqueda de patrones y / o relaciones y para predecir luego el orden de rendimiento destinado a nuevos conjuntos aplicando el conocimiento que se tiene de patrones almacenados en la base histórica.

Capítulo 4

Un caso de estudio: el TSP

En este capítulo se presenta los tres algoritmos paralelos TSP (el algoritmo exhaustivo, el algoritmo de poda local y el algoritmo de poda global) que han sido desarrollados, estudiados y luego usados para mostrar la utilidad de la metodología práctica derivada en la sección 3.3. Estos algoritmos generan una solución óptima y fueron desarrollados con el objeto de analizar la influencia de distintos grados de indeterminismo en la predicción de rendimiento. Asimismo, se describen los pasos (las hipótesis) que permitieron descubrir los parámetros relevantes para definir la complejidad asintótica en tiempo de los algoritmos de poda local y de poda global.

4.1. Algoritmos TSP

Tres algoritmos para obtener la solución exacta TSP en una máquina paralela fueron desarrollados con el objeto de analizar la influencia de distintos grados de indeterminismo en la predicción de rendimiento. Estos algoritmos siguen la estructura maestro / trabajador (Figura 4.1) y buscan recursivamente todos los caminos posibles (permutaciones) a través de un proceso de ramificación y poda [35, 36].

Cada ciudad se representa por dos coordenadas en el plano euclideano. Considerando C ciudades diferentes, donde C_1 es la ciudad inicial, el maestro define un cierto nivel L para dividir las tareas. Las tareas son las posibles permutaciones de $C-1$ ciudades en L elementos ($\#tareas = (C-1)*(C-2)*...*(C-L)$). La granularidad G de una tarea es el número de ciudades que define la tarea sub-árbol $G = C - L$.

Un diagrama de las posibles permutaciones para 5 ciudades, considerando que el viajante comienza y finaliza su viaje en la ciudad 1, puede verse en la Figura 4.2. El maestro puede dividir este problema en 1 tarea de nivel 0 o 4 tareas de nivel 1 o 12 tareas de nivel 2 por ejemplo. Las tareas del primer nivel estarían representadas por las ciudades 1 y 2 para la primera tarea, 1 y 3 para la segunda, seguida por 1 y 4 para la tercera y finalmente 1 y 5 para la cuarta.

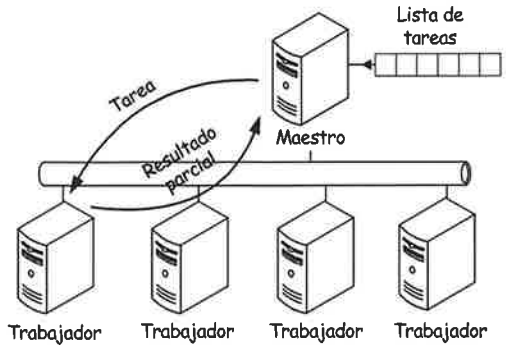


Figura 4.1: Paradigma maestro / trabajador seguido por los algoritmos TSP.

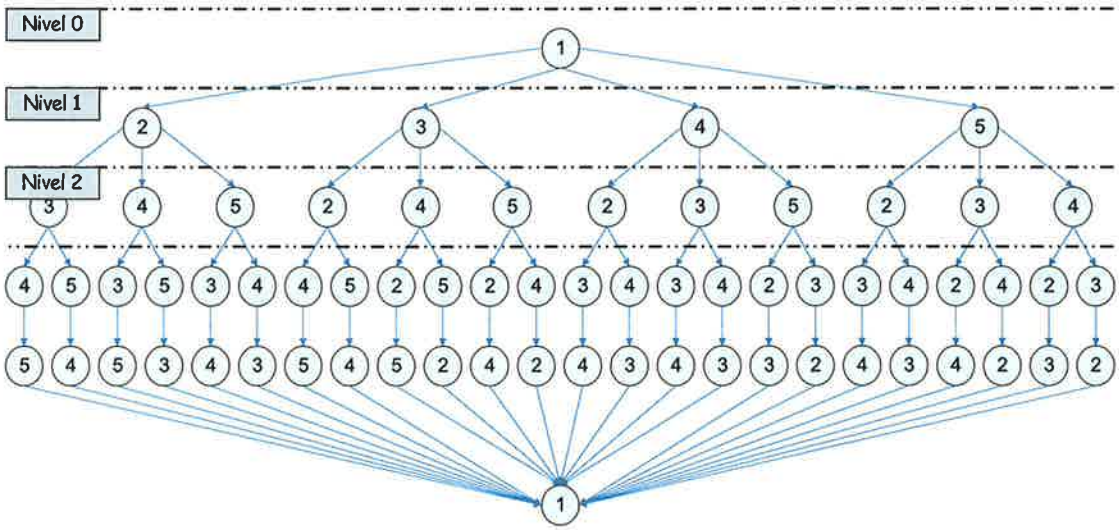


Figura 4.2: Caminos posibles para el viajante considerando 5 ciudades.

Cuando comienza la ejecución de un algoritmo, el maestro envía las coordenadas de todas las ciudades a cada trabajador. Cada trabajador, a su vez, es responsable de calcular las distancias de las permutaciones de las ciudades dejadas en la tarea sub árbol y luego enviar al maestro el mejor camino y distancia hallado.

La diferencia entre los algoritmos desarrollados radica en la estrategia de poda. Una de las características del problema TSP es que, una vez que la distancia para un camino es superior a la distancia mínima ya calculada, es posible podar este camino en el árbol.

Un proceso de poda se ilustra en la Figura 4.3 donde cada flecha contiene la distancia entre las dos ciudades que conecta. La distancia total para el primer camino seguido, a la izquierda de la Figura 4.3 es de 38 unidades. La distancia entre 1 y 2 en el segundo camino, a la derecha, es de 42 unidades. En consecuencia, el algoritmo no necesita seguir calculando las distancias desde la ciudad 2 hacia abajo porque es imposible alcanzar una

mejor solución por esta rama.

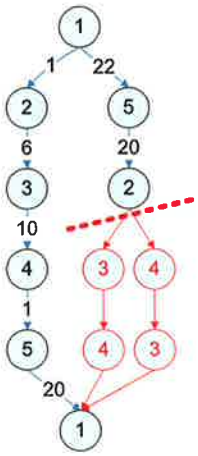


Figura 4.3: Proceso de poda factible en los algoritmos TSP.

Como se mencionó anteriormente, los tres algoritmos desarrollados para la solución de TSP difieren en la estrategia de poda. En el primer enfoque, que es llamado **exhaustivo** (*algoritmo a*), no hay poda y la distancia se calcula para todas las posibles permutaciones. En este caso se espera un funcionamiento predecible porque la cantidad de operaciones no cambia en función de la ordenación de las ciudades o de la ciudad inicial, pero el problema real es su costo prohibitivo en términos de tiempo.

Para el segundo enfoque, llamado **poda local** (*algoritmo b*), cada trabajador comienza una tarea sin una distancia mínima y poda de acuerdo con una distancia mínima encontrada hasta el momento dentro de dicha tarea. El problema que se puede encontrar es que los trabajadores muchas veces efectúan un trabajo inútil por desconocer una distancia global mínima. Existe una mejora importante en cuanto al tiempo de ejecución respecto del algoritmo exhaustivo pero, en contrapartida, no es tan predecible debido a que la ciudad inicial y el orden de asignación de las tareas resultan determinantes.

Para el tercer enfoque, **poda global** (*algoritmo c*), el maestro envía una tarea conjuntamente con una distancia global mínima alcanzada hasta el momento y los trabajadores podan, siempre que sea posible, basados en el valor de dicha distancia global o en un mínimo actual encontrado en la tarea. El tiempo de ejecución mejora notablemente con respecto a los otros dos algoritmos y la predicción si bien depende de la ciudad inicial, por otro lado no se ve influida por factores externos como la asignación de tareas.

El grado de predicción del rendimiento decrece del enfoque exhaustivo al de poda global, como así también decrece del enfoque de poda global al de poda local como se ilustra en la Tabla 4.1.

Tabla 4.1: Características de los algoritmos TSP presentados.

Algoritmo	Estrategia de poda	Dependiente de la ciudad inicial	Grado de predicción
Exhaustivo	No hay poda. Cada camino es evaluado	No	Alto
Poda global	Las ramas son podadas dentro de una tarea en base a una distancia global mínima o a un mínimo actual encontrado	Si	Promedio
Poda local	Las ramas son podadas dentro de una tarea en base a una distancia mínima encontrada en dicha tarea	Si	Bajo

4.2. Funcionamiento

Cada uno de los algoritmos TSP que generan una solución exacta, reciben el número de ciudades y sus coordenadas, la ciudad de inicio, el nivel (determinante de la granularidad de las tareas) y el número de procesadores.

A través de un proceso de ramificación y poda (Figura 4.4), que recursivamente busca todos los posibles caminos (permutaciones), obtiene el camino óptimo y su distancia, el tiempo consumido en alcanzar la solución y otros datos estadísticos junto con información relativa a la evolución del algoritmo.

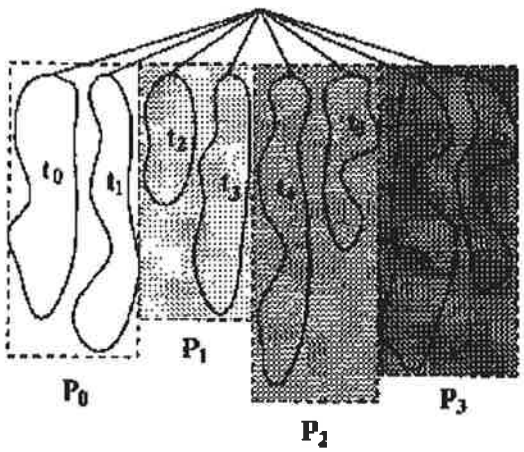


Figura 4.4: Proceso de ramificación y poda.

4.3. Implementación

En los algoritmos TSP, se conjugan dos entornos dominantes de hoy en día, C y MPI. C es un lenguaje de programación de propósito general y si bien se lo asocia al sistema UNIX, no está ligado a ningún sistema operativo ni a ninguna máquina. C proporciona las instrucciones de control de flujo lineal que se necesitan en los programas estructurados, tales como condiciones, ciclos, agrupamientos y subprogramas, pero por otro lado no permite multiprogramación, operaciones paralelas, sincronización. Todo esto último se soluciona con la interfaz MPI [60, 61]. Con MPI se tiene la posibilidad de contar con comunicaciones punto a punto, comunicaciones colectivas, sincronización, creación de procesos, entre otras operaciones.

4.3.1. Versión serie

Antes de programarse la versión paralela del algoritmo exhaustivo TSP, se construyó una versión serie donde cada una de las funciones participantes fue probada por separado para garantizar un adecuado funcionamiento.

El Algoritmo 3 muestra un seudocódigo del problema TSP en la versión serie. En esta primera aproximación no hay poda, por lo cual se calculan todos los posibles caminos. En el procedimiento principal *main* se inicializan las variables relacionadas con el registro de tiempos, se analizan los parámetros ingresados, se leen las coordenadas de las ciudades, se inicializan las variables que hacen referencia al camino óptimo y a su distancia, se ejecuta la función que busca el camino óptimo llamada $BuscarCaminoOptimo()$ y antes de finalizar se escribe el camino óptimo obtenido junto con la información del tiempo insumido.

A su vez, en la función $BuscarCaminoOptimo()$ se inicializa una solución parcial del camino óptimo junto con su distancia y luego se ejecuta la función de $Ramificación()$. En esta última, se van obteniendo los distintos caminos posibles de manera recursiva, guardándose en todo momento una solución parcial junto con su distancia y el camino óptimo actual también con su distancia.

4.3.2. Versiones paralelas

Para la programación de las versiones paralelas, se incluyeron en el código las siguientes librerías MPI, *mpi.h* y *mpicomm.h*.

Al igual que cualquier otra aplicación que trabaja con MPI, las aplicaciones TSP posibilitan las llamadas a funciones MPI realizando una inicialización del entorno mediante la primitiva $MPI_Init()$. De modo similar, después de haberse trabajado con todas las llamadas necesarias a funciones MPI se debe realizar una terminación de uso del entorno mediante la primitiva $MPI_Finalize()$.

MPI cuenta con un comunicador por defecto llamado *MPI_COMM_WORLD* que incluye todos los procesos envueltos en una ejecución paralela. Un comunicador es un dominio de comunicación que define un conjunto de procesos que se pueden comunicar entre sí. En consecuencia, los comunicadores MPI se usan para las comunicaciones de

Algoritmo 3 : Seudocódigo del problema TSP en la versión serie.

```
main (int argc, char *argv[]) {
  InicializarVariablesdeTiempos
  AnalizarParametrosIngresados
  LeerCiudades (ciudades, nroCiudades)
  Inicializar caminoOptimo y distCaminoOptimo
  .
  BuscarCaminoOptimo (ciudades, nroCiudades, caminoOptimo, distCaminoOptimo)
  .
  Escribir caminoOptimo, distCaminoOptimo y tiempos
}

BuscarCaminoOptimo (ciudades, nroCiudades, caminoOptimo, distCaminoOptimo) {
  Inicializar soluciónParcial y distSoluciónParcial
  .
  Ramificación (ciudades, nroCiudades, 0, soluciónParcial, distSoluciónParcial
                caminoOptimo, distCaminoOptimo)
}

Ramificación (ciudades, nroCiudades, nivel, soluciónParcial, distSoluciónParcial,
              caminoOptimo, distCaminoOptimo) {
  Si nivel = nroCiudades-1 {
    distSoluciónParcial += distancia (soluciónParcial[nivel], soluciónParcial[0])
    .
    Si distSoluciónParcial <distCaminoOptimo {
      distCaminoOptimo = distSoluciónParcial
      caminoOptimo = soluciónParcial
    }
  }
  Sino {
    Para cada ciudad (ciu=0;ciu<nroCiudades;ciu++) {
      Si ciu no está en soluciónParcial {
        distSoluciónParcial += distancia (soluciónParcial[nivel], ciu)
        soluciónParcial[nivel+1] = ciu
        Ramificación (ciudades, nroCiudades, nivel+1, soluciónParcial,
                      distSoluciónParcial, caminoOptimo, distCaminoOptimo)
        distSoluciónParcial -= distancia (soluciónParcial[nivel], ciu)
      }
    }
  }
}
```

paso de mensajes punto a punto o colectivas. Un proceso tiene un único identificador *id* dentro de un comunicador (0 a $nodos - 1$), donde *nodos* es el número de procesos.

En las aplicaciones TSP, los procesos cuyos identificadores van de 1 a $nodos - 1$ (procesos trabajadores) reciben datos del proceso 0 (proceso maestro) y envían datos también al proceso 0. Como el interés que se tiene es medir el tiempo de ejecución con la menor distorsión posible, se fuerza a que cada proceso se ejecute en un nodo independiente ($\#procesos = \#nodos$).

Las funciones $MPI_Comm_size()$ y $MPI_Comm_rank()$ se usan para determinar la cantidad de procesos total (*nodos*) con que se trabaja y el identificador del proceso actual *id*, respectivamente. La secuencia de llamada a estas rutinas es la siguiente:

$$MPI_Comm_size(\ MPI_Comm\ comm,\ int * nodos\ )$$

$$MPI_Comm_rank(\ MPI_Comm\ comm,\ int * id\ )$$

donde *comm* es el comunicador, *nodos* es el número de procesos asociado a un comunicador y *id* es el identificador del proceso.

Las aplicaciones TSP siguen el paradigma de programación maestro / trabajador, por lo cual cada declaración de variable global se duplica en cada proceso. Las variables que no necesitan duplicarse se declaran localmente dentro del código del proceso como se muestra en el seudocódigo del Algoritmo 4. También, es interesante remarcar que todas las comunicaciones son realizadas a través del maestro. Esto implica que los trabajadores no dan a conocer la distancia mínima encontrada a sus pares sino que la comunican al maestro. A su vez, el maestro, dependiendo del algoritmo en cuestión retransmite o no este valor a los otros trabajadores.

Algoritmo 4 : Declaración de variables locales en la aplicación TSP.

```
MPI_Comm_rank(MPI_COMM_WORLD, &id);
Si (id == 0) {
    int x, y;                    // Variables locales al proceso 0 (maestro)

} Sino {
    Si (id == 1) {
        int x, y;                // Variables locales al proceso 1 (trabajador 1)

    }
}
```

En la aplicación TSP, el proceso maestro realiza la distribución de los parámetros y de los datos a través de la primitiva de envío bloqueante $MPI_Send()$, mientras los procesos trabajadores realizan la recepción de esta información a través de la primitiva bloqueante $MPI_Recv()$.

El formato de la primitiva de envío bloqueante es:

*MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)*

donde *buf* es la dirección del buffer en la que se encuentran los datos a enviar, *count* es el tamaño de los elementos a enviar, *datatype* es el tipo de dato que se envía, *dest* es el nodo destino, *tag* es la marca del mensaje y *comm* es el comunicador.

El formato de la primitiva de recepción bloqueante es:

*MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)*

donde *buf* es la dirección del buffer donde se reciben los datos enviados, *count* es el tamaño de los elementos a recibir, *datatype* es el tipo de dato que se recibe, *source* es el nodo fuente, *tag* es la marca del mensaje, *comm* es el comunicador y *status* es el estado del objeto.

Los tiempos en la aplicación TSP se miden usando la primitiva *MPI_Wtime()* y la función *getrusage*. *MPI_Wtime()* mide el tiempo total en que incurre una tarea desde su inicio hasta su fin pudiendo contemplar tiempos de tareas ejecutadas por otros usuarios en forma concurrente. Por otro lado, la función *getrusage* permite conocer el tiempo que se demora en procesar una tarea desde su principio hasta su fin para el usuario dueño de dicha tarea y, además, el tiempo de sistema. El Algoritmo 5 muestra el registro que llevan los tiempos en la aplicación TSP.

Algoritmo 5 : Registro que lleva los tiempos en la aplicación TSP.

```
void uswtime(USWtime *tm)
{
struct rusage buffer;

tm->wall = MPI_Wtime();
getrusage(RUSAGE_SELF, &buffer);
tm->user = (double) buffer.ru_utime.tv_sec + 1.0e-6 * buffer.ru_utime.tv_usec;
tm->sys = (double) buffer.ru_stime.tv_sec + 1.0e-6 * buffer.ru_stime.tv_usec;
}
```

4.3.2.1. Algoritmo exhaustivo

Hay que recordar que en el algoritmo exhaustivo no hay poda y la distancia se calcula para todas las posibles permutaciones. Por lo tanto, se espera un funcionamiento predecible ya que la cantidad de operaciones no cambia en función de la ciudad inicial. El problema real es su costo prohibitivo en términos de tiempo. En las validaciones experimentales, los tiempos de ejecución generados por este algoritmo sirven para marcar las diferencias significativas con respecto a los tiempos de ejecución obtenidos por los otros algoritmos.

Algoritmo 6 : Seudocódigo del algoritmo paralelo exhaustivo.

```
#include "mpi.h"
#include "mpicomm.h"

main (int argc, char *argv[]) {
  .
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nodos);
  MPI_Comm_rank(MPI_COMM_WORLD, &id);
  .
  Si (id == MAESTRO) {
    Maestro()
  } Sino {
    Trabajador()
  }.
  MPI_Finalize();
}.

Maestro() {
  LeerCiudades (ciudades, nroCiudades)
  .
  Para cada trabajador (trab = 1; trab <nodos; trab++) {
    MPI_Send(trab, nroCiudades, nivel, ciudades)
  }
  Inicializar caminoOptimo y distCaminoOptimo
  .
  Para cada trabajador (trab = 1; trab <nodos; trab++) {
    MPI_Send(trab, tarea)
    tareasEnviadas++;
  }
  Mientras (tareasRecibidas<tareasEnviadas) {
    MPI_Recv(trab, soluciónParcial, distSoluciónParcial)
    tareasRecibidas++;
    .
    Si distSoluciónParcial <distCaminoOptimo {
      distCaminoOptimo = distSoluciónParcial
      caminoOptimo = soluciónParcial
    }
    .
    MPI_Send(trab, tarea)
    tareasEnviadas++;
  }
}
```

```
Trabajador() {
  MPI_Recv(nroCiudades, nivel, ciudades)
  MPI_Recv(tarea)
  Mientras reciba tareas {
    InicializarTiempos
    BuscarCaminoOptimo (ciudades, nroCiudades, nivel, caminoOptimo,
                distCaminoOptimo)
    FinalizarTiempos
    .
    MPI_Send(caminoOptimo, distCaminoOptimo)
    MPI_Recv(tarea)
  }
}

BuscarCaminoOptimo (ciudades, nroCiudades, nivel, caminoOptimo,
                distCaminoOptimo) {
  // Inicializar soluciónParcial y distSoluciónParcial
  Para las primeras ciudades según nivel (ciu=0; ciu <nivel; ciu++) {
    soluciónParcial [ciu] = ciudades [ciu]
    Si ciu >0 {
      distSoluciónParcial += distancia(ciudades [ciu-1], ciudades [ciu])
    }
  }
  .
  RamificaciónyPoda (ciudades, nroCiudades, nivel-1, soluciónParcial,
              distSoluciónParcial, caminoOptimo, distCaminoOptimo)
}

RamificaciónyPoda (ciudades, nroCiudades, nivel, soluciónParcial, distSoluciónParcial,
            caminoOptimo, distCaminoOptimo) {
  .
  Si nivel = nroCiudades-1 {
    distSoluciónParcial += distancia (soluciónParcial[nivel], soluciónParcial[0])
    .
    Si distSoluciónParcial <distCaminoOptimo {
      distCaminoOptimo = distSoluciónParcial
      caminoOptimo = soluciónParcial
    }
  } Sino {
    Para cada ciudad (ciu=0; ciu <nroCiudades; ciu++) {
      Si ciu no está en soluciónParcial {
        distSoluciónParcial += distancia (soluciónParcial[nivel], ciu)
        soluciónParcial[nivel+1] = ciu
        RamificaciónyPoda (ciudades, nroCiudades, nivel+1, soluciónParcial,
                    distSoluciónParcial, caminoOptimo, distCaminoOptimo)
        distSoluciónParcial -= distancia (soluciónParcial[nivel], ciu)
      }
    }
  }
}
```

El Algoritmo 6 exhibe un seudocódigo del algoritmo exhaustivo TSP que contiene las primitivas mencionadas a lo largo de esta sección. Básicamente, la aplicación TSP inicializa el ambiente MPI, determina el número de procesos (*nodos*) asociado al comunicador MPI_COMM_WORLD y obtiene el *id* del proceso.

En el procedimiento principal *main* se analiza si el proceso es un proceso maestro ($id = 0$) o un proceso trabajador ($id \neq 0$) y se actúa, en consecuencia, llamando a la función $Maestro()$ o $Trabajador()$ según corresponda.

En la función $Maestro()$, se analizan los parámetros ingresados, se comunican todos los datos necesarios de inicialización junto con la primera tarea a cada trabajador y luego se van repartiendo una a una las restantes tareas los trabajadores que lo solicitan, hasta que todas las tareas han sido procesadas. Antes de finalizar el ambiente MPI, el proceso maestro informa el camino óptimo y su distancia junto con el tiempo que le ha llevado la obtención del mismo.

A su vez, la función $Trabajador()$ recibe en un comienzo los datos necesarios de inicialización y una primera tarea, inicializa las variables asociadas al registro de tiempos, busca el camino óptimo para la tarea recibida a través de la función $BuscarCaminoOptimo()$, finaliza el registro de tiempos, comunica el camino óptimo encontrado y su distancia al proceso maestro y aguarda la llegada de una nueva tarea que le ha de suministrar el proceso maestro.

A su vez, en la función $BuscarCaminoOptimo()$ se inicializa una solución parcial del camino óptimo junto con su distancia y luego se ejecuta la función de $RamificaciónyPoda()$. En esta última, se van obteniendo los distintos caminos posibles de manera recursiva, guardándose en todo momento una solución parcial junto con su distancia y el camino óptimo actual también con su distancia.

4.3.2.2. Algoritmo de poda local

El algoritmo de poda local no presenta grandes cambios respecto del algoritmo exhaustivo: sin embargo, el rendimiento es completamente diferente entre uno y otro.

En este algoritmo, cada trabajador comienza una tarea sin una distancia mínima y en la función $RamificaciónyPoda()$ poda de acuerdo con una distancia mínima encontrada hasta el momento dentro de dicha tarea.

Esto no es ni más ni menos que preguntar si la distancia de la solución parcial que se tiene es menor a la distancia del camino óptimo encontrado en dicha tarea. Si la respuesta es afirmativa se continúa procesando como si nada en búsqueda de un camino mejor.

$$\text{Si distSoluciónParcial} < \text{distCaminoOptimo}$$

En el caso de que la respuesta fuera negativa se podará toda una rama del árbol de búsqueda. El Algoritmo 7 exhibe un seudocódigo de la función $RamificaciónyPoda()$

para el algoritmo de poda local resaltándose el cambio agregado con respecto al algoritmo exhaustivo.

Algoritmo 7 : Procedimiento RamificaciónyPoda en los algoritmos de poda local y global.

```
RamificaciónyPoda (ciudades, nroCiudades, nivel, soluciónParcial, distSoluciónParcial,
        caminoOptimo, distCaminoOptimo) {

  Si nivel = nroCiudades-1 {
    distSoluciónParcial += distancia (soluciónParcial[nivel], soluciónParcial[0])

    Si distSoluciónParcial <distCaminoOptimo {
      distCaminoOptimo = distSoluciónParcial
      caminoOptimo = soluciónParcial
    }
  } Sino {
    Para cada ciudad (ciu=0;ciu<nroCiudades;ciu++) {
      Si ciu no está en soluciónParcial {
        distSoluciónParcial += distancia (soluciónParcial[nivel], ciu)
        soluciónParcial[nivel+1] = ciu
        Si distSoluciónParcial <distCaminoOptimo {
          RamificaciónyPoda (ciudades, nroCiudades, nivel+1, soluciónParcial,
                    distSoluciónParcial, caminoOptimo, distCaminoOptimo)
          distSoluciónParcial -= distancia (soluciónParcial[nivel], ciu)
        }
      }
    }
  }
}
```

En el algoritmo de poda local existe una mejora importante en cuanto al tiempo de ejecución respecto del algoritmo exhaustivo pero en contrapartida se puede decir no es tan predecible debido a que la ciudad inicial y el orden de asignación de las tareas resultan determinantes.

4.3.2.3. Algoritmo de poda global

El algoritmo de poda global tampoco presenta grandes cambios respecto del algoritmo exhaustivo y del algoritmo de poda local, aunque el rendimiento es sobradamente mejor.

En este algoritmo, cada trabajador comienza una tarea con una distancia global mínima alcanzada hasta el momento y, en la función $RamificaciónyPoda()$, el trabajador poda siempre que sea posible.

Esto no es ni más ni menos que preguntar si la distancia de la solución parcial que se obtiene es menor a la distancia del camino óptimo encontrado en dicha tarea. Si la

respuesta es afirmativa se continúa procesando como si nada en búsqueda de un camino mejor.

Si distSoluciónParcial <distCaminoOptimo

En el caso de que la respuesta sea negativa se podará toda una rama del árbol de búsqueda. El Algoritmo 7 exhibe un seudocódigo de la función *RamificaciónyPoda*() para el algoritmo de poda global resaltándose el cambio agregado con respecto al algoritmo exhaustivo.

Asimismo, el procedimiento *Maestro*() debe ser modificado para permitir el envío de la distancia global mínima a cada proceso trabajador.

MPI_Send(trab, tarea, distCaminoOptimo)

A su vez, el procedimiento *Trabajador*() debe ser modificado para permitir la recepción de la distancia global mínima que le llega por parte del proceso maestro.

MPI_Recv(tarea, distCaminoOptimo)

El Algoritmo 8 exhibe un seudocódigo de las funciones *Maestro*() y *Trabajador*() para el algoritmo de poda global resaltándose el cambio agregado con respecto al algoritmo exhaustivo.

En el algoritmo de poda global el tiempo de ejecución mejora notablemente con respecto a los otros dos algoritmos y la predicción si bien depende de la ciudad inicial, por otro lado no se ve influenciada por factores externos como la asignación de tareas.

Algoritmo 8 : Procedimientos Maestro() y Trabajador() en el algoritmo de poda global.

```
Maestro() {
  LeerCiudades (ciudades, nroCiudades)
  .
  Para cada trabajador (trab = 1 ;trab <nodos; trab++) {
    MPI_Send(trab, nroCiudades, nivel, ciudades)
  }
  Inicializar caminoOptimo y distCaminoOptimo
  .
  Para cada trabajador (trab=1; trab <nodos; trab++) {
    MPI_Send(trab, tarea, distCaminoOptimo)
    tareasEnviadas++;
  }
  Mientras (tareasRecibidas<tareasEnviadas) {
    MPI_Recv(trab, soluciónParcial, distSoluciónParcial)
    tareasRecibidas++;
    .
    Si distSoluciónParcial <distCaminoOptimo {
      distCaminoOptimo = distSoluciónParcial
      caminoOptimo = soluciónParcial
    }
    .
    MPI_Send(trab, tarea, distCaminoOptimo)
    tareasEnviadas++;
  }
}

Trabajador() {
  MPI_Recv(nroCiudades, nivel, ciudades)
  MPI_Recv(tarea, distCaminoOptimo)
  Mientras reciba tareas {
    InicializarTiempos
    BuscarCaminoOptimo (ciudades, nroCiudades, nivel, caminoOptimo,
                distCaminoOptimo)
    FinalizarTiempos
    .
    MPI_Send(caminoOptimo, distCaminoOptimo)
    MPI_Recv(tarea, distCaminoOptimo)
  }
}
```

4.4. Definiendo la complejidad asintótica en tiempo

Los tres algoritmos paralelos TSP (el algoritmo exhaustivo, el algoritmo de poda local y el algoritmo de poda global) presentados en la sección 4.1 han sido desarrollados, estudiados y luego usados para mostrar la utilidad de la metodología práctica derivada en el capítulo 3.

Es indudable que el orden del tiempo de ejecución de los algoritmos de poda local y de poda global depende del número de ciudades (C), del número de procesadores (P) y de otros parámetros, $O(C, P, ?)$. Encontrar estos otros parámetros fue la clave para obtener una buena predicción de rendimiento. En este sentido, se aplicó la ciencia computacional para tomar ventaja del hardware y de ciertas técnicas matemáticas existentes. Gracias a esta ciencia se hizo posible el análisis masivo de datos almacenados en una base histórica.

Al comienzo de la investigación, el paso de la metodología donde más se trabajó fue el proceso KDD. También puede decirse que, en consecuencia, fue el paso más útil y provechoso. Un gran volumen de datos fue procesado con el único fin de encontrar alguna propiedad, característica, relación y/o patrón común entre esos datos. La intuición guió unas cuantas pruebas que permitieron hallar unos cuantos hechos.

Por otra parte como se sabe, la complejidad asintótica en tiempo para el algoritmo exhaustivo sólo depende del número de ciudades (C) y de la cantidad de procesadores (P), $O(C, P)$. Unas simples ejecuciones corroboraron dicha hipótesis. Este algoritmo es de suma importancia ya que se usa como testigo para mostrar situaciones donde su costo es prohibitivo en términos de tiempo (aún para valores de C pequeños).

A continuación, se describen los pasos (las hipótesis) que permitieron descubrir los parámetros relevantes para definir la complejidad asintótica en tiempo de los algoritmos de poda local y de poda global. Previamente se da la especificación del cluster utilizado para llevar a cabo toda la experimentación.

Especificación del cluster de PCs

Las ejecuciones fueron obtenidas en un Cluster de PCs homogéneo de 32 nodos (Pentium IV 3.0GHz., 1Gb DDR-DSRAM 400Mhz., Gigabit Ethernet) en el Departamento de Arquitectura de Computadores y Sistemas Operativos (DACSO-CAOS) de la Universidad Autónoma de Barcelona. Todas las comunicaciones han sido efectuadas usando una red conmutada con una distancia media de dos saltos entre dos puntos extremos de comunicación. Los conmutadores permiten rutas dinámicas con el fin de solapar las comunicaciones

4.4.1. Hipótesis → localización de las ciudades

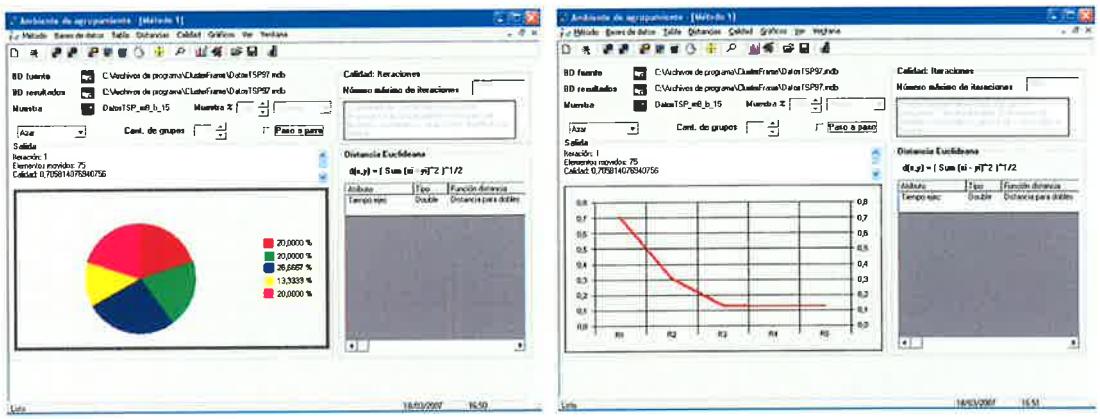
Hay que recordar que la formulación de las hipótesis es la primera etapa en la fase de extracción del conocimiento de la metodología, ver Figura 4.5.

fue realizada al azar y la función objetivo elegida fue el error cuadrático (indicador de la distancia de los n datos a los respectivos centroides de los clusters), Ecuación 4.1.

$$\sum_{j=1}^{k}\sum_{i=1}^{n}|x_i^{(j)} - c_j|^2 \tag{4.1}$$

donde $|x_i^{(j)} - c_j|^2$ es la medida de distancia elegida entre un dato $x_i^{(j)}$ y el centroide del cluster c_j.

La Figura 4.7(a) exhibe las muestras obtenidas por cluster y la Figura 4.7(b) la evolución de la compactación en el ambiente Cluster-Frame. Para más información sobre este ambiente de clustering referirse al apéndice B.



(a) Muestras obtenidas por cluster (b) Evolución de la compactación

Figura 4.7: Ambiente Cluster-Frame.

En la Tabla 4.3 se indica el cluster (Cl) asignado según k-means para cada muestra. Para los clusters 1 a 5, los centroides obtenidos fueron 558.78 seg., 243.33 seg., 112.87 seg., 80.29 seg. y 151.90 seg., respectivamente. Como se puede apreciar había una correspondencia entre los centroides y los valores de las medias de cada patrón gráfico.

La validación de las hipótesis planteadas en un comienzo implica el cotejo de los resultados obtenidos con suposiciones conocidas, verificándose que las conclusiones que se arrojan sean válidas y suficientemente satisfactorias.

Los tiempos de ejecución fueron agrupados de manera similar a los patrones gráficos fijados al comienzo, la capacidad de predicción fue de 89 %. Esto significó que para el algoritmo de poda local había una estrecha relación entre los patrones y los tiempos de ejecución. En consecuencia, se asoció cada patrón gráfico con un promedio de tiempo de ejecución que era el centroide y se almacenó esta información en la base histórica. Esta base histórica es la que se trabaja en la etapa de comparación de datos de la fase de predicción.

Todo lo antes expresado para el algoritmo de poda local, se repitió para el algoritmo de poda global.

La Tabla 4.4 muestra los tiempos de ejecución por patrón gráfico correspondiente al algoritmo de poda global usando 8 nodos. Un comentario inmediato que se derivó de estas columnas fue que los tiempos pertenecientes a un patrón gráfico (PD) eran bastante similares excepto para algunos casos.

Nuevamente se eligió la técnica de clustering y dentro de ésta se aplicó el algoritmo k-means conservándose los parámetros prefijados. Nuevamente, en la Tabla 4.4 se indica el cluster asignado según k-means para cada muestra. Para los clusters 1 a 5, los centroides obtenidos fueron 92.22 seg., 16.94 seg., 37.17 seg., 10.19 seg. y 7.94 seg., respectivamente.

Se detectó una correspondencia menor entre los valores de los centroides y los valores de las medias del tiempo de ejecución por patrón gráfico. No obstante, los tiempos de ejecución fueron agrupados de manera similar a los patrones fijados al comienzo. En esta ocasión, la capacidad de predicción para el algoritmo de poda global fue de casi 75 %.

4.4.1.2. Análisis y valoración

Dado un experimento siempre se cumple que, el tiempo de ejecución para el algoritmo de poda global es menor que el tiempo de ejecución para el algoritmo de poda local e insignificante respecto del valor que se obtiene con el algoritmo exhaustivo. Hay que recordar que los tres algoritmos trabajan para alcanzar un mismo fin, obtener el camino mínimo exacto. Sin embargo, el rendimiento observado en cada uno de los algoritmos es completamente diferente a los otros.

Los tiempos de ejecución para el algoritmo exhaustivo fueron casi idénticos; la desviación media fue inferior al 3 % en todos los casos.

La hipótesis inicial para los algoritmos de poda local y de poda global fue corroborada pero quedó un tema abierto para discutir y resolver: el por qué razón aparecían tiempos singulares en cada uno de los patrones gráficos.

La complejidad asintótica en tiempo para el algoritmo de poda local y para el algoritmo de poda global fue definida como $O(C, P, PD)$. A su vez, como los patrones no afectaban el tiempo final de ejecución en el algoritmo exhaustivo la complejidad fue fijada como $O(C, P)$.

4.4.2. Hipótesis → localización de las ciudades y ciudad de comienzo

Las hipótesis trabajadas fueron que la localización de las ciudades (PD) y la ciudad inicial (C_1) afectaban directamente los tiempos de ejecución. Con esta combinación de variables, la predicción mejoró significativamente. En esta ocasión, la técnica de minería de datos analizaba, en forma conjunta, los tiempos de ejecución y la ciudad inicial para generar los clusters [63].

4.4.2.1. Experimentación

Se continuó el análisis con el mismo conjunto de entrenamiento trabajado y definido en la sección 4.4.1.1. Los gráficos y tablas fueron analizados nuevamente por completo. Comparándose la Figura 4.6 con la Tabla 4.4 fue fácil inferir algunos hechos importantes.

En la Figura 4.6(a), las dos ciudades alejadas se corresponden con los dos valores más altos de la Tabla 4.4(Grupo1); la Figura 4.8 exhibe esta relación. Estas dos ciudades alejadas, concretamente (24,15) y (14,17), son en definitiva las ciudades iniciales de la muestra de datos 1 y 2 respectivamente.

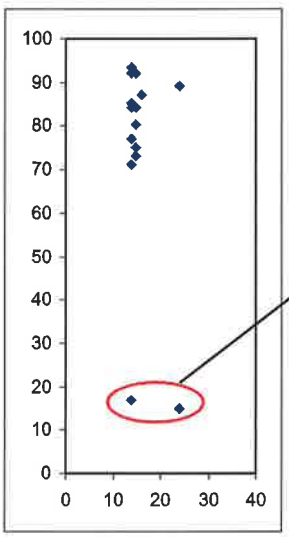
Las cuatro ciudades alejadas en la Figura 4.6(b) se corresponden con los cuatro valores más altos de la Tabla 4.4(Grupo2). Nuevamente, las ciudades alejadas son en definitiva las ciudades de inicio que generan los mayores tiempos.

Las seis ciudades alejadas en la Figura 4.6(c) se corresponden con los seis valores más altos de la Tabla 4.4(Grupo3).

Se deduce una regla: si las ciudades alejada son consideradas ciudades de inicio del tour, los tiempos de ejecución que se tendrán serán tiempos elevados respecto de los restantes tiempos.

Las ciudades en la Figura 4.6(d) están distribuidas entre dos sectores de modo tal que los tiempos de ejecución se asemejan, ver Tabla 4.4(Grupo4).

Finalmente, las ciudades en la Figura 4.6(d) están bastante agrupadas por lo cual los tiempos de ejecución nuevamente son bastante similares, ver Tabla 4.4(Grupo5).



Muestra	Grupo 1
1	216.17
2	214.44
3	77.25
4	72.64
5	70.94
6	74.21
7	75.59
8	73.72
9	69.47
10	74.96
11	75.89
12	70.17
13	73.73
14	70.87
15	73.30
Media	92.23

Figura 4.8: Correspondencia entre ciudades distantes y tiempos de ejecución.

Otra observación importante que surgió de la Tabla 4.4 fue que la media de los tiempos de ejecución por grupo decrecía a medida que las ciudades se aproximaban.

4.4.2.2. Análisis y valoración

Las hipótesis fueron corroboradas. Sin ninguna duda, la ciudad inicial y la localización de las ciudades jugaban un papel importante en los tiempos de ejecución. Igualmente, como en la primera aproximación, quedó un tema abierto para discutir y resolver: la relación entre la proximidad de las ciudades y los tiempos de ejecución.

La complejidad asintótica en tiempo tanto para el algoritmo de poda local como para el algoritmo de poda global fue redefinida como $O(C, P, PD, C_1)$.

Asimismo, se decidió dejar a un lado el estudio del algoritmo de poda local por su nula utilidad práctica. El exhaustivo ha servido de referencia para verificar la corrección de resultados del algoritmo, pero el de poda local al tener un tiempo de ejecución muy superior al de poda global no tiene ningún interés real.

4.4.3. Hipótesis → suma de distancias y desviación media

Al llegar a este punto, uno se formula la siguiente cuestión: ¿sería posible establecer una caracterización numérica del parámetro topológico (la distribución espacial), una especie de índice que evalúe los invariantes propios de cada distribución espacial, de manera que, por un lado se simplifique el proceso de predicción al eliminar la evaluación de similitud y por otro se consiga un índice numérico simple en la caracterización de las ciudades?

Una vez más el análisis de resultados permite establecer una nueva hipótesis, esta vez basada en los conceptos de suma de las distancias de la ciudad inicial a las restantes ciudades (SD, definida como se muestra en Ecuación 4.2) y de la desviación media de estos valores $MDSD$, como alternativa a la distribución espacial.

$$\forall j : 1 \leq j \leq C \qquad SD(C_j) = \sum_{i=1, j \neq i}^{C} \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{4.2}$$

Esta hipótesis representa un cambio cualitativo muy significativo. De trabajar con aproximaciones basadas en patrones gráficos y su comparación (con todo lo que esto implica de nivel de incertidumbre), se pasa a un modelo de valoración / identificación numérica de los datos de entrada, de cara a hacer la predicción.

La suma de las distancias de la ciudad inicial a las restantes ciudades es un índice útil para la predicción del tiempo de ejecución; éste refleja numéricamente en cierta medida el patrón gráfico. Por otro lado, la desviación media de las sumas de las distancias indica cuán separadas están las ciudades entre sí. En definitiva la suma de las distancias y el desvío de estas sumas permite estimar el tiempo de ejecución.

4.4.3.1. Experimentación

Si una ciudad en particular está muy alejada de las otras ciudades, el valor de su SD será considerablemente mayor que el resto y, en consecuencia, el tiempo de ejecución también

$$y' = y + \triangle y \tag{5.2}$$

donde x e y son las coordenadas originales y x' e y' las nuevas coordenadas.

Escalamiento: Las coordenadas de una ciudad escalada por un factor S_x en la dirección x y en la dirección y (la ciudad es escalada en tamaño cuando S_x es mayor a uno y reducida en tamaño cuando S_x está entre cero y uno) se puede expresar como

$$x' = xS_x \tag{5.3}$$

$$y' = yS_x \tag{5.4}$$

Rotación: Las coordenadas de una ciudad rotada a través de un ángulo θ sobre el origen del sistema de coordenadas se puede expresar como

$$x' = x\cos\theta + y\sin\theta \tag{5.5}$$

$$y' = -x\sin\theta + y\cos\theta \tag{5.6}$$

De acuerdo con la teoría de reconocimiento de patrones geométricos (que se verá en detalle en la sección 5.3), la similitud calculada aplicando la distancia Hausdorff como una función de la posición relativa entre un patrón conocido y un nuevo conjunto de ciudades, que en este caso se trata de un patrón conocido trasladado, escalado o rotado, es del 100 %. Por lo tanto, el orden de predicción de rendimiento para el nuevo conjunto de ciudades será el tiempo de ejecución del patrón conocido.

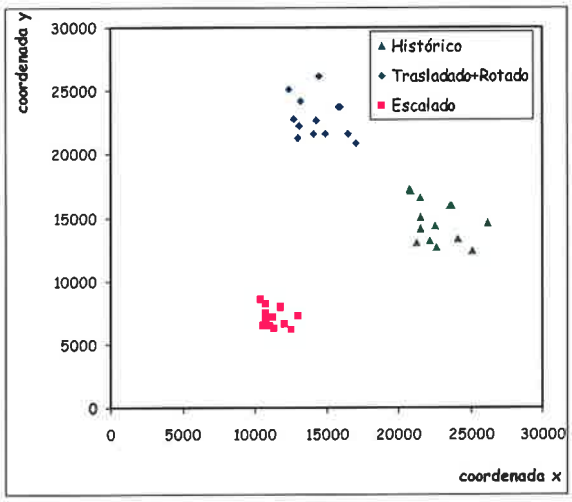


Figura 5.1: Un patrón histórico, el mismo patrón histórico trasladado más rotado y el mismo patrón histórico escalado, consistiendo de 15 ciudades cada uno.

Un conjunto de entrenamiento consistente en 15 ciudades fue elegido de la base histórica. A partir de este conjunto se obtuvo un experimento que evaluó la transformación de traslación más rotación y otro que analizó el escalado. Las transformaciones de traslación

más rotación se obtuvieron intercambiando la coordenada x por la coordenada y y el escalamiento dividiendo por dos ambas coordenadas. La Fig 5.1 muestra el patrón histórico estudiado, el mismo patrón histórico trasladado y rotado y el mismo patrón histórico escalado. Se fijó el nivel en 5 y se ejecutaron los algoritmos exhaustivo y de poda global usando 8, 16 y 32 nodos.

La Tabla 5.1 exhibe los tiempos de ejecución para el algoritmo exhaustivo y para el algoritmo de poda global, por transformación y por muestra (ciudad de comienzo) habiendo usado 32 nodos. Como se puede observar, los tiempos de ejecución por algoritmo y muestra resultaron ser bastante similares para las dos transformaciones. Esto era lo esperado. La desviación media (MD) por algoritmo y muestra fue inferior al 2 % para todos los casos.

Tabla 5.1: Comparativa de los tiempos de ejecución (en seg.) del patrón histórico estudiado [PHist], del mismo patrón histórico rotado más trasladado [TrasRot] y del mismo patrón histórico escalado [Esc] usando 32 nodos. También se exhibe la desviación media (MD).

Algoritmo exhaustivo				**Algoritmo de poda global**			
PHist	**TrasRot**	**Esc**	**MD**	**PHist**	**TrasRot**	**Esc**	**MD**
596.43	621.54	612.50	9.15	13.18	13.24	13.34	0.06
596.86	628.23	612.48	10.47	13.32	13.29	13.29	0.01
595.83	628.76	611.97	11.05	13.20	13.09	13.17	0.04
597.36	628.34	612.09	10.50	13.18	13.12	13.15	0.02
597.09	627.72	613.19	10.39	13.12	13.22	13.12	0.04
596.29	630.15	614.54	11.58	13.13	13.20	13.25	0.04
596.69	628.72	610.92	11.07	13.27	13.20	13.26	0.03
596.69	628.84	612.05	10.88	13.10	13.02	13.22	0.07
611.56	629.73	613.74	7.59	13.27	13.17	13.16	0.05
611.20	628.76	612.14	7.59	13.28	13.19	13.25	0.03
619.52	627.65	612.74	5.12	13.19	13.18	13.23	0.02
619.52	628.54	611.84	5.71	13.17	13.19	13.23	0.02
619.37	629.19	613.30	5.71	13.21	13.23	13.25	0.01
618.71	629.11	612.57	5.99	13.27	13.29	13.30	0.01
618.44	629.22	613.14	5.97	13.32	13.24	13.35	0.04

5.1.2. La circunferencia

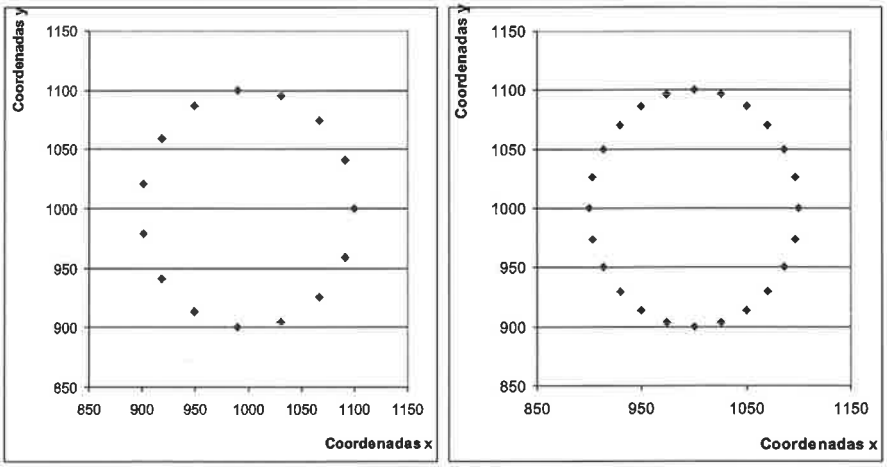
Un caso sintético es tener las ciudades uniformemente distribuidas en una circunferencia. Como la suma de las distancias de una ciudad a las restantes es siempre la misma y, en consecuencia, el valor de $MDSD$ es cero (o cercano a cero por problemas de redondeo), los tiempos de ejecución que se esperan han de ser muy similares entre sí.

Se crearon experimentos compuestos por 15 a 24 ciudades. Las fórmulas utilizadas para generar las coordenadas (x, y) fueron las siguientes

$$\forall i : 1 \leq i \leq C \qquad x_i = (\cos(radianes((360/C) * (i-1))) + 10) * 100 \tag{5.7}$$

$$\forall i : 1 \leq i \leq C \qquad y_i = (\sin(radianes((360/C) * (i-1))) + 10) * 100 \tag{5.8}$$

Los experimentos compuestos por 15 a 17 ciudades sirvieron de entrada tanto para el algoritmo exhaustivo como para el algoritmo de poda global, mientras que los experimentos compuestos de 18 a 24 ciudades sólo sirvieron de entrada para el algoritmo de poda global. La Figura 5.2(a) y la Figura 5.2(b) exhiben los gráficos correspondientes a 15 y 24 ciudades uniformemente distribuidas en una circunferencia, respectivamente.



(a) 15 ciudades (b) 24 ciudades

Figura 5.2: Ciudades uniformemente distribuidas en una circunferencia.

La Tabla 5.2 exhibe la media y la desviación media de los tiempos de ejecución teniendo en cuenta de 15 a 17 ciudades uniformemente distribuidas en una circunferencia para el algoritmo exhaustivo usando 32 nodos. Los tiempos medios de ejecución expresados para las ciudades 18, 19 y 20 son estimaciones que surgen de la siguiente fórmula

$$TiempodeEjecucion(C) = TiempodeEjecucion(C-1) * C \tag{5.9}$$

Por otro lado, la Tabla 5.3 muestra la media y la desviación media de los tiempos de ejecución teniendo en cuenta de 15 a 24 ciudades para el algoritmo de poda global usando 32 nodos.

Analizando la información suministrada del algoritmo exhaustivo, los tiempos de ejecución crecen de acuerdo con el número de ciudades según la Ecuación (5.9). Sin embargo, para el algoritmo de poda global esta relación no es válida; sólo se puede apreciar un mínimo

Tabla 5.2: Media y desviación media de los tiempos de ejecución (en seg.) para el algoritmo exhaustivo de acuerdo con el número de ciudades trabajadas usando 32 nodos.

	Algoritmo exhaustivo					
# ciudades	**15**	**16**	**17**	**18**	**19**	**20**
Media	597.48	8924.95	143153.88 $\simeq$ 39 horas	2425985 $\simeq$ 28 días (Eq. 5.9)	43667742 $\simeq$ 505 días (Eq. 5.9)	829687100 $\simeq$ 26 años (Eq. 5.9)
Desviación media	1.29	13.72	270.60			

Tabla 5.3: Media y desviación media de los tiempos de ejecución (en seg.) para el algoritmo de poda global de acuerdo con el número de ciudades trabajadas usando 32 nodos.

Algoritmo de poda global					
Número de ciudades	**15**	**16**	**17**	**18**	**19**
Media	12.71	17.47	23.42	31.87	42.95
Desviación media	0.03	0.04	0.08	0.08	0.07
Suma de distancias	1900.33	2024.70	2154.22	2284.76	2417.06

Algoritmo de poda global					
Número de ciudades	**20**	**21**	**22**	**23**	**24**
Media	54.94	68.67	129.53	367.29	1085.57
Desviación media	0.10	0.10	0.11	0.30	2.12
Suma de distancias	2541.16	2670.35	2796.94	2922.83	3062.71

incremento en los tiempos. La desviación media para todos los casos ha sido inferior al 1 %.

Por otro lado, con respecto al algoritmo exhaustivo importa observar el problema de su costo prohibitivo en términos de tiempo a medida que el número de ciudades crece. A partir de 17 ciudades todo cálculo deja de tener sentido. En contrapartida, el algoritmo de poda global solamente al trabajar con 24 ciudades comienza a tener un tiempo de ejecución significativo.

5.1.3. La alineación

Otro caso sintético se presenta al tener todas las ciudades alineadas separadas entre sí por un valor constante. A priori, se podría decir que los tiempos de ejecución tendrán un valor máximo para las ciudades extremas y éstos irán decreciendo a medida que se va hacia la ciudad central (o las ciudades centrales). La Figura 5.3 exhibe lo expresado.

Se construyeron experimentos compuestos por 15 a 20 ciudades, los cuales sirvieron de

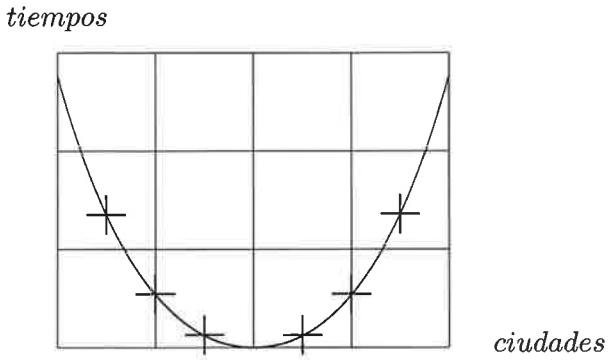


Figura 5.3: Tendencia de los tiempos de ejecución (en seg.) para ciudades alineadas.

entrada sólo para el algoritmo de poda global. Las coordenadas de las ciudades (x, y) se eligieron siguiendo la fórmula

$$\forall i : 1 \leq i \leq C \qquad (x, y) = ((i-1) * 400, 0) \tag{5.10}$$

La Figura 5.4(a) y la Figura 5.4(b) exhiben los gráficos correspondientes a 15 y 24 ciudades alineadas separadas entre sí por un valor constante (400), respectivamente.

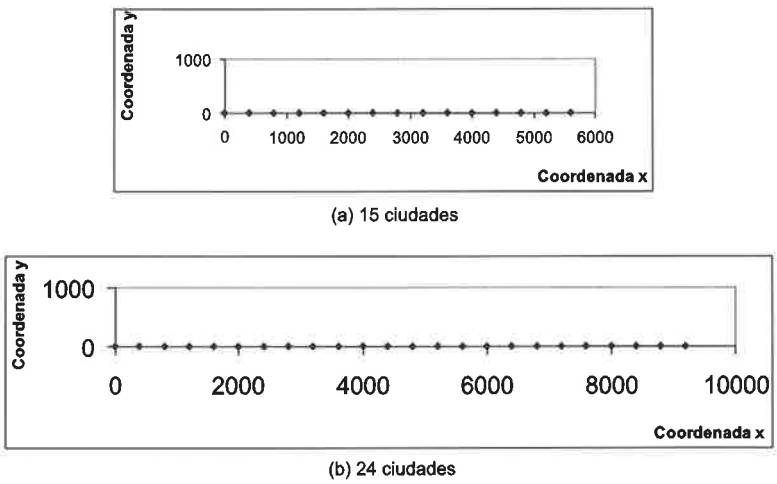


Figura 5.4: Ciudades alineadas separadas entre sí por un valor constante.

Si se realiza un simple análisis de la ciudad que se encuentra más a la derecha (ciudad con mayor valor en la ordenada) para continuar con las sucesivas ciudades hacia la izquierda, se observará que SD decrece a medida que se va hacia el centro para luego volver a crecer hacia el extremo izquierdo. La Ecuación 4.2 puede volverse a escribir como la Ecuación 5.11

$$\forall j : 1 \leq j \leq C \qquad SD(C_j) = \sum_{i=1, j \neq i}^{C} \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} = \sum_{i=1, j \neq i}^{C} \mid x_j - x_i \mid \tag{5.11}$$

Entonces, los tiempos de ejecución para el algoritmo de poda global debieran mostrar un comportamiento análogo; en este escenario se puede apreciar cómo la ciudad inicial es determinante.

Las Tablas 5.4(a) y 5.4(b) exhiben las coordernadas de la ciudad inicial, la suma de las distancias (SD) de la ciudad inicial a las restantes ciudades y el tiempo de ejecución del algoritmo de poda global usando 32 nodos para 18 y 19 ciudades respectivamente. Debe apreciarse en las tablas, que el comportamiento descrito anteriormente se cumple; los tiempos decrecen a medida que la ciudad inicial se va acercando a la ciudad(es) central(es). Cabe observar la simetría que se presenta tanto en la suma de las distancias como en los tiempos de ejecución producto del caso singular analizado.

Tabla 5.4: Coordenadas de la ciudad inicial, suma de las distancias y tiempo de ejecución (en seg.) del algoritmo de poda global para (a) 18 (b) 19 ciudades usando 32 nodos.

(a) 18 ciudades

x	y	SD	Tiempo de ejecución
6800	0	61200	67.92
6400	0	54800	61.96
6000	0	49200	59.57
5600	0	44400	54.82
5200	0	40400	54.12
4800	0	37200	51.13
4400	0	34800	51.05
4000	0	33200	49.43
3600	0	32400	49.32
3200	0	32400	48.85
2800	0	33200	48.40
2400	0	34800	49.59
2000	0	37200	51.84
1600	0	40400	52.63
1200	0	44400	53.03
800	0	49200	55.60
400	0	54800	58.91
0	0	61200	61.68

(b) 19 ciudades

x	y	SD	Tiempo de ejecución
7200	0	68400	176.39
6800	0	61600	173.03
6400	0	55600	158.46
6000	0	50400	144.14
5600	0	46000	133.00
5200	0	42400	129.53
4800	0	39600	126.41
4400	0	37600	129.66
4000	0	36400	131.63
3600	0	36000	132.23
3200	0	36400	130.46
2800	0	37600	134.60
2400	0	39600	133.30
2000	0	42400	138.39
1600	0	46000	141.28
1200	0	50400	146.95
800	0	55600	153.31
400	0	61600	170.59
0	0	68400	179.61

La Figura 5.5 exhibe los tiempos de ejecución (en seg.) del algoritmo de poda global para 20 ciudades usando 32 nodos.

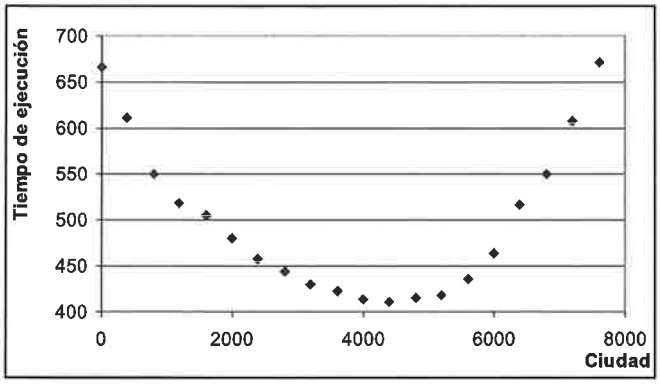


Figura 5.5: Tiempos de ejecución (en seg.) para el algoritmo de poda global consistente en 20 ciudades alineadas usando 32 nodos.

5.1.4. Análisis y valoración

Los experimentos sintéticos relacionados con las transformaciones geométricas, la circunferencia y la alineación han servido para corroborar intuiciones en cuanto a los tiempos de ejecución para el algoritmo de poda global.

Los tiempos de ejecución permanecen prácticamente iguales aún si los patrones gráficos sufren transformaciones como rotaciones, traslaciones o escalado.

Para el caso de las ciudades uniformemente distribuidas en una circunferencia, la SD es siempre la misma y, en consecuencia, la desviación media de estas sumas resulta ser cero siendo los tiempos de ejecución prácticamente iguales.

En cuanto a tener todas las ciudades alineadas separadas entre sí por un valor constante, si se comienza un análisis por el extremo derecho, el valor SD decrece a medida que se va hacia el centro para luego volver a crecer cuando se va hacia el extremo izquierdo. En consecuencia, lo mismo ocurre con los tiempos de ejecución.

5.2. Escalabilidad

El objetivo en esta sección es verificar que los algoritmos estudiados escalan en cuanto al número de procesadores y en cuanto al tamaño del problema.

Validar que los algoritmos escalan correctamente al aumentar el número de procesadores es interesante para saber hasta qué punto se mejora el rendimiento agregando máquinas. Comprobar que los algoritmos escalan correctamente al aumentar el tamaño del problema, es decir el número de ciudades (C), conservándose un patrón gráfico similar e igual número de procesadores (P), es imprescindible para tener una noción del tiempo que puede emplear la resolución del problema

5.2.1. Escalabilidad en cuanto al número de procesadores

Un conjunto integrado por 16 ciudades cualesquiera es elegido para ejecutar el algoritmo de poda global usando 8, 16 y 32 nodos del cluster de PCs. La Figura 5.6 exhibe el patrón gráfico (la distribución) de las 16 ciudades.

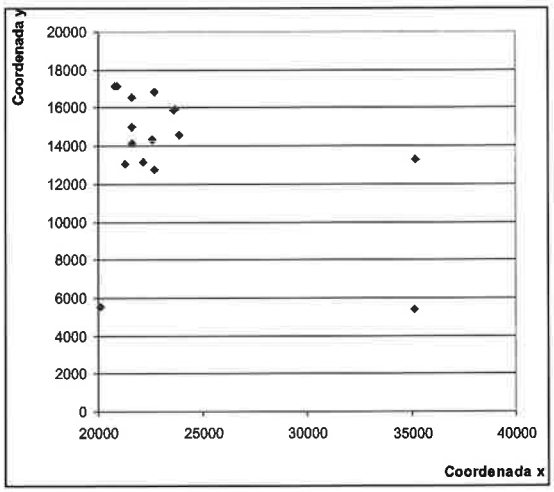


Figura 5.6: Patrón gráfico para las 16 ciudades estudiadas.

Por lo expresado en la sección 4.4.3 y observando el dibujo, se puede inferir que el valor $MDSD$ será de un tamaño considerable y en consecuencia los tiempos también debieran ser significativos. Esto refuerza el valor de este experimento ya que se podrá observar la escalabilidad sin la intromisión de la comunicación.

La Figura 5.7 muestra los tiempos de ejecución (en seg.) obtenidos de acuerdo con los parámetros enunciados y considerando cada una de las ciudades en algún momento como ciudad inicial.

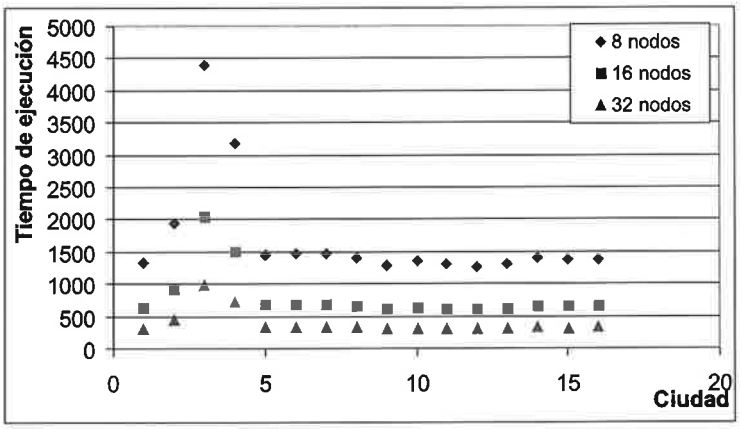


Figura 5.7: Escalabilidad en cuento al número de procesadores para el algoritmo de poda global manteniendo constante 16 ciudades.

Como se puede observar, los tiempos de ejecución decrecen en forma proporcional a medida que el número de máquinas aumenta. La escalabilidad quedó demostrada.

5.2.2. Escalabilidad al aumentarse el tamaño del problema

Dos conjuntos compuestos por 15 y 16 ciudades, donde $conjunto15 \subset conjunto16$, son elegidos para ejecutar el algoritmo de poda global usando 8, 16 y 32 nodos del cluster de PCs. La Figura 5.8 exhibe los patrones gráficos de las 15 y 16 ciudades resaltándose la ciudad que sólo pertenece al conjunto integrado por 16 ciudades.

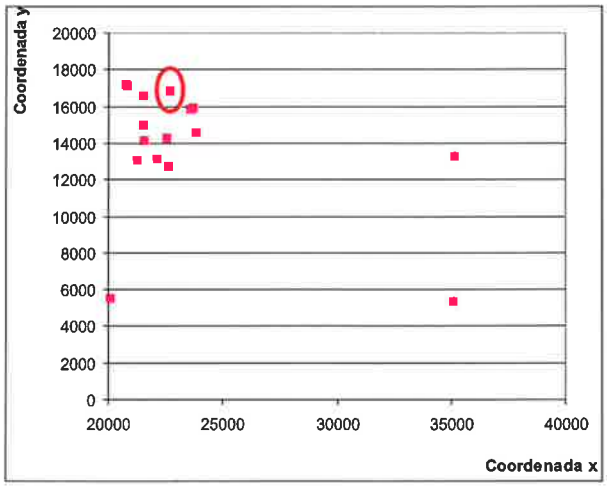


Figura 5.8: Patrón gráfico para las 15 y 16 ciudades estudiadas.

La Figura 5.9 muestra los tiempos de ejecución (en seg.) obtenidos para el algoritmo de poda global de acuerdo con los parámetros enunciados y considerando cada una de las ciudades en algún momento como ciudad inicial. Hay que notar que la ciudad 10 sólo pertenece al conjunto integrado por 16 ciudades.

5.2.3. Análisis y valoración

La escalabilidad de arquitectura o hardware permite a un sistema incrementar el uso de procesadores para obtener un mejor rendimiento. La escalabilidad algorítmica permite saber si un algoritmo paralelo puede trabajar con más datos, es decir, si se pueden incrementar los pasos computacionales en una cantidad razonable.

En los algoritmos analizados, el tiempo de ejecución es inversamente proporcional al número de procesadores y, a su vez, es proporcional al número de ciudades como era de esperarse. La comunicación no pesa en lo más mínimo.

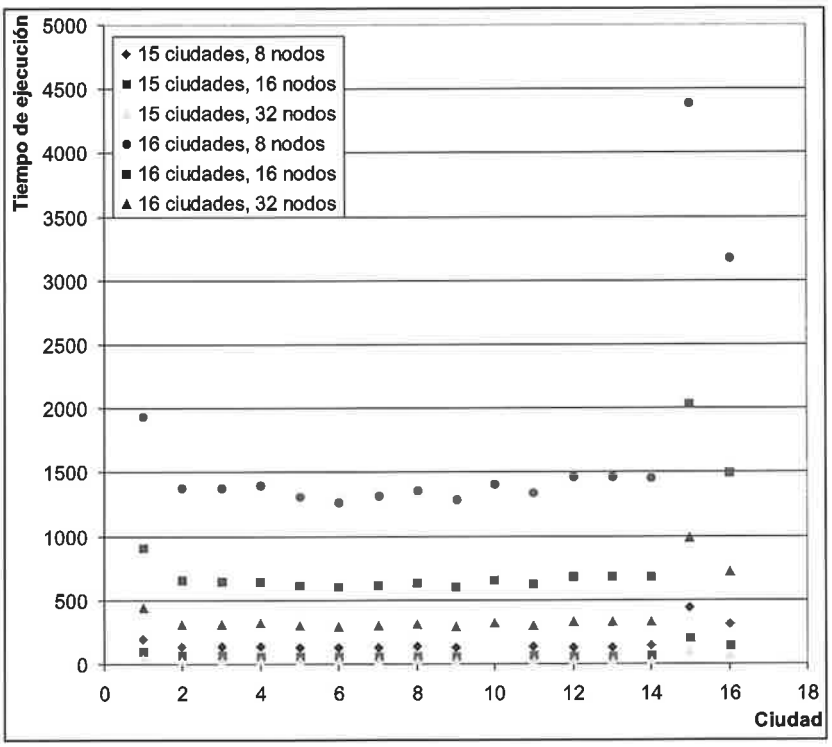


Figura 5.9: Aumento del tamaño del problema para el algoritmo de poda global usando 8, 16 y 32 nodos.

5.3. Predicción

La fase de predicción a lo largo de la investigación ha sufrido un cambio muy significativo. De aproximaciones basadas en patrones gráficos y su comparación (con lo que esto implica de nivel de incertidumbre), a un modelo de valoración / identificación numérica de los datos de entrada, de cara a hacer la predicción.

a partir de la experiencia y de los trabajos previos

5.3.1. Una primera aproximación

El modelo de predicción que se ha desarrollado hasta este momento, considera como parámetros de entrada el número de ciudades, la distribución espacial o patrón de las mismas y la ciudad de origen. Al incluir un elemento de características topológicas (la distribución espacial), el esquema de predicción incluye un paso de comparación de similitud del patrón de entrada con respecto a los patrones disponibles.

La etapa de comparación de datos para el problema TSP descansa en la geometría de reconocimiento de patrones (geometric pattern matching - GPM). A continuación se ex-

ponen ciertos conceptos imprescindibles sobre GPM antes de explicarse cómo se obtiene el orden de predicción de rendimiento para un nuevo conjunto de ciudades.

Geometría de reconocimiento de patrones

La geometría de reconocimiento de patrones (geometric pattern matching - GPM) es un problema computacional fundamental. La GMP tiene numerosas aplicaciones en diversas áreas de ciencias de la computación, tales como visión por computador, compresión de imagen o vídeo, reconocimiento de objetos y química computacional [64].

La mayoría de los estudios de GPM se focalizan sobre métodos del campo de la geometría computacional [65, 66, 67, 64]. Para resolver el problema de geometría de reconocimiento hay diversas variantes, diferentes enfoques y varias técnicas algorítmicas que dependen de la naturaleza de los datos de entrada, del grupo de transformaciones permitidas y de la distancia empleada. En particular, la distancia de Hausdorff con movimiento es la que se utiliza para el problema TSP.

Distancia de Hausdorff

Para comparar puntos (ciudades) bajo la distancia de Hausdorff como una función de traslación y movimiento rígido existen muchos trabajos [68]. Debido a que la complejidad computacional puede ser grande en algunas ocasiones se consideran algoritmos que trabajan con aproximaciones [69].

La distancia Hausdorff es la distancia máxima de un conjunto al punto más cercano en otro conjunto. Formalmente, la distancia *Hausdorff dirigida* de un conjunto A a otro conjunto B es la función máxima, definida como

$$h(A,B) = \operatorname*{máx}_{\{a \in A\}} \{ \operatorname*{mín}_{\{b \in B\}} |a-b| \} \tag{5.12}$$

donde a y b son puntos de los conjuntos A y B respectivamente, y $|a-b|$ es la norma euclideana entre estos puntos.

Debe notarse que la distancia Hausdorff es orientada: esto significa que muchas veces $h(A,B)$ no es igual a $h(B,A)$. Esta asimetría es una propiedad de las funciones maximum, mientras que las funciones minimum son simétricas. Una definición más general de la distancia Hausdorff entre A y B sería

$$H(A,B) = \text{máx}\{h(A,B), h(B,A)\} \tag{5.13}$$

En consecuencia, se puede decir que la distancia Hausdorff calcula el grado de similitud entre dos conjuntos de puntos, midiendo la distancia del punto de A que está más alejado de cualquier punto de B y viceversa. Estos dos conjuntos están en posiciones fijas con respecto uno del otro.

Distancia de Hausdorff con movimiento

En general hay que ser capaz de medir la similitud entre todas las posiciones relativas de dos conjuntos y esto se realiza calculando la distancia Hausdorff como una función de la posición relativa [68, 70].

Para cualquier grupo G, la distancia mínima Hausdorff bajo movimiento euclideano se define como

$$M_G(A, B) = \min_{g_1, g_2 \in G} H(g_1 A, g_2 B) \tag{5.14}$$

Si el grupo G es tal que, para cualquier $g \in G$ y para dos puntos cualesquiera x_1, x_2 se cumple $|gx_1 - gx_2| = |x_1 - x_2|$, entonces es posible considerar sólo la transformación de uno de los conjuntos

$$M_G(A, B) = \min_{g \in G} H(A, gB) \tag{5.15}$$

Esta propiedad se tiene cuando G es el grupo de traslaciones y $|.|$ es una norma y también cuando G es el grupo de movimientos rígidos y $|.|$ es la norma euclideana.

5.3.1.1. Experimentación

Con el objeto de validar la metodología de predicción y su robustez, se modificaron sólo unas pocas coordenadas de ciertas ciudades del conjunto de entrenamiento definido y trabajo en la sección 4.4.1.1. Este nuevo conjunto de datos, *conjunto de testeo*, fue usado para calcular el error de las hipótesis construidas con los datos de entrenamiento. Hay que destacar que las hipótesis fueron evolucionando por lo cual este análisis fue repetido una y otra vez.

Para el algoritmo exhaustivo la predicción fue directa y exacta. Comparando cada patrón gráfico del *conjunto de testeo* con los patrones almacenados en la base histórica y teniendo en cuenta la ciudad inicial, fue posible hacer predicciones del tiempo de ejecución para los algoritmos de poda local y de poda global. La geometría de reconocimiento de patrones bajo movimiento euclideano permite comparar dos conjuntos de datos otorgando una medida de similitud que varía entre 0 y 1. El tiempo del patrón histórico correspondiente al par con mayor valor de similitud resulta ser la predicción de rendimiento para el *conjunto de testeo*.

Para analizar cuán buena es la estimación, al *conjunto de testeo* se lo somete a todos los pasos de la metodología. Los tiempos de ejecución obtenidos resultaron ser muy similares (en el orden del 90 %) a los otorgados por el patrón histórico.

El Algoritmo 9 (Algoritmo 2 redefinido) describe el procedimiento que se sigue; donde SA es la similitud aceptable, $C_1, ..., C_C$ son las ciudades del nuevo conjunto, d va almacenando la máxima medida de similitud encontrada o en su defecto se inicializa con la similitud aceptable, BDH es la base de datos histórica, BDH_i es la instancia i de la base de datos histórica y max es el índice de la instancia en BHD que tiene la máxima medida de similitud de la base de datos histórica.

Algoritmo 9 : Similitud aceptable usando Hausdorff.

```
Similitud_Aceptable (C_1, ..., C_C, BDH, SA) {
    d = SA
    max = -1
    Para cada uno de los datos historicos BDH_i de BHD {
        aux = DistHausdorffPosicRelativa(C_1, ..., C_C, BDH_i)
        Si aux > d {
            d = aux
            max = i
        }
    }
    Si max = -1
        Retornar -1
    Sino
        Retornar Orden(BDH_max)
}
```

5.3.1.2. Análisis y valoración

La geometría de reconocimiento de patrones y, en particular, la distancia Hausdorff con movimiento permite dar aproximaciones del tiempo de ejecución para nuevos conjuntos de ciudades. La precisión del resultado dependerá del límite aceptable que se fije como valor de similitud (relacionado con el valor de la distancia Hausdorff).

5.3.2. Una segunda aproximación

El análisis de resultados de las hipótesis localización de las ciudades y ciudad de comienzo permitió establecer una nueva hipótesis, esta vez basada en los conceptos de suma de distancias entre ciudades y desviación media de estos valores, como alternativa a la distribución espacial, sección 4.4.1.

Una vez que la suma de distancias de la ciudad inicial a las restantes ciudades y la desviación media de estos valores se determinaron como variables claves en el tiempo de ejecución, el módulo de comparación de patrones dejó de tener efecto.

De cara a hacer la predicción, se pasó de estimaciones basadas en patrones gráficos y su comparación, a un modelo de valoración / identificación numérica de los datos de entrada.

5.3.2.1. Análisis y valoración

Queda pendiente cuantificar la relación que existe entre la suma de distancias de una ciudad a las restantes ciudades y la desviación media de éstas con el tiempo de ejecución.

5.4. Sumario

En este capítulo se han expuesto un conjunto de experimentos sintéticos, seleccionados para evaluar la robustez de la metodología. Se han presentado otros escenarios con el objeto de demostrar la escalabilidad tanto en el número de procesadores como en el tamaño del problema. Asimismo, las estimaciones del tiempo de ejecución para nuevos conjuntos de ciudades fueron bastantes buenas del orden del 90 %.

Capítulo 6

Conclusiones y principales aportes

¿Podemos predecir en algoritmos paralelos no deterministas? A lo largo de este documento ha quedado demostrado que sí se puede. El ejemplo de Galileo, Newton, Cavendish, Einstein y el camino que ellos crearon para *hacer ciencia*, lo hemos aplicado a nuestro problema de predicción, en un intento de hacer ciencia según ese modelo.

En esta Tesis se presentó una metodología práctica de predicción para estimar el tiempo de ejecución de aplicaciones paralelas dependientes de los datos.

Concretamente, nuestra propuesta partió de ciertas premisas y utilizó la ciencia computacional para extraer conocimiento de los experimentos diseñados. A medida que se fueron corroborando ciertas conjeturas, los experimentos fueron guiados hacia nuevos objetivos, los cuales a su vez dieron fruto a otros nuevos planteamientos. Las predicciones se fueron cumpliendo una a una.

La idea de trabajar en este sentido surgió de la necesidad de dar respuesta a un número de aplicaciones paralelas frecuentes que normalmente son dejadas a un lado. Cualquier aporte en el área no determinista por mínima que sea representa un gran avance ante la falta de conocimiento general. No sólo hay que pensar en el problema que se resuelve sino en el beneficio que representa para la familia de problemas que contiene al problema resuelto.

La metodología se compone de dos fases, la extracción del conocimiento y la predicción. La fase de extracción del conocimiento comienza con la formulación de las hipótesis, la composición de experimentos y la ejecución de la aplicación estudiada. Luego continúa, con el proceso de descubrir conocimiento (mediante el uso de un ambiente de clustering) con el fin de encontrar clusters (grupos). Termina con la interpretación del modelo y la evaluación de calidad. Finalmente, en la fase de predicción, el módulo de comparación de datos permite lograr una estimación del orden de rendimiento.

Así como los grandes genios experimentaron con objetos, manzanas, espejos, en nuestro caso tres algoritmos paralelos TSP, que generan una solución óptima, fueron desarrollados con el objeto de analizar la influencia de diferentes niveles de indeterminismo en la

predicción de rendimiento: el algoritmo exhaustivo, el algoritmo de poda local y el algoritmo de poda global. Asimismo, los algoritmos fueron usados para mostrar la utilidad y el beneficio de la metodología presentada. Todos siguen el paradigma de programación maestro / trabajador basados en el cálculo de la distancia de las posibles permutaciones y sólo difieren en la estrategia de poda. Específicamente, en el algoritmo exhaustivo no hay poda y el tour mínimo se calcula entre todas las posibles permutaciones, por lo cual la predicción resulta muy exacta. En el algoritmo de poda local, cada trabajador poda según la distancia mínima que va encontrando a medida que procesa una tarea. Por otro lado, en el algoritmo de poda global, los trabajadores podan siempre que sea posible de acuerdo con un valor de distancia global mínima.

Para el **algoritmo exhaustivo**, es indudable que ni los patrones ni la ciudad de origen ni la suma las distancias de la ciudad inicial a las restantes ciudades ni la desviación media de estas sumas afectan el tiempo de ejecución final. En consecuencia, la complejidad asintótica en tiempo se definió como $O(C, P)$. Para el **algoritmo de poda local** y el **algoritmo de poda global**, la suma de las distancias de la ciudad inicial a las restantes ciudades y la desviación media de estos valores se juegan un rol crucial en el tiempo de ejecución. Para estos algoritmos, la complejidad asintótica en tiempo se definió finalmente como $O(C, P, SD, MDSD)$. Las hipótesis que permitieron llegar a esta conclusión fueron corroborándose una a una. Es decir, en forma progresiva se fue mejorando la precisión de las estimaciones.

El modelo de predicción, en un principio, consideraba como parámetros de entrada el número de ciudades, la distribución espacial o patrón de las mismas y la ciudad de origen. Al involucrar un elemento de características topológicas (la distribución espacial), el esquema de predicción incluía un paso de comparación de similitud del patrón de entrada con respecto a los patrones disponibles almacenados en una base histórica. Las estimaciones para nuevos patrones fueron bastantes buenas, del orden del 90 %, lo cual era sumamente alentador.

Una vez más el análisis de resultados permitió establecer una nueva hipótesis, esta vez basada en los conceptos de suma de distancias entre ciudades y desviación media de estos valores, como alternativa a la distribución espacial. Así fue posible establecer una caracterización numérica del parámetro topológico (la distribución espacial), una especie de índice que evalúa los invariantes propios de cada distribución espacial, de manera que, por un lado se simplifica el proceso de predicción al eliminar la evaluación de similitud y por otro se consigue un índice numérico simple en la caracterización de las ciudades.

6.1. Problemas en los que sería factible la aplicación de la metodología expuesta

A continuación se detallan una serie de problemas donde la metodología propuesta podría aplicarse. Cada uno de estos problemas representa un desafío en sí mismo ya que pertenecen a la clase de problemas NP-completo o NP-duro (ver sección 2.1.8).

Problema de satisfacibilidad booleana (SAT): En teoría de la complejidad computacional, el problema de satisfacibilidad booleana fue el primer problema identificado como perteneciente a la clase de complejidad NP-completo [4].

Se trata de un problema donde interesa saber si una expresión booleana con variables y sin cuantificadores tiene asociada una asignación de valores para sus variables que determina que la expresión sea verdadera. Por ejemplo, una instancia de SAT sería el saber si existen valores para x_1, x_2, x_3 y x_4 tales que la expresión:

$$(x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \tag{6.1}$$

sea cierta.

Problema de la mochila: El problema de la mochila pertenece a la clase de problemas NP-duro. El objetivo del problema es llenar una mochila de tal forma que se maximice el valor de los objetos transportados en ella [5].

Se tienen n objetos distintos y una mochila. Cada uno de los objetos tiene asociado un peso positivo w_i y un valor positivo v_i para $i = 1, ..., n$. La mochila puede llevar un peso que no sobrepase una cantidad W. Hay que tener en cuenta que la suma de los pesos de los objetos seleccionados no debe superar la capacidad máxima W de la mochila. Por lo tanto, para obtener la solución deseada, se debe decidir para cada uno de los objetos, si se introduce o no en la mochila. Cada objeto tiene asociado una variable x_i que toma el valor 1 si el objeto se mete en la mochila y 0 en caso contrario. Cualquier objeto se puede introducir en la mochila si hay espacio para él, pero lo que no se puede hacer es transportar en ella una fracción o parte de un objeto.

Partición de grafos: El problema de partición de grafos pertenece a la clase de problemas NP-completo.

Dado un grafo con pesos $G(V, E)$ y enteros j, k y m. La partición de vértices en m sub-conjuntos de tal manera que cada sub-conjunto tenga tamaño de a lo sumo j, mientras que el costo de los bordes que atraviesan los subconjuntos esté limitado por k [4].

Problema de empaquetar un conjunto de objetos: El problema de empaquetar un conjunto de objetos pertenece a la clase de problemas NP-duro y se encuentra en la misma familia que el problema de la mochila. El objetivo del problema es almacenar una serie de objetos usando el menor número de compartimientos [4].

Se define un conjunto de n objetos con tamaños $d_1, ..., d_n$ y un conjunto de m compartimientos con capacidades $c_1, ..., c_m$.

Problema de la planificación de movimiento: Este problema, también conocido como el problema de la navegación, aunque es muy simple para los seres humanos, es uno de los desafíos más importantes en el área de robótica. El problema consiste en encontrar un camino en un cuarto desde una posición inicial hasta una posición final sorteando una serie de obstáculos [6].

6.2. Contribuciones

A continuación se resumen las contribuciones de esta Tesis:

- Elaboración de una metodología práctica para la predicción de rendimiento de aplicaciones dependientes de los datos.
- Aporte en el área no determinista, un ámbito poco explorado.
- Utilización de la ciencia computacional para extraer conocimiento desde los mismos datos de entrada con el fin de construir un modelo de predicción.
- Facilidad para comprobar nuevas conjeturas y/o hipótesis.
- Sencillez para afinar y refinar la metodología con el fin de mejorar la precisión del orden de las predicciones.
- Desarrollo de tres algoritmos paralelos de TSP euclideano (el algoritmo exhaustivo, el algoritmo de poda local y el algoritmo de poda global) con el objeto de analizar la influencia de distintos grados de indeterminismo en la predicción de rendimiento y al mismo tiempo validar la metodología presentada.
- Construcción de un ambiente dinámico y abierto de agrupamiento donde puede aplicarse tanto el algoritmo k-means como sus derivaciones fuzzy k-means, k-prototypes, k-modes, k-medoid, k-means^{+} y k-means^{++}, entre otros.
- Análisis de otros problemas donde factiblemente se podría aplicar la metodología enunciada.

6.3. Líneas abiertas

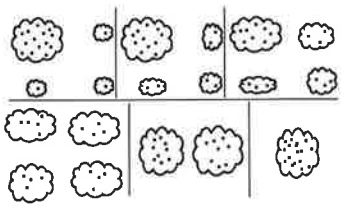
En cuanto al problema del viajante sería interesante cuantificar la relación que existe entre la suma de distancias de una ciudad a las restantes ciudades y la desviación media de éstas con el tiempo de ejecución. Los experimentos que ya fueron analizados para otras cuestiones y que pueden aportar información en este sentido son:

- Un círculo de ciudades más una ciudad que se va alejando en forma constante.

circulo.15.1000
circulo.15.2000
circulo.15.3000
circulo.15.4000

- Unas ciudades alejadas se van acercando hacia el grupo mayoritario de ciudades. Es decir, las ciudades aisladas del patrón wi29.patron.mc.14.1.1.1 estarán ubicadas a mitad de camino que en wi29.patron.14.1.1.1. Por otro lado las ciudades aisladas del patrón wi29.patron.mc2.14.1.1.1, a su vez, estarán a un cuarto de camino respecto de wi29.patron.14.1.1.1 o lo que es lo mismo a mitad de camino de wi29.patron.mc.14.1.1.1.

wi29.patron.14.1.1.1	wi29.patron.mc.14.1.1.1	wi29.patron.mc2.14.1.1.1
wi29.patron.11.2.2.2	wi29.patron.mc.11.2.2.2	wi29.patron.mc2.11.2.2.2
wi29.patron.8.3.3.3	wi29.patron.mc.8.3.3.3	wi29.patron.mc2.8.3.3.3
wi29.patron.5.4.4.4	wi29.patron.mc.5.4.4.4	wi29.patron.mc2.5.4.4.4
wi29.patron.5.4.4.4	wi29.patron.mc.5.4.4.4	wi29.patron.mc2.5.4.4.4
wi29.patron.17	wi29.patron.mc.17	wi29.patron.mc2.17



La consideración de casos más generales, como por ejemplo, la extensión de ciudades R^n en lugar de trabajar con R^2 es otro punto que se debiera investigar. La geometría de reconocimiento de patrones quedaría a un lado siendo de utilidad la teoría de reconocimiento de patrones espacial.

La metodología de predicción presentada intenta estimar el tiempo de ejecución de aplicaciones paralelas dependientes de los datos. Hay que tener presente en todo momento que, la metodología es general y está destinada para un cierto tipo de aplicaciones no deterministas. Estas aplicaciones deben cumplir con determinadas características tales como llevar aparejado un elevado tiempo de cómputo en su resolución. Es indudable, que a medida que se estudia una aplicación en profundidad, la metodología va sufriendo cambios / adecuaciones particulares e imprescindibles para la mejora del objetivo final, la predicción concreta del problema analizado. Estos cambios no deben desestimar en lo más mínimo la metodología general porque se debe reconocer que sin ella no hubiera sido posible la generación de estas adecuaciones. Dicho todo esto, ahora cabe la necesidad de aplicar la metodología expuesta en alguno de los problemas enunciados en la sección 6.1. Estos problemas no son casuales. Además de observarse problemas en los que sería factible la aplicación de la metodología expuesta, se tuvo en cuenta que fueran desafiantes y, a la vez, útiles.

Apéndice A

Descubrir conocimiento en bases de datos

El término descubrir conocimiento en bases de datos (knowledge discovery in databases - KDD) se refiere al proceso de analizar datos desde diferentes perspectivas y resumir esto en información útil. Técnicamente, KDD es el proceso de encontrar correlaciones o patrones entre docenas de campos en una gran base de datos relacional. Resulta de interés para investigadores de aprendizaje automático, reconocimiento de patrones, base de datos, estadísticas, inteligencia artificial, adquisición de conocimiento para sistemas expertos y visualización de datos.

Un proceso KDD, Figura A.1, comprende la preparación de los datos, la definición de un estudio, la lectura de los datos junto con la construcción del modelo, la interpretación del modelo y finalmente la predicción. Es un proceso interactivo e iterativo, que supone numerosos pasos con muchas tomas de decisiones que debe realizar el usuario final [71].

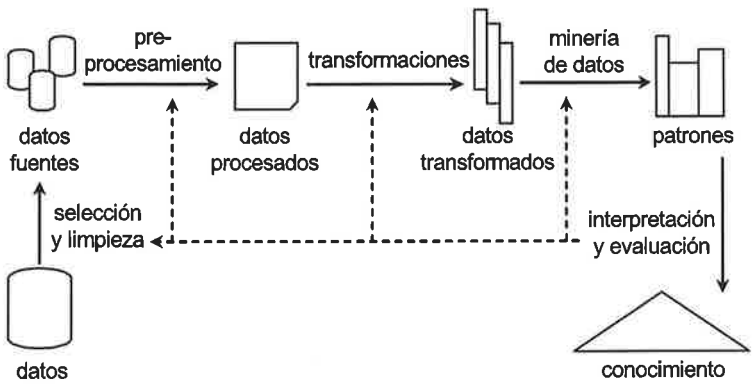


Figura A.1: Proceso KDD.

Preparación de datos: La preparación de datos implica la limpieza de datos, el tratamiento de los valores perdidos, la derivación de los datos y la combinación de los

mismos.

La limpieza de datos puede ser entendida como la tarea de estandarizar los valores (problema de consistencia), borrar los errores tipográficos y modificar los valores incorrectos de acuerdo con otros campos.

La identificación de excepciones es de utilidad para descubrir datos incorrectos o para borrar muestras no representativas. Si se desea trabajar con campos donde existen valores perdidos es vital definir previamente ciertas estrategias.

En lo que concierne a la combinación de los datos, a veces, los campos relevantes pueden ser combinaciones / derivaciones de otros campos. En general, la conversión de datos resulta necesaria de modo tal que las medidas compartan escalas similares. Asimismo, si los datos de interés provienen de diversas bases de datos, se hace imprescindible un almacenamiento común con las consiguientes conversiones / combinaciones / derivaciones.

Definición de un estudio: La definición de un estudio involucra plantear el objetivo que se quiere, elegir una salida adecuada para mostrar ese objetivo y realizar la reducción dimensional, es decir, determinar los campos que intervendrán en el estudio.

Por lo demás, no siempre es necesario minar el conjunto completo de datos. En general, para un análisis es suficiente trabajar con un subconjunto de los datos elegidos al azar. Algunas veces, se usa un subconjunto de los datos para construir un modelo, otro subconjunto para validar dicho modelo y aún un tercer subconjunto para hacer predicciones sobre la base del modelo construido.

En esta etapa, se debe seleccionar la técnica de exploración de datos o minería de datos (data mining) a aplicar. Entre las posibles técnicas se encuentran la clasificación, la regresión, el clustering, el modelo de dependencia, el resumen de datos y la detección de cambios y desvíos.

Una vez escogida la técnica de minería, se debe seleccionar el algoritmo / método con el cual se trabajará para la búsqueda de estructuras en los mismos datos.

Finalmente, se deben buscar las estructuras de interés en una particular representación: reglas de clasificación o árboles de clasificación, agrupamiento, etc.

Lectura de datos y construcción de un modelo: Una vez que el estudio ha sido definido, un producto de minería de datos lee un conjunto de datos y construye un modelo. Cualquier modelo resume una gran cantidad de datos por acumulación de indicadores.

La invariabilidad del modelo será imperfecta si el conjunto de datos usado contiene anomalías (ruido). El diseño de filtros para disminuir el impacto de tales anomalías es una solución que definitivamente mejora la exactitud de un modelo.

Interpretación del modelo: Una vez que los datos han sido minados, el proceso de interpretar el modelo creado desde el conjunto de datos, implica diferentes aspectos. Entre otras cosas, un modelo debería mostrar pesos, frecuencias y conjunciones importantes que sirven para describir el objetivo. Por supuesto un modelo solamente es útil si éste se basa en suficientes muestras representativas.

La validación del modelo construido supone hacer predicciones usando un conjunto de datos y comparar dichas predicciones con resultados conocidos. Elegir un conjunto de datos desvinculado de los datos usados para la construcción del modelo es una buena manera de entender luego la capacidad de predicción.

En esta etapa también es útil inferir por qué razones el modelo tiene éxito en predecir correctamente o por qué motivos falla como parte del proceso de validación.

La evaluación también debe incluir medidas de utilidad, de novedad y de interés.

Esta etapa puede determinar un retorno a pasos previos con el fin de realizar actividades adicionales para proveer información extra.

Predicción: La consolidación del conocimiento descubierto es crucial y puede ser muy desafiante. La predicción consiste en el proceso de elegir las mejores posibles salidas basándose en datos históricos.

Aunque la mayoría de los trabajos KDD se focalizan en la etapa de minería de datos, las otras etapas también son muy importantes para la satisfacción general del proceso KDD.

A.1. Modelos de minería de datos

En el corazón de todo producto de minería de datos, se halla el proceso de construir un modelo para representar un conjunto de datos, pero la manera en que se construye un modelo no es común a todos los productos de minería de datos. Hay diversos enfoques como el uso de los árboles de decisión, los algoritmos genéticos, las redes neuronales, las redes de agente, las estadísticas y los modelos híbridos [71].

Árboles de decisión: Por largo tiempo en ciencias de la computación, se usó la idea de crear una estructura de árbol para describir un conjunto de datos. Sin embargo, los árboles de decisión no constituyen el enfoque preferido para describir conocimiento.

Entre los beneficios que presentan los árboles reside su poder de entendimiento y el empleo de diferentes algoritmos matemáticos para determinar cómo agrupar y dar importancia a las distintas variables.

Algoritmos genéticos: Los algoritmos genéticos son un método de optimización combinatoria y se basan en procesos de evolución biológica. La evolución envuelve la optimización de un modelo de datos usando métodos genéticos para obtener modelos más aptos.

Redes neuronales: Las redes neuronales son un método de aprendizaje cuya finalidad inicial era la de emular los procesadores biológicos de información. Las redes neuronales parten de la presunción de que la capacidad humana de procesar información se debe a la naturaleza biológica de nuestro cerebro.

Redes de agentes: Las redes de agentes comprenden todos los elementos de datos o categorías de datos elementales como agentes, los cuales están conectados unos a otros de una manera significativa. Las redes de agentes distinguen condiciones requeridas de condiciones del contexto.

Estadísticas: En este enfoque, por ejemplo, la regresión lineal usa probabilidades, análisis de datos e inferencia estadística para modelar.

Modelos híbridos: Existen diversos modelos híbridos que combinan los distintos enfoques mencionados.

A.2. Técnicas de minería de datos

Las técnicas de minería de datos pueden ser clasificadas en: modelo predictivo, clustering (agrupamiento), modelo de dependencia, resumen de datos y detección de cambios y desvíos.

Modelo predictivo: Dentro del modelo predictivo se encuentran la clasificación y la regresión. La clasificación es una función que clasifica un dato en una de las diversas clases predefinidas. La regresión es una función que clasifica un dato en un valor real de una variable de predicción.

Clustering (agrupamiento): Clustering es el proceso de dividir un conjunto de datos en clusters (sub conjuntos), de modo tal que los datos en cada cluster (idealmente) compartan un cierto rasgo común (con frecuencia proximidad de acuerdo con una medida de distancia predefinida). Un ejemplo gráfico de clustering se muestra en la Figura A.2.

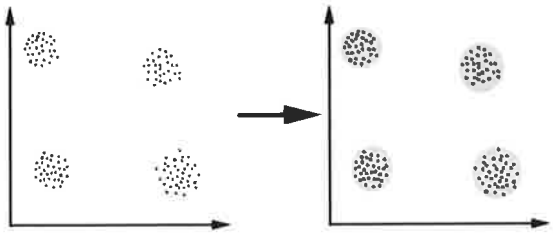


Figura A.2: Un simple ejemplo gráfico de clustering.

Modelo de dependencia: El modelo de dependencia consiste en encontrar un modelo que describa las dependencias significativas entre las variables.

Los modelos de dependencia a nivel estructural especifican qué variables dependen de otras y a nivel cuantitativo muestra las fuerzas de las dependencias usando alguna escala numérica.

Resumen de datos: El resumen de datos considera métodos adecuados para encontrar una descripción compacta para un sub conjunto de datos. Hay dos clases de métodos que representan grupos horizontales de datos (casos) o grupos verticales de casos (campos).

Detección de cambios y desvíos: La detección de cambios y desvíos se focaliza en descubrir los cambios significativos en los datos en relación con valores medidos previamente. Las secuencias pueden ser dadas como series en el tiempo. La característica distintiva de esta clase de métodos es que el orden de las observaciones resulta importante.

A.2.1. Clustering

El interés en clustering (o agrupamiento) puede hallarse tanto en gran cantidad de estudios científicos como en posibles aplicaciones prácticas. Se advierte en los problemas de reconocimiento de voz y palabras en acústica, de taxonomía en biología, de clasificación de enfermedades por sus síntomas en medicina, de clasificación de piezas arqueológicas en arqueología, de segmentación del mercado en marketing.

Al término clustering se lo conoce como análisis de grupos (clusters analysis) dentro de la terminología de estadística. El análisis de grupos consiste en la búsqueda, a través de los datos, de observaciones que sean suficientemente similares entre sí como para poder identificarse como integrantes de un grupo común. Este proceso es muy intuitivo y a la vez un objetivo natural. Por ejemplo, las galaxias de estrellas en el universo pueden describirse como grupos usando tres dimensiones.

Si se busca precisión sobre qué es un grupo (cluster), se puede entrar en controversia y encontrar que no hay una definición precisa, aceptada por todos. Algunos especialistas dirán que los grupos se corresponden con descubrir poblaciones subyacentes. Otros tenderán a pensar grupos como estructuras débiles pero al mismo tiempo encontrarán útiles las estructuras determinadas por los datos. Por ahora, es suficiente pensar que los grupos poseen rasgos que se parecen entre sí y que a la vez los grupos están claramente separados entre ellos. Si cada observación o rasgo está asociado con uno y sólo un grupo, entonces una partición de los datos puede ser útil para propósitos estadísticos.

Por ejemplo, es posible resumir un gran conjunto de datos multi-variables en términos de un miembro típico de cada grupo. Esto sería más significativo que sólo ver a un único miembro típico de un dato y mucho más conciso que efectuar descripciones individuales de cada observación.

Encontrar la partición en grupos no es una tarea fácil como aparenta serlo. Excepto en problemas pequeños, donde se pueden considerar todas las posibles particiones de los datos en grupos para luego quedarse con la mejor, el resto de los problemas está computacionalmente fuera de cuestión. En consecuencia, numerosos métodos involucran procedimientos para encontrar grupos en una forma razonablemente eficiente.

Si X es el conjunto de datos (conjunto de entrenamiento), k la cantidad de particiones que queremos obtener y S_i cada partición mutuamente exclusiva, tenemos que:

$$X = S_1 \cup \ldots \cup S_k = \cup_{i=1}^{k} S_i \tag{A.1}$$

Algunos ejemplos de clustering son: el método de k-means, los métodos aglomerativos jerárquicos, los modelos mixtos que usan el método de máxima estimación (EM), etc.

Los métodos de clustering pueden ser organizados en tres clases dado el número k de grupos. Estas clases son:

Métodos basados en la distancia métrica: Usando una medida de distancia y una función objetivo, se trata de encontrar la mejor k partición de los datos. Esto es, que los elementos de un grupo estén más cercanos entre sí que con respecto a elementos de otro grupo.

Métodos basados en modelos: Cada grupo recibe un valor relacionado con la función probabilidad.

$$P(CM_k|X) = P(X|CM_k)\frac{P(CM_k)}{P(X)} \tag{A.2}$$

La idea es encontrar el mejor ajuste para cada grupo. *P(X)* es una constante que puede ignorarse para propósitos de comparación. $P(CM_k)$ es la probabilidad anteriormente asignada al grupo CM_k. En métodos de máxima probabilidad, todos los

grupos se asumen como igualmente probables; en consecuencia el término es ignorado. Al ignorar este término el problema reside en que siempre se prefieran modelos más complejos y esto lleva a un sobre ajuste (over fitting) en los datos.

Métodos basados en partición: Básicamente, enumeran varias particiones y luego otorgan un puntaje a las mismas de acuerdo con algún criterio. Muchos métodos de inteligencia artificial se encuentran en esta categoría. Los mismos utilizan funciones ad hoc para otorgar el puntaje.

A su vez, existe otra manera de clasificar los métodos de clustering, el método jerárquico y el método particionado.

A.2.1.1. Métodos jerárquicos

Los métodos de agrupamiento jerárquico toman como entrada un conjunto de datos y construyen un árbol que agrupa elementos con características similares. Estos métodos permiten partir el conjunto de entrada en distintas cantidades de grupos dependiendo del nivel de abstracción en el cual el analista se halle interesado. La manera más común de partir un árbol construido por un método de agrupamiento jerárquico consiste en cortar horizontalmente el árbol en un determinado nivel. Esta partición asegura que dos elementos de la misma clase estén más cerca uno de otro que dos elementos de diferentes clases. Sin embargo, el analista podría requerir más detalles en algunas clases que en otras; por lo tanto, en este caso, sería conveniente tomar grupos que se encuentren en distintos niveles del árbol.

Existen dos tipos de métodos jerárquicos: el método jerárquico aglomerativo y el método jerárquico divisivo.

El agrupamiento jerárquico aglomerativo forma un grupo por cada dato del conjunto de casos. Luego une estos grupos atómicos en grupos cada vez más grandes hasta que todos los datos pertenezcan a un único grupo. En este tipo de métodos no influye el orden en el cual se obtienen los elementos pero hay que mencionar que son susceptibles a la toma de decisiones locales. La mayoría de los métodos jerárquicos pertenecen a esta categoría.

El agrupamiento jerárquico divisivo forma un sólo grupo con todos los datos del conjunto de casos. Dicho grupo se va subdividiendo y así sucesivamente.

En ambos casos los resultados se describen mejor usando un árbol binario o un dendograma. En un árbol, cada hoja es un caso (objeto) que representa un grupo, los niveles intermedios son grupos que contienen varias observaciones y la raíz es un sólo grupo que contiene todos los casos.

A.2.1.2. Métodos particionados

Los métodos particionados dividen los datos (objetos) en k subconjuntos llamados grupos, de forma tal que los objetos dentro de un grupo son más similares entre sí que comparán-

dolos con los objetos de otros grupos. Cada grupo se representa por un centro de gravedad o por uno de los objetos del grupo ubicado cerca del centro. k es un parámetro de entrada para estos métodos, o sea, se requiere cierto conocimiento del dominio.

En general, los métodos particionados comienzan con una partición inicial y luego usan una estrategia de control iterativa para optimizar una función objetivo. Asimismo los objetos son asignados a grupos cuyo elemento representativo es el más cercano.

Existen diversos métodos de particionamiento entre los que se pueden mencionar la enumeración exhaustiva y la optimización iterativa.

La enumeración exhaustiva examina las $k^m/k!$ formas de dividir el conjunto de m datos en k grupos. No es aplicable, salvo para valores de k y m pequeños.

La optimización iterativa comienza con una partición inicial, luego prueba con todos los movimientos e intercambios de los datos de un grupo a otro para ver si eso mejora la calidad del agrupamiento. Puede encontrar un mínimo local, pero la calidad del mismo depende en gran medida de la partición inicial seleccionada. Estos métodos tienen complejidad exponencial.

El método particionado más conocido es el algoritmo de k-means.

K-means

Las ideas originales relacionadas con el algoritmo de k-means fueron presentadas por MacQueen en 1967 [37]. En un principio el método fue desarrollado para reconocer patrones estadísticos; con el correr del tiempo y gracias a su eficiencia en el manejo de grandes volúmenes de datos su uso se difundió en diversos campos de estudio.

K-means es uno de los más simples algoritmos particionados de aprendizaje no supervisado que resuelve el bien conocido problema de clustering. El trabajar con valores numéricos limita su uso en el terreno de exploración de datos ya que en la mayor parte de los casos se cuenta con datos categóricos. A veces el proceso de clustering no converge por lo que no pueden hallarse los grupos en una forma correcta. En consecuencia, se puede especificar un número máximo de iteraciones que una vez alcanzado hace que concluya el proceso.

El procedimiento sigue una simple y fácil manera de clasificar un conjunto de datos dado un cierto número de clusters (se asumen k clusters) fijados a priori.

La idea principal reside en definir k centroides, uno por cada cluster. Inicialmente, estos centroides deberían elegirse de una manera astuta ya que distintos comienzos pueden causar diferentes resultados. En consecuencia, la mejor opción es elegir los centroides tan apartados unos de otros como sea posible. El próximo paso consiste en tomar cada muestra perteneciente a un conjunto de datos dado y asociar ésta con el centroide más cercano en términos de la distancia euclideana. Cuando no existe muestra sin asignar, el primer paso ha finalizado teniéndose un agrupamiento inicial. En este punto, es necesario recalcular los k nuevos centroides como baricentros de los clusters resultantes del paso previo. Si alguno de los centroides ha cambiado, una reasignación de los datos al centroide más

Algoritmo 10 : Pasos del algoritmo k-means.

1. Elegir k centroides como medias de los clusters
2. Asignar cada muestra al cluster que tiene el centroide más cercano
3. Cuando todos las muestras han sido asignadas, recalcular las posiciones de los k nuevos centroides
4. Repetir los pasos 2 y 3 hasta que los centroides no cambien o se llegue a la máxima cantidad de iteraciones (convergencia del algoritmo)

cercano debe realizarse. Mientras el Algoritmo 10 describe los principales pasos envueltos en k-means, la Figura A.3 muestra su funcionamiento.

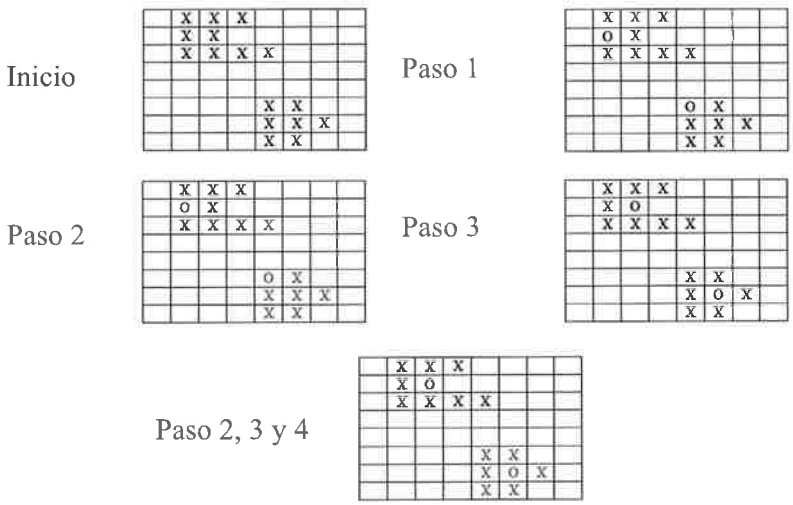


Figura A.3: Funcionamiento de k-means para k=2.

Si se tiene $X = x_1, x_2, .., x_m$ un conjunto de m casos en R^n un agrupamiento duro consiste de k conjuntos disjuntos de X tal que $X = \cup_{l=1}^{k} S_l$. El método debe seleccionar los subconjuntos $S_1, S_2, ..., S_k$ de modo tal que minimicen $q_l \in R^n$ la media de los vectores de datos en S_l. En otras palabras, se eligen los grupos de tal manera que se minimicen las distancias de los objetos en los grupos al *centro* del grupo (se favorece la alta compactación intra grupo).

La función de costo a minimizar es:

$$E = \sum_{l=1}^{k} \sum_{i=1}^{m} y_{i,l} d(x_i, q_l) \tag{A.3}$$

donde m es la cantidad de objetos en un conjunto de datos X, $x_i \in X$, q_l es la media del grupo l, $y_{i,l}$ es un elemento de una matriz de partición Y_{m*k}, d es una medida de disimilitud (generalmente definida como el cuadrado de la distancia euclideana).

Además, la matriz de partición Y_{m*k} tiene las siguientes dos propiedades:

- $\forall y_{i,l} : y_{i,l} \in 0, 1$ esta propiedad hace que Y sea una partición dura
- $\forall i/1 \leq i \leq m : \sum_{l=1}^{k} y_{i,l} = 1$

Debido a que el algoritmo k-means puro sólo trabaja con datos numéricos y la medida de distancia elegida debe ser la euclideana, se crearon diversas derivaciones producto de la necesidad. Entre éstas derivaciones están el algoritmo fuzzy k-means, k-prototypes, k-modes, k-medoid, k-means^{+}, k-means^{++}, entre otras. Básicamente, los algoritmos mencionados difieren en cómo seleccionan los k centroides iniciales, en las medidas de distancia o disimilitud usadas y en la estrategia para calcular la media de un grupo.

Apéndice B

Ambiente Cluster-Frame

Cluster-Frame es un ambiente dinámico y abierto de clustering. Este ambiente permite la evaluación de algoritmos de clustering como k-means, k-prototypes, k-modes, k-means^{++}. Los resultados alcanzados aplicando un algoritmo o distintos algoritmos, usando diversos parámetros pueden ser analizados desde distintos ángulos y comparados fácilmente [38].

B.1. Requerimientos del sistema

Para utilizar la aplicación se necesita contar con los siguientes requerimientos de software y hardware:

- PC IBM compatible 486 DX o superior.
- Microsoft Windows 95 o superior.
- 16Mb de memoria RAM.
- Definición de video VGA, resolución 800 x 600 pixeles.

B.2. Instalación

La instalación del ambiente Cluster-Frame es similar a la instalación de cualquier otra aplicación Windows.

Inserte el CD en su CD-ROM y luego seleccione el drive correspondiente. Ejecute la aplicación **Setup** realizando doble click. Una vez copiados ciertos archivos básicos se pide una reconfirmación de instalación, Figura B.1. Para continuar presione el botón de **Aceptar**. En el caso que presione el botón **Salir** se borrarán todos los archivos copiados hasta el momento.

Figura B.1: Reconfirmación de instalación.

El programa de instalación agrega los elementos al grupo de programas seleccionado. Elija **Continuar** si desea crear un nuevo grupo de programas llamado Cluster-Frame; en caso contrario seleccione un grupo existente. A continuación comienza la copia de los distintos archivos con la distribución de los mismos a los directorios correspondientes.

Al finalizar la copia de todos los archivos que hacen que funcione el ambiente Cluster-Frame se despliega una pantalla de aviso.

B.3. Ejecución del ambiente Cluster-Frame

Para acceder a la aplicación ejecute Cluster-Frame (Inicio / Programas / Cluster-Frame). La pantalla principal, Figura B.2, cuenta con las opciones de:

Método / Nuevo: permite abrir una ventana para ejecutar un nuevo método.

Método / Salida: Facilita salir de la aplicación.

Ver: Brinda la posibilidad de visualizar - no visualizar la barra de herramientas o la barra de estado.

Ayuda / Acerca de Ambiente de Agrupamiento: Muestra información del sistema y de la aplicación Cluster-Frame.

Ayuda / Sugerencias del día: Displaya las sugerencias (archivo TipOfDay.txt).

Ayuda / Contenido: Nos presenta una ayuda en formato HTML de las características de la aplicación.

Una vez abierta una nueva ventana para ejecutar un método se deben realizar diversas selecciones, Figura B.3.

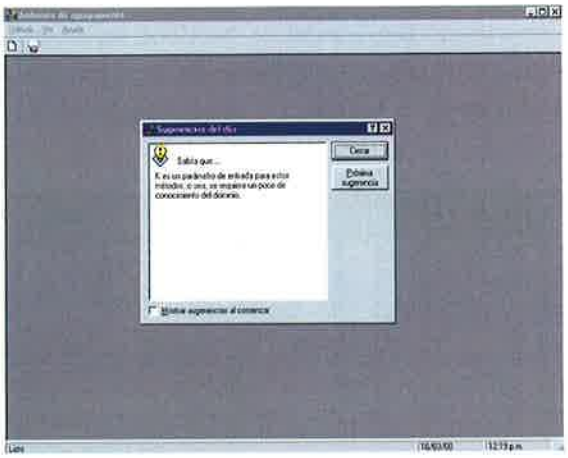


Figura B.2: Pantalla principal.

- Escoger la base de datos de origen; esto se realiza a través de **Base de datos / Abrir base de datos origen**.
- Escoger la base de datos de resultados; esto se realiza a través de **Base de datos / Abrir base de datos resultados**.
- **Tabla / Seleccionar muestra de casos** permite elegir la tabla muestra con la que se va a trabajar.
- La distancia a aplicar en cada atributo y la distancia global se selecciona accediendo a **Distancias / Configurar Distancias**.

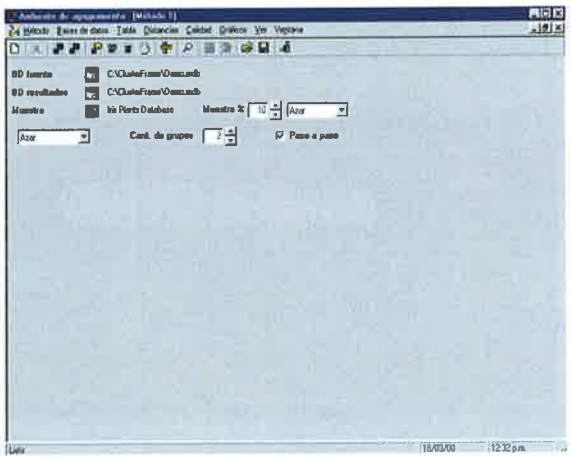


Figura B.3: Selección de la muestra de casos.

La pantalla de configuración de distancias, Figura B.4, despliega el conjunto de casos sobre los que se aplicará el método, un conjunto de distancias globales habilitadas y la

estructura de la tabla que contienen los casos. Se debe seleccionar la distancia global a utilizar y en el panel que contiene la estructura de la tabla de casos se puede otorgar un peso y escoger una distancia a aplicar de acuerdo con el tipo de dato del atributo previamente marcado.

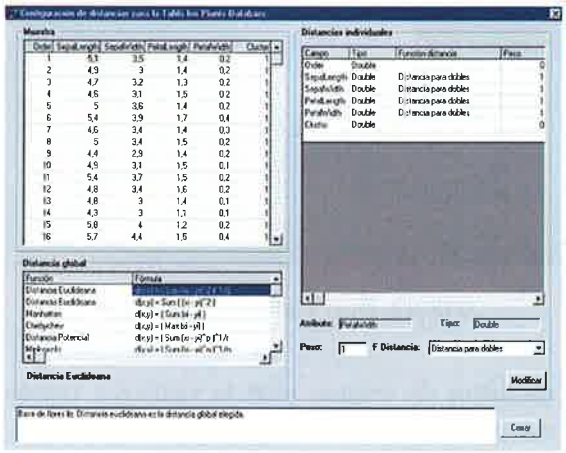


Figura B.4: Configuración de distancias.

Para elegir la forma de parada del método (es decir, la convergencia del algoritmo) hay que seleccionar **Calidad / Configurar Calidad**, ver Figura B.5. Las tres posibilidades que se cuentan son: mínima cantidad de cambios permitidos, máxima cantidad de iteraciones y medida de compactación aceptada. Para todas se debe especificar un parámetro o bien se puede dejar el que aparece por defecto.

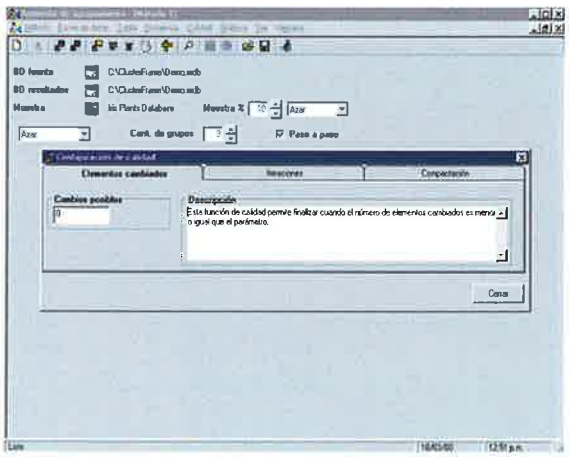


Figura B.5: Calidad aceptable.

Finalmente antes de comenzar la ejecución que puede ser paso a paso o total, se tiene que especificar la cantidad de grupos buscados (el k), el porcentaje de casos que intervendrán en la ejecución y la forma de seleccionar estos casos. Luego de todo esto, se está en

condiciones de ejecutar el método mediante la opción **Método / Correr**. Las ejecuciones paso a paso pueden detenerse mediante la opción **Método / Parar**. Ver Figura B.6.

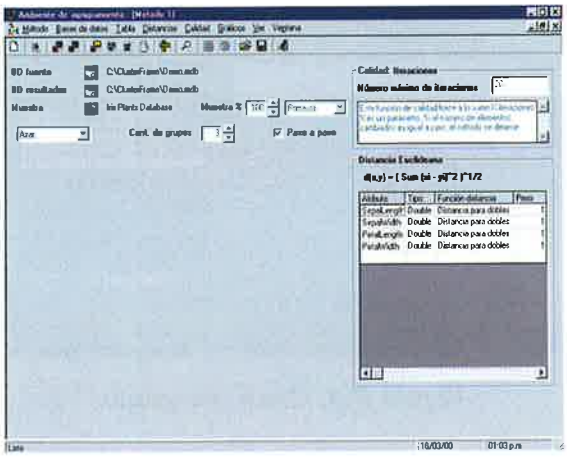


Figura B.6: Antes de ejecutar el método.

Al finalizar la ejecución del método se pueden observar los resultados de diversas maneras. Una de ellas, es un gráfico x-y (cambiable al formato deseado mediante doble click) que mide la evolución de la compactación de los grupos, Figura B.7. Si desea observar la evolución de las distintas medidas de calidad seleccione **Calidad / Configurar calidad**.

Figura B.7: Evolución de la compactación.

Seleccionando **Gráficos / Torta** se puede observar la distribución de los casos por grupo, Figura B.8. Alternativamente por la opción de menú Gráficos se puede cambiar del gráfico de compactación al gráfico de cantidad de casos por grupo.

Al seleccionar una porción del gráfico de torta de la Figura B.8, se abre una nueva ventana con información relacionada con el grupo elegido. La grilla superior despliega los casos que recayeron en dicho grupo y se agrega un atributo que es el valor de la distancia del

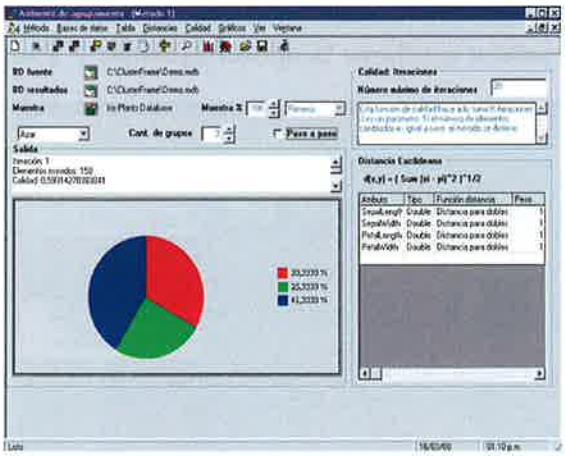


Figura B.8: Casos por grupo.

caso (objeto) al medioide. En la parte inferior, aparecen unos cuantos datos estadísticos como ser el medioide.

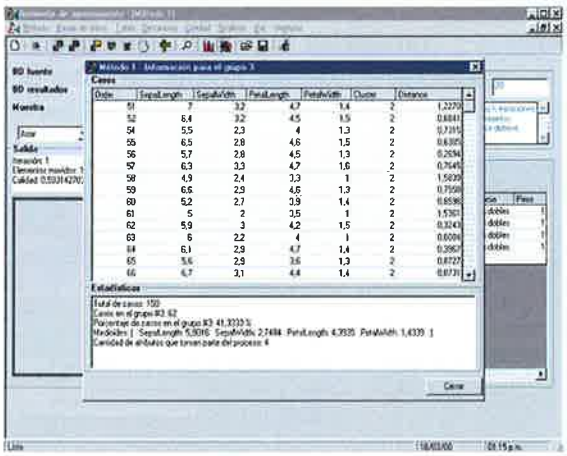


Figura B.9: Información de un grupo.

Apéndice C

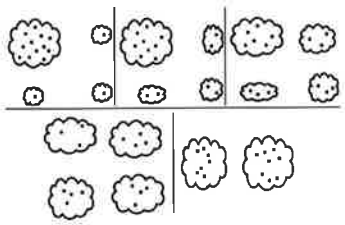
Experimentación

En este apéndice se presentan y describen los experimentos que se usaron para testar las fases de la metodología de predicción de rendimiento con respecto al problema TSP. Los mismos han servido para ir refinando la definición de la complejidad asintótica en tiempo de los algoritmos TSP trabajados como así también para predecir el orden de rendimiento de nuevos conjuntos de datos.

A continuación se especifican las clases analizadas. Una clase contiene diversos experimentos donde cada uno de éstos sigue un patrón gráfico. A su vez, por cada experimento se realizaron C ejecuciones ya que se consideró una ciudad inicial diferente en cada oportunidad.

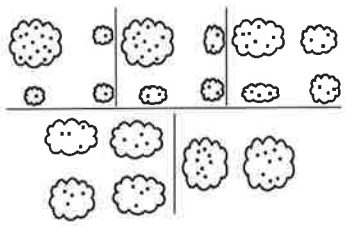
Clase 1 :

patron.12.1.1.1
patron.9.2.2.2
patron.6.3.3.3
patron.3.4.4.4
patron.7.8



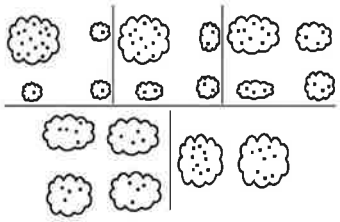
Clase 2 :

patron.13.1.1.1
patron.10.2.2.2
patron.7.3.3.3
patron.4.4.4.4
patron.8.8



Clase 3 :

patron.14.1.1.1
patron.11.2.2.2
patron.8.3.3.3
patron.5.4.4.4
patron.9.8

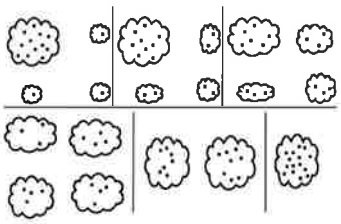


Clase 4 :

wi29.patron.12.1.1.1
wi29.patron.9.2.2.2
wi29.patron.6.3.3.3
wi29.patron.3.4.4.4
wi29.patron.7.8
wi29.patron.15

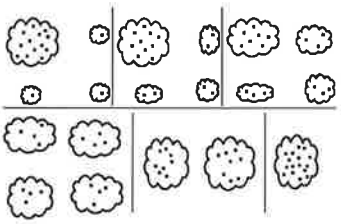
wi29.div_2.patron.12.1.1.1
wi29.div_2.patron.9.2.2.2
wi29.div_2.patron.6.3.3.3
wi29.div_2.patron.3.4.4.4
wi29.div_2.patron.7.8
wi29.div_2.patron.15

wi29.x3.patron.12.1.1.1
wi29.x3.patron.9.2.2.2
wi29.x3.patron.6.3.3.3
wi29.x3.patron.3.4.4.4
wi29.x3.patron.7.8
wi29.x3.patron.15



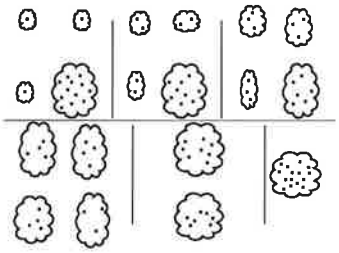
Clase 5 :

wi29.patron.13.1.1.1
wi29.patron.10.2.2.2
wi29.patron.7.3.3.3
wi29.patron.4.4.4.4
wi29.patron.8.8
wi29.patron.16



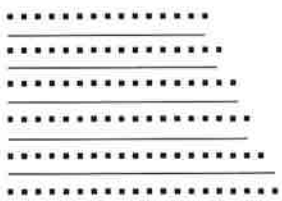
Clase 6 :

wi29.swap.patron.12.1.1.1
wi29.swap.patron.9.2.2.2
wi29.swap.patron.6.3.3.3
wi29.swap.patron.3.4.4.4
wi29.swap.patron.7.8
wi29.swap.patron.15



Clase 7 :

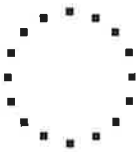
coordenada_y0.patron.15
coordenada_y0.patron.16
coordenada_y0.patron.17
coordenada_y0.patron.18
coordenada_y0.patron.19
coordenada_y0.patron.20



Clase 8 :

circulo.15
circulo.16
circulo.17
circulo.18
circulo.19
circulo.20
circulo.21
circulo.22
circulo.23
circulo.24

circulo.x3.16
circulo.x3.17
circulo.x3.18
circulo.x3.19
circulo.x3.20
circulo.x3.21
circulo.x3.22



Clase 9 :

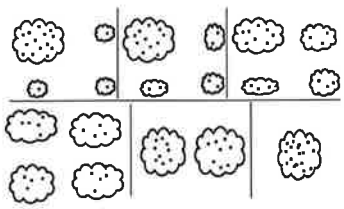
circulo.15.1000
circulo.15.2000
circulo.15.3000
circulo.15.4000

Clase 10 :

wi29.patron.14.1.1.1
wi29.patron.11.2.2.2
wi29.patron.8.3.3.3
wi29.patron.5.4.4.4
wi29.patron.5.4.4.4
wi29.patron.17

wi29.patron.mc.14.1.1.1
wi29.patron.mc.11.2.2.2
wi29.patron.mc.8.3.3.3
wi29.patron.mc.5.4.4.4
wi29.patron.mc.5.4.4.4
wi29.patron.mc.17

wi29.patron.mc2.14.1.1.1
wi29.patron.mc2.11.2.2.2
wi29.patron.mc2.8.3.3.3
wi29.patron.mc2.5.4.4.4
wi29.patron.mc2.5.4.4.4
wi29.patron.mc2.17



La idea en la clase 10 es que las ciudades se van acercando hacia el grupo mayoritario. Es decir, las ciudades aisladas del patrón wi29.patron.mc.14.1.1.1 estarán ubicadas a mitad de camino que en wi29.patron.14.1.1.1. Por otro lado las ciudades aisladas del patrón wi29.patron.mc2.14.1.1.1, a su vez, estarán a un cuarto de camino respecto de wi29.patron.14.1.1.1 o lo que es lo mismo a mitad de camino de wi29.patron.mc.14.1.1.1.

En la Tabla C.1, en la Tabla C.2 y en la Tabla C.3 se exhiben los experimentos analizados ($\surd$, $\times$) para cada uno de los algoritmos paralelos TSP usando 8, 16 y/o 32 nodos. El símbolo ($\surd$) significa que la ejecución se realizó con éxito y el símbolo ($\times$) que la ejecución tuvo un problema (overflow).

Tabla C.1: Experimentos analizados.

			Exhaustivo			**Poda local**			**Poda global**		
Clase	**Problema**	**#Ciudades**	**8**	**16**	**32**	**8**	**16**	**32**	**8**	**16**	**32**
	patron.12.1.1.1	15	$\surd$			$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	patron.9.2.2.2	15	$\surd$			$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
1	patron.6.3.3.3	15	$\surd$			$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	patron.3.4.4.4	15	$\surd$			$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	patron.7.8	15	$\surd$			$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	patron.13.1.1.1	16				$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	patron.10.2.2.2	16				$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
2	patron.7.3.3.3	16				$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	patron.4.4.4.4	16				$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	patron.8.8	16				$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	patron.14.1.1.1	17				$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	patron.11.2.2.2	17					$\surd$	$\surd$		$\surd$	$\surd$
3	patron.8.3.3.3	17					$\surd$	$\surd$		$\surd$	$\surd$
	patron.5.4.4.4	17					$\surd$	$\surd$		$\surd$	$\surd$
	patron.9.8	17					$\surd$	$\surd$		$\surd$	$\surd$
	wi29.patron.12.1.1.1	15	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	wi29.patron.9.2.2.2	15	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
4	wi29.patron.6.3.3.3	15	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	wi29.patron.3.4.4.4	15	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	wi29.patron.7.8	15	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	wi29.patron.15	15	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
	wi29.div_2.patron.12.1.1.1	15			$\surd$			$\surd$			$\surd$
	wi29.div_2.patron.9.2.2.2	15			$\surd$			$\surd$			$\surd$
4	wi29.div_2.patron.6.3.3.3	15			$\surd$			$\surd$			$\surd$
	wi29.div_2.patron.3.4.4.4	15			$\surd$			$\surd$			$\surd$
	wi29.div_2.patron.7.8	15			$\surd$			$\surd$			$\surd$
	wi29.div_2.patron.15	15			$\surd$			$\surd$			$\surd$

Tabla C.2: Experimentos analizados (continuación).

			Exhaustivo			**Poda local**			**Poda global**		
Clase	**Problema**	**#Ciudades**	**8**	**16**	**32**	**8**	**16**	**32**	**8**	**16**	**32**
	wi29.x3.patron.12.1.1.1	15			×			×			×
	wi29.x3.patron.9.2.2.2	15			×			×			×
4	wi29.x3.patron.6.3.3.3	15			×			×			×
	wi29.x3.patron.3.4.4.4	15			×			×			×
	wi29.x3.patron.7.8	15			×			×			×
	wi29.x3.patron.15	15			×			×			×
	wi29.patron.13.1.1.1	16				√	√	√	√	√	√
	wi29.patron.10.2.2.2	16				√	√	√	√	√	√
5	wi29.patron.7.3.3.3	16				√	√	√	√	√	√
	wi29.patron.4.4.4.4	16				√	√	√	√	√	√
	wi29.patron.8.8	16				√	√	√	√	√	√
	wi29.patron.16	16				√	√	√	√	√	√
	wi29.swap.patron.12.1.1.1	15	√	√	√	√	√	√	√	√	√
	wi29.swap.patron.9.2.2.2	15	√	√	√	√	√	√	√	√	√
6	wi29.swap.patron.6.3.3.3	15	√	√	√	√	√	√	√	√	√
	wi29.swap.patron.3.4.4.4	15	√	√	√	√	√	√	√	√	√
	wi29.swap.patron.7.8	15	√	√	√	√	√	√	√	√	√
	wi29.swap.patron.15	15	√	√	√	√	√	√	√	√	√
	coordenada_y0.patron.15	15	√	√	√	√	√	√	√	√	√
	coordenada_y0.patron.16	16	√	√	√	√	√	√	√	√	√
7	coordenada_y0.patron.17	17				√	√	√	√	√	√
	coordenada_y0.patron.18	18									√
	coordenada_y0.patron.19	19									√
	coordenada_y0.patron.20	20									√
	circulo.15	15			√	√	√	√	√	√	√
	circulo.16	16			√	√	√	√	√	√	√
	circulo.17	17			√	√	√	√	√	√	√
	circulo.18	18									√
8	circulo.19	19									√
	circulo.20	20									√
	circulo.21	21									√
	circulo.22	22									√
	circulo.23	23									√
	circulo.24	24									√
	circulo.x3.16	16									√
	circulo.x3.17	17									√
	circulo.x3.18	18									√
8	circulo.x3.19	19									√
	circulo.x3.20	20									√
	circulo.x3.21	21									√
	circulo.x3.22	22									√

Tabla C.3: Experimentos analizados (continuación).

			Exhaustivo			Poda local			Poda global		
Clase	**Problema**	**#Ciudades**	**8**	**16**	**32**	**8**	**16**	**32**	**8**	**16**	**32**
9	circulo.15.1000	16									√
	circulo.15.2000	16									√
	circulo.15.3000	16									√
	circulo.15.4000	16									√
10	wi29.patron.14.1.1.1	17				√					√
	wi29.patron.11.2.2.2	17									√
	wi29.patron.8.3.3.3	17									√
	wi29.patron.5.4.4.4	17									√
	wi29.patron.9.8	17									√
	wi29.patron.17	17									√
10	wi29.patron.mc.14.1.1.1	17									√
	wi29.patron.mc.11.2.2.2	17									√
	wi29.patron.mc.8.3.3.3	17									√
	wi29.patron.mc.5.4.4.4	17									√
	wi29.patron.mc.9.8	17									√
	wi29.patron.mc.17	17									√
10	wi29.patron.mc2.14.1.1.1	17									√
	wi29.patron.mc2.11.2.2.2	17									√
	wi29.patron.mc2.8.3.3.3	17									√
	wi29.patron.mc2.5.4.4.4	17									√
	wi29.patron.mc2.9.8	17									√
	wi29.patron.mc2.17	17									√

C.1. Resumen del volumen de información procesado

Como ya se expresó, el proceso de descubrir conocimiento, involucrando la técnica y el algoritmo de minería de datos como apoyo fundamental, fue una de las etapas más importantes que permitió plantear las suficientes y adecuadas hipótesis para el problema TSP. Por tal motivo, fue necesario procesar un gran volumen de experimentos con el fin de extraer conocimiento y al mismo tiempo generar una base de datos histórica.

Tanto la Figura C.1 como la Figura C.2 muestran el mismo volumen de experimentos pero desde dos perspectivas distintas.

La Figura C.1 exhibe la cantidad de experimentos ejecutados por algoritmo (exhaustivo, poda local y poda global) y número de nodos (8, 16 y 32). Para que las pruebas resultaran más significativas se trató de trabajar en todo momento con la máxima cantidad de nodos disponible. Asimismo, el algoritmo de poda global fue el más estudiado debido a que no

tiene un costo prohibitivo en cuanto al tiempo y no depende de factores externos como la asignación de las tareas.

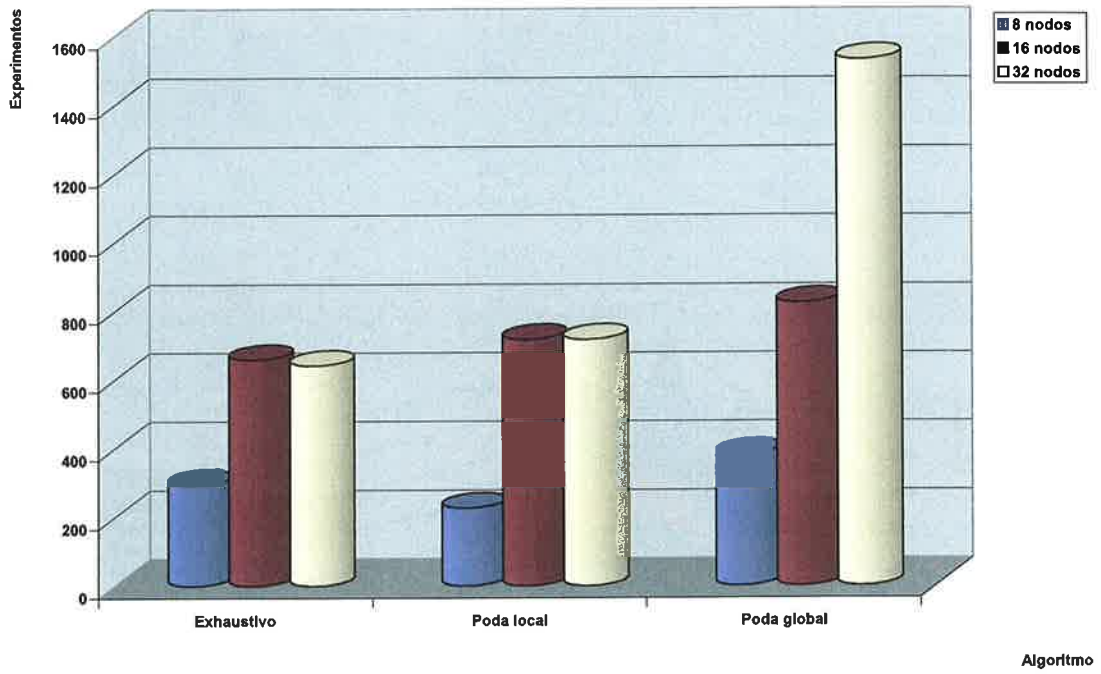


Figura C.1: Experimentos ejecutados por algoritmo y número de nodos.

La Figura C.2 grafica la cantidad de experimentos ejecutados de acuerdo con el número de ciudades (de 15 a 24) y el número de nodos (8, 16 y 32). Debido a que los tiempos de ejecución de los algoritmos TSP se van deteriorando a medida que el número de ciudades crece, se ha optado por considerar una adecuada cantidad de ciudades según el objetivo deseado. Asimismo, es un error pensar que basta esperar algunos años para que ciertos algoritmos costosos se puedan ejecutar en un tiempo razonable.

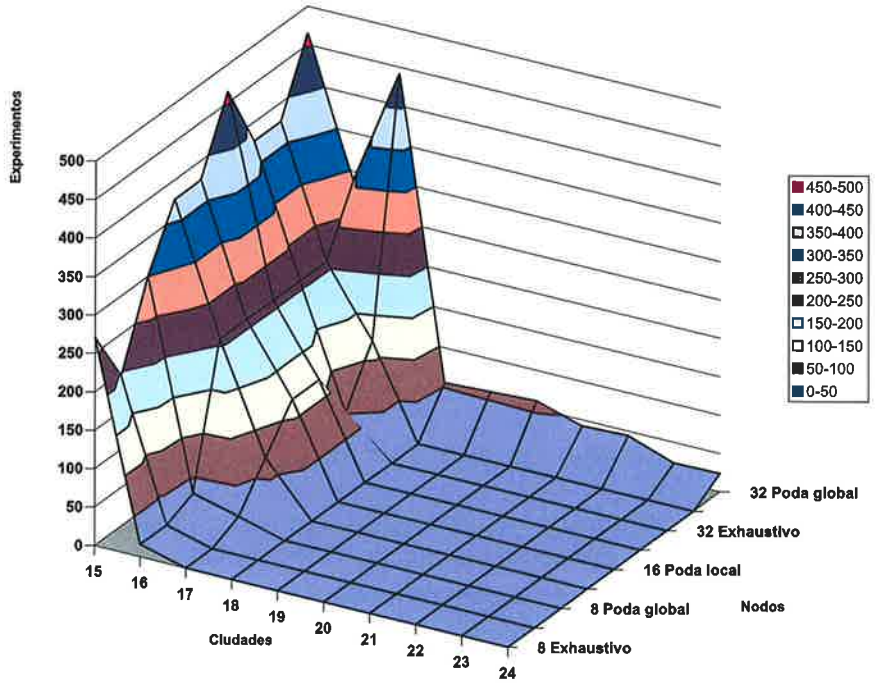


Figura C.2: Experimentos ejecutados por número de ciudades y número de nodos.

Bibliografía

[1] Overview of Computational Science. The shodor education foundation, inc. http://www.shodor.org/chemviz/overview/compsci.html, 2007.

[2] R. Jain. *The Art of Computer Systems Performance Analysis. Techniques for Experimental Design, Measurement, Simulation, and Modeling.* John Wiley & Sons, Inc., 1991.

[3] X. Molero, C. Juiz, and M. Rodeno, editors. *Evaluación y Modelado del Rendimiento de los Sistemas Informáticos.* Prentice Hall, 2004.

[4] S. Skiena. *The algorithm design manual.* Springer-Verlag New York, Inc., New York, NY, USA, 1998.

[5] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations.* John Wiley & Sons, Inc., New York, NY, USA, 1990.

[6] R. Geraerts and M.H. Overmars. Reachability analysis of sampling based planners. *Proceedings of the 2005 IEEE, International Conference on Robotics and Automation*, 2005.

[7] TSP page. http://www.tsp.gatech.edu/, 2007.

[8] P.C. Gilmore and R.E. Gomory. Sequencing a one-state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12:655–679, 1964.

[9] F. Alizadeh, R. Karp, Newberg, and D. Weisser. Physical mapping of chromosomes: A combinatorial problem in molecular biology. *Symposium on Discrete Algorithms*, ACM Press:371–381, 1993.

[10] L. Svobodova. Performance measurement and evaluation methods: Analysis and applications. *Elsevier Computer Science Library, Computer Design and Architecture Series. New York: Elsevier*, 1976.

[11] H. Gautama and A. van Gemund. Low-cost static performance prediction of parallel stochastic task compositions. *IEEE Transactions on Parallel and Distributed Systems*, 17(1), 2006.

[12] V.S. Adve and M.K. Vernon. The influence of random delays on parallel execution times. *ACM SIGMETRICS'93*, pages 61–73, 1993.

[13] E. Cinlar. Introduction to stochastic processes. *Prentice-Hall, Englewood Cliffs, NJ*, 1975.

[14] K. Jensen. Coloured petri nets: A high-level language for system design and analysis. *in High level Petri Nets: Theory and Application. Springer Verlag.*, pages 44–122, 1991.

[15] Meyer J.F., Movaghar A., and Sanders W.H. Stochastic activity networks: Structure, behavior, and application. *Proc. of the International Conf. on Timed Petri Nets, Torino.*, pages 106–115, 1985.

[16] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77:541–580, 1989.

[17] E.D. Lazowska, J. Zahorjan, G. Scott Graham, and K.C. Sevcik. Quantitative system performance: Computer system analysis using queueing network models. *Prentice-Hall, Englewood Cliffs*, 1984.

[18] K.S. Trivedi. Probability and statistics with reliability, queuing, and computer science applications. *John Wiley and Sons*, 2001.

[19] H. Hermanns, U. Herzog, and J.P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274(1-2):43–87, 2002.

[20] U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with the tipptool. *Performance Evaluation*, 39(1-4):5–35, 2000.

[21] E. Gelenbe, E. Montagne, R. Suros, and C. M. Woodside. A performance model of block structured parallel programs. In *Proceedings of the international workshop on Parallel algorithms & architectures*, pages 127–138, Amsterdam, The Netherlands, The Netherlands, 1986. North-Holland Publishing Co.

[22] H. Jonkers. Performance analysis of parallel systems: A hybrid approach. *Ph.D. thesis, Delft University of Technology*, 1995.

[23] J. Simon and J.M. Wierum. Accurate performance prediction for massively parallel systems and its applications. *Proc. of the European Conf. on Par. Proc. (EuroPar'96)*, LNCS 1124, 675-688.

[24] M. Pahud. Ip3t a performance prediction software tool for irregular parallel applications. *IEEE International Performance, Computing and Communications Conference Proceedings*, 2000.

[25] E. Duman. Precedence constrained tsp arising in printed circuit board assembly. *International Journal of Production Research*, 42(1):67–78, 2004.

[26] R.E. Bland and D.F. Shallcross. Large traveling salesman problem arising from experiments in x-ray crystallography: a preliminary report on computation. *Technical Report No. 730, School of OR/IE, Cornell University, Ithaca, New York*, 1987.

[27] H.D. Ratliff and A.S. Rosenthal. Order-picking in a rectangular warehouse: A solvable case for the traveling salesman problem. *PDRC Report Series No. 81-10. Georgia Institute of Technology, Atlanta, Georgia*, 1981.

[28] D. Sankoff and M. Blanchette. The median problem for breakpoints in comparative genomics. *Proceedings of the 3rd Annual International Conference on Computing and Combinatorics (COCOON'97), London, UK*, Springer-Verlag:251–264, 1997.

[29] C. Korostensky and G.H. Gonnet. Using traveling salesman problem algorithms for evolutionary tree construction. *BIOINF: Bioinformatics*, 16, no. 7:619–627, 2000.

[30] D. Miller and J. Pekny. Exact solution of large asymmetric traveling salesman problems. *Science*, 251:754–761, 1991.

[31] B.L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

[32] A. Barvinok, A. Tamir, S.P. Fekete, G.J. Woeginger, D.S. Johnson, and R. Woodroofe. The geometric maximum traveling salesman problem. *Journal of the ACM*, 50(5):641–664, 2003.

[33] E.M. Arkin, Y.J. Chiang, J.S.B. Mitchell, S.S. Skiena, and T.C. Yang. On the maximum scatter tsp. *In Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 97). ACM New York*, pages 211–220, 1996.

[34] N. Christofides. Vehicle routing. *The Traveling Salesman Problem, Lawler, Lenstra, Rinooy Kan and Shmoys, eds., John Wiley*, pages 431–448, 1985.

[35] E. Argollo. Performance prediction and tuning in a multi-cluster environment. *PhD Thesis, Universitat Autonoma de Barcelona, Spain*, 2006.

[36] E. Argollo, A. Gaudiani, D.I. Rexachs, and E. Luque. Tuning application in a multi-cluster environment. *Lecture Notes in Computer Science (EuroPar 2006)*, 4128:78–88, 2006.

[37] J.B. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press*, 1:281–297, 1967.

[38] C.D. Polakoff and P. Fritzsche. Cluster-frame ambiente para clustering: Algoritmo k-means++. *Thesis of Licenciature in Computer Science. Buenos Aires University*, 2000.

[39] R. Miller and L. Boxer. *Algorithms Sequential and Parallel: A Unified Approach.* Charles River Media. Computer Engineering Series, 2005.

[40] R. Floyd. Nondeterministic algorithms. *J. ACM*, 14(4):636–644, 1967.

[41] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity.* Elsevier and MIT Press, 1990.

[42] D. Ding-Zhu and K. Ker-I. Review of theory of computational complexity. *SIGACT News*, 34(3):11–12, 2003.

[43] N.L. Biggs, E.K. Lloyd, and R.J. Wilson. Graph theory 1736-1936. *Clarendon Press, Oxford*, 1976.

[44] A. Schrijver. On the history of combinatorial optimization (till 1960). *CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands*, 1990.

[45] Clay Mathematics Institute. http://www.claymath.org/, 2007.

[46] G. Gutin and A. P. Punnen. *The Traveling Salesman Problem and Its Variations.* Springer, 2006.

[47] C.H. Papadimitriou. The euclidean traveling salesman problem is np-complete. *Theorical Computer Science*, 4:237–244, 1977.

[48] M.R. Garey, R.L. Graham, and D.S. Johnson. Some np-complete geometric problems. *STOC '76: Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 10–22, 1976.

[49] R.M. Karp. Reducibility among combinatorial problems: In complexity of computer computations. *Plenum Press, New York*, pages 85–103, 1972.

[50] G. Gutin and A.P. Punnen. Traveling salesman problem and its variations. *Kluwer Academic Publishers*, 12, 2002.

[51] L. Wang, A. Maciejewski, H. Siegel, and V. Roychowdhury. A comparative study of five parallel genetic algorithms using the traveling salesman problem. *Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS SPDP'98)*, pages 345–349, 1998.

[52] H.K. Tsai, J.M. Yang, and C.Y. Kao. Solving traveling salesman problems by combining global and local search mechanisms. *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*, 2:1290–1295, 2002.

[53] J. Pepper, B. Golden, and E. Wasil. Solving the traveling salesman problem with annealing-based heuristics: a computational study. *IEEE Transactions on Man and Cybernetics Systems, Part A*, 32(1):72–77, 2002.

[54] M. Zachariasen and M. Dam. Tabu search on the geometric traveling salesman problem. *In I. H. Osman and J. P. Kelly, editors, Metaheuristics: theory and applications (Proceedings from Metaheuristics International Conference Colorado)*, pages 571–587, 1995.

[55] N. Aras, I. Altinel, and J. Oommen. A kohonen-like decomposition method for the euclidean traveling salesman problem-knies/spl i.bar/decompose. *IEEE Transactions on Neural Networks*, 14(4):869–890, 2003.

[56] M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.

[57] K. Hwang and Z. Xu. Scalable parallel computing. *McGraw-Hill*, 1998.

[58] R. Buyya. High performance cluster computing. programming and applications. *Prentice Hall PTR*, 2, 1999.

[59] Barry Wilkinson and Michael Allen. *Parallel programming: techniques and applications using networked workstations and parallel computers.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[60] B.W. Kerninghan and D.M. Ritchie. El lenguaje de programación c. *Prentice Hall Hispanoamerica*, 1991.

[61] A. Grama, A. Gupta, G. Karypis, and V. Kumar. Introduction to parallel computing. *Addison-Wesley*, 2, 2003.

[62] P.C. Fritzsche, D. Rexachs, and E. Luque. A computational approach to tsp performance prediction using data mining. *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, pages 252–259, 2007.

[63] P.C. Fritzsche, D. Rexachs, and E. Luque. A knowledge-based methodology to predict performance order of data-dependent applications. *2007 International Conference on Data Mining, Las Vegas, Nevada, USA*, submitted for publication.

[64] M. Cho and D. Mount. Improved approximation bounds for planar point pattern matching. *Lecture Notes in Computer Science*, 3608:432–443, 2005.

[65] M. Goodrich, J. Mitchell, and M. Orletsky. Practical methods for approximate geometric pattern matching under rigid motion. *In Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 103–112, 1994.

[66] H. Alt and L. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. *Technical Report 96-11. Institut fur Informatik, Freie Unversitat Berlin, Berlin, Germany*, 1996.

[67] R. Veltkamp. Shape matching: Similarity measures and algorithms. *Technical Report UU-CS 2001-03, Utrecht University: Information and Computing Sciences, Utrecht, The Netherlands*, 2001.

[68] D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, September 1993.

[69] P. Heffernan and S. Schirra. Approximate decision algorithms for point set congruence. *Comput. Geom. Theory Appl.*, 4:137–156, 1994.

[70] L. Chew, M. Goodrich, D. Huttenlocher, K. Kedem, J. Kleinberg, and D. Kravets. Geometric pattern matching under euclidean motion. *Comput. Geom. Theory Appl.*, 7:113–124, 1997.

[71] R. Groth. Data mining: a hands-on approach for business professionals. *Prentice Hall PTR*, 1998.