Heurísticas para el TSP-2D Euclideo y Simétrico Basadas en la Triangulación de Delaunay y sus Subgrafos

Natalio Krasnogor [1],[2]
Directores: Pablo Moscato[3] *y Gabriel Baum*

[1] natk@sol.info.unlp.edu.ar
[2] http://www-lifia.info.unlp.edu.ar/~ natk
[3] http://www.densis.fee.unicamp.br/~ moscato/pmoscato_home.html

Indice

0.1 Dedicatorias y Agradecimientos

La comprensión humana no es simple luz sino que recibe infusión de la voluntad y los afectos.

F.Bacon

I get by, with a little help, with a little help from my friends.

The Beatles

0.1.1 Dedicatorias

Recuerdo vívidamente el día en que me inscribí en la Universidad Nacional de La Plata. Cientos de sueños y expectativas iluminaban ese momento. Otros, infinitamente menos afortunados que yo, perdían para siempre los suyos. Había estallado una bomba en la Embajada de Israel en Argentina. Y el horror se repitió unos años después en la sede de AMIA. Muchos de los sueños y expectativas que me guiaron hasta aquí han sido alcanzados. Aquellos de los muertos nunca podrán siquiera ser. Y lo peor es que todavía no tenemos respuestas.

Dedico esta tesis a la memoria de cada ser humano muerto en aquellos atentados.

Lo colectivo no tiene razón de ser si no es a partir de lo individual, esta tesis esta dedicada, por sobre todas las cosas, a la memoria de mi bobe, Taba Krasnogor, y de mi tío, Alberto Feldman.

0.1.2 Agradecimientos

Cuando en el año 90 comencé mis estudios de Ingeniería Eléctrica en la Universidad Nacional de Tucumán no podía imaginar donde llevaría el sendero que años atrás había comenzado con mi primer lectura: "El Tigre de la Malasia" de Emilio Salgari; mi abuela Adela fué quién me regaló aquel libro. Hoy, ocho años más tarde, terminando mi Licenciatura en Informática en el Departamento de Informática de la Universidad Nacional de La Plata, puedo comprender perfectamente lo imposible de anticipar en aquel entonces las curvas de camino.

Estoy convencido que ningún ideal u objetivo deja de ser "solo" "un sueño Jázaro" sino es por causa de la gente que nos acompaña. Toda la gente que conocí en estos años y me ayudo de una u otra manera en la concreción de mis metas no es suceptible de calificativos como buena o mala, incapaz o inteligente. Solo se me ocurre un adjetivo... IMPORTANTE. Ustedes viven en mí como gente importante.

Agradesco a:

Mis profesores y amigos del Departamento de Computación de la UNT quienes me permitieron siendo alumno de 2^{do} año de Eléctrica cursar las optativas que dictaban para la Licenciatura en Matemáticas. Fueron ellos quienes acertadamente me orientaron a elegir la UNLP para continuar mis estudios.

Al Ingeniero A. Quijano por haberme permitido trabajar en el CeTAD durante los años 94 y 95, brindándome la posibilidad de acceder a hard y soft al cuál normalmente no hubiese tenido acceso. Fué allí en el CeTAD donde conocí a M. Norman. Mike fue el primer *Phd* con el que pude interactuar. De el aprendí, entre otras cosas, que las chicas platenses "se pintan" los pantalones y que con suficiente café, cualquier algoritmo puede hacerse funcionar en $N * Log(N)$. Fué también el primero en hacerme una muy conceptuosa carta de recomendación, antes aún de habersela yo pedido, para que la tuviese "just in case".

A Priscila Nieman quien estuvo a mi lado la primera mitad de mi carrera.

A Gustavo Rossi con quien las discusiones y diferencias de opinión han sido muchas, pero con quién comparto el ideal de ver a nuestro laboratorio sano y populoso en ideas, ciencia y estudios. El L.I.F.I.A. ha sido mucho más que mi lugar de trabajo en los últimos años, porque me llevo los problemas del laboratorio a casa y también viceversa. Por eso mi gratitud es grande y la hago extensiva a todos los miembros el L.I.F.I.A. .

A toda la gente del L.I.F.I.A. $-\lambda$ por lo bueno y lo malo del trabajo cotidiano. En especial a la gente del grupo BioCom, Wanda, Germán, Esteban, Germán Esteban, Walter, el Colo y Fidel porque nos divertimos, discutimos y trabajamos juntos.

El más profundo agradecimiento a Vero, Majo, Fer Lyardet y Fabio porque como amigos siempre supieron ser críticos cuando así las circunstancias lo requerían y no muy críticos cuando me enojo porque me critican.

A David Pelta por su amistad y ese humor con el que mira la vida; no por los trabajos realizados con mutuo esfuerzo, sino por todos los que todavía faltan realizar.

A mis Directores; y no digo "solo" directores de tesis. Gabriel Baum y Pablo Moscato han sido motivo de inspiración y dolores de cabeza. De Gabriel aprendí lo importante del escepticismo científico, de Pablo la pasión por la ciencia. De ambos aprendí el método científico...se puede pedir algo más de un director? Yo creo que no.

A Pablo le debo también el conocimiento de la ensalada de Rúcula, la certeza de que "con una cucharita es suficiente" y de innumerables cenas en familia. A través tuyo Pablo quiero agradecer también a Cesar, Raul y Elva.

A Natalia Romero quien una vez más, ahora que trato de expresar solo un poco de mis sentimientos, vuelve a dejarme sin palabras. Gracias por estar cerca Natita... nada hubiese sido igual sin vos.

Finalmente quiero agradecer a mi familia, a mi Papá, mi Mamá y mi Hermano. De mis padres no solo heredé los genes, que pueden haber sido buenos o malos quien sabrá?!. Heredé tambien sus memes, es decir, sus ideas, sus valores, sus ejemplos y sus concejos y sé a ciencia cierta que fueron buenos. Sin ellos yo, simplemente, no sería yo. A mi Hermano quién tal vez sea la persona de la que más aprendí. Porque por sobre todo, me enseñó sobre mi persona, lo importante de las diferencias, y que "no es fácil gatito" pero que aún así vale la pena y se puede.

A todos ustedes gracias.

Ah! me olvidaba..., gracias a Frankie, "la Pantera Rosa" y a todos los que aportaron los acentos de esta tesis.

Capítulo 1

Introducción

Hasta lo que pensamos podría estarlo pensando él tambien

Remordimiento por cualquier defunción
J.L. Borges

1.1 Objetivo General del Trabajo

El objetivo de esta tesis es el desarrollo de nuevas heurísticas para el *Traveling Salesman Problem*, *TSP* [26] en adelante, mediante el estudio de estructuras geométricas discretas basadas en la triangulación de Delaunay y sus subgrafos[25].

Dichas heurísticas deberán proporcionar soluciones factibles a grandes instancias euclideas del *TSP* en el plano. Las mismas poseerán baja complejidad computacional y las soluciones que encuentren serán comparadas empíricamente con las encontradas por otros algoritmos existentes en la literatura.

Para llevar a cabo esta tarea se incursionará en temas de complejidad computacional, teoría de grafos, geometría computacional y estructuras de datos, convergiendo estos en la más amplia y multidisciplinaria optimización combinatoria.

1.2 El TSP en relación a otros problemas

El *Problema del Viajante De Comercio*, *TSP* , es un representante "importante" de la clase de problemas NP-Completos. Decimos importante ya que, a pesar de su simple formulación, todavía no se ha encontrado un algoritmo que lo resuelva en forma exacta en tiempo polinomial [15], siendo tal vez el más extensamente estudiado de su grupo. Además, a menos que P=NP, no debería esperarse tal algoritmo.

Debido a las innumerables situaciones prácticas donde se presenta este problema es que tiene sentido investigar el desarrollo de heurísticas rápidas para resolver instancias grandes. El estudio de heurísticas y su performance para grandes instancias esta motivado científica y tecnológicamente. Actualmente, los procesos de desarrollo de circuitos VLSI involucran problemas de *min TSP*

de 10^5 "ciudades" y se espera que pronto se necesiten algoritmos que den soluciones aproximadas en tiempos razonables para instancias de 10^7 o aún mayores. En adelante, cuando se haga referencia al tamaño de una instancia del *TSP* , nos estaremos refiriendo a la cantidad de puntos o ciudades a recorrer; generalmente se denotará esa cantidad con el número N. Desde hace cierto tiempo se conocen instancias del *TSP* del orden de 10^4 ciudades o nodos, estas instancias surgen de problemas de soldado de circuitos y cristalografía de rayos X. Asimismo, soluciones a este problema tienen impacto en problemas de biología molecular, principalmente en estudios conformacionales y de alineacion de secuencias, ver por ejemplo [19],[31], como así también en optimización de "queries" en grandes sistemas hipermediales [22],[23].

Es habitual encontrar en la literatura experimentos computacionales donde para probar nuevos métodos se recurre a problemas que no han sido estudiados en profundidad, y a veces ni siquiera se conoce cual es su complejidad computacional. Creemos que para realizar estudios empíricos del comportamiento de heurísticas y/o metaheurísticas se deben utilizar problemas de optimización combinatoria bien conocidos, como es el caso del *TSP*. El *Problema Del Viajante De Comercio* puede enunciarse asi:

> " Dadas N ciudades $(c_1, c_2, \ldots, c_N)$ y una matriz de distancias entre las mismas $D : N * N$; $D_{i,j} \in \mathcal{Z}^+$ (siendo $D_{i,j}$ la distancia entre la ciudad c_i y la ciudad c_j), el objetivo es encontrar un "tour", es decir una permutacion $(c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(N)})$ de las N ciudades, que minimice la longitud total que se define como:
>
> $$L(Tour) = \sum_{k=0}^{k=N-1} D_{Tour(k),Tour(k+1)} \qquad (1.1)$$
>
> donde $Tour(N)$ se identifica con $Tour(0)$ de tal forma de cerrar un ciclo."

Los estudios sobre el *TSP* vienen de larga data; mencionaremos a continuacion algunas referencias y hechos interesantes.

En [14] se introdujo un algoritmo de aproximación para el viajante de comercio euclideo con una performance del peor caso garantizada en $\sqrt{2 * N} + 1.75$ cuando los puntos se encuentran distribuidos en un cuadrado de superficie unidad.

Los métodos de mejoras iterativas de S. Lin, $edge - exchange$ [27], conocidos como $r - opt$ y los posteriores $r - opt$ *variables* desarrollados por el mismo Lin y Kernighan, generalmente devuelven configuraciones proximas al óptimo [28].

En el año 1976 N. Christofides presentó un algoritmo basado en *matching de pesos mínimos* cuya complejidad es de $O(N^3)$ y las soluciones que genera son a lo sumo 50% más largas que el tour óptimo [12].

Un año más tarde R. Karp presenta su algoritmo de *patching*, cuyo error esperado tiende a cero a medida que N tiende a infinito. Luego extendió su trabajo para el *ATSP* , *viajante de Comercio Asimétrico*. Gran cantidad de instancias del *TSP* han sido resueltas a optimalidad y pueden encontrarse en la literatura, por ejemplo, [24], [28], [34], [36]. A traves de *Internet* se puede

acceder a *TSPLIB*[1] , una base de datos que contiene gran cantidad de instancias del *TSP* y del *ATSP* resueltas a optimalidad mediante métodos exactos en supercomputadores. Así mismo, *TSPBIB*[2] [40] es quizás la base de datos on-line mas actualizada en lo referente a trabajos del *TSP* y problemas afines. Moscato y Norman [32] desarrollaron un método de construcción de instancias arbitrariamente grandes, con "patrones" específicos y óptimo global conocido.

Beardwood, Halton y Hammersley presentan en 1959 la siguiente fórmula

$$K = \lim_{N \to \infty} \frac{L_{opt}(N, R)}{\sqrt{NR}} \tag{1.2}$$

que permite calcular el valor asimptótico de la longitud del tour óptimo en el caso de que las ciudades se encuentren ubicadas con distribución aleatoria uniforme en un rectángulo de R unidades de área [3]. Varios investigadores han tratado de establecer el valor de la constante K; por ejemplo, D. Stein en 1977 presenta cotas empíricas que arrojaron los siguientes resultados:

$$0.765 \leq K \leq 0.765 + \frac{4}{N} \tag{1.3}$$

E. Bonomi y J.L. Lutton obtiene para K el valor 0.749 cuando $N \to \infty$ [8]. D.S. Johnson obtiene una cota mínima para la longitud del tour de 71.5 y conjetura que el tour óptimo tiene una longitud de 72.5 cuando $N = 10000$.

En [32] se estudia una instancia fractal del *TSP* que presenta para K un comportamiento similar, ver [41].

Moscato y Norman en [30] presenta un algoritmo memético para la resolución paralela de instancias del *TSP* .

El trabajo de Reinelt [37] muestra un estudio detallado de la utilización en tandem de heurísticas clásicas como $2-opt$, *nearest neighbor* y *Lin-Kernighan*, luego de obtener un tour factible a partir de un grafo "vecindario", GV. GV es un **supergrafo** de la triangulación de Delaunay o del grafo de Delaunay. Es en [37] donde esta tesis encuentra su origen y motivación. GV de grado k, GV_k, se define en el trabajo mencionado de la siguiente forma:

Sea S un conjunto de ciudades a visitar, y $TD(S)$ la triangulación de Delaunay de dicho conjunto, entonces, GV_k tiene el mismo conjunto de vértices (ciudades) S y el conjunto de aristas se define por:
$E = \{(u, v) | u, v \in S, \exists C_{TD(S)}(u, v, k)\}$, es decir u estará conectado a v en GV_k si existe en TD(S) un camino C de longitud a lo sumo k desde u a v.

Luego, se construye a partir de GV_k un tour factible que más tarde será optimizado con alguna heurística tradicional de las mencionadas mas arriba.

A diferencia del estudio realizado por Reinelt, investigaremos sobre el uso de algunos **subgrafos** de $TD(S)$ con propiedades geométricas conocidas. Si nuestros resultados resultan comparables a los de la utilización de supergrafos de $TD(S)$, debiésemos entonces optar por los primeros por ser estos menos costosos en espacio y construibles todos en $O(N * LogN)$ con $N = |S|$; caso contrario el enfoque de Reinelt resultaría beneficiado.

[1] TSPLIB es compilada y mantenida por Gerhard Reinelt, E-Mail Gerhard.Reinelt@IWR.Uni-Heidelberg.DE

[2] TSPBIB es compilada y mantenida por Pablo A. Moscato, E-Mail moscato@densis.fee.unicamp.br

1.3 Notas:

Capítulo 2

Complejidad Computacional, un Resumen

Pregunta:
Que hacer para no perder el tiempo?
Respuesta:
Sentirlo en toda su magnitud.
Medios:
...oir conferencias en una lengua que no se conoce, escoger los itinerarios del tren mas largos y menos cómodos y viajar de pie.

La Peste
A. Camus

2.1 Introducción

En esta sección se hará un repaso de algunos conceptos fundamentales de complejidad computacional en particular y optimización combinatoria en general. Presentaremos las definiciones de NP-completitud, reducciones y se enunciará el teorema de Cook. Como ejemplo mostraremos que el problema del ciclo hamiltoniano es NP-completo.

Informalmente, los problemas de optimización combinatoria son aquellos para los cuales el espacio de soluciones factibles es finito pero muy grande; donde cada solución factible es compatible con las restricciones específicas de cada problema pero no necesariamente óptimas.

Asociado con cada problema de "decisión" NP-Completo existe uno de "optimización" NP-Completo. La versión de optimización consta de una función de valoración a ser minimizada o maximizada. Esta función recibe diversos nombres de acuerdo a la disciplina donde se la estudie, por ejemplo función objetivo, de valoración, de fitness, etc. La cantidad de configuraciones posibles crece por lo menos exponencialmente en este tipo de problemas al aumentar el tamaño de

la instancia; es un camino yermo tentar algoritmos exhaustivos en estos caso, excepto por supuesto, cuando de instancias muy pequeñas se trata.

2.2 Algunos Conceptos de Complejidad Computacional

En esta sección se repasaran, entre otros, los conceptos de Máquina de Turing (MT en adelante), problemas de decisión, NP-Completitud, el teorema de Cook, reducibilidad, etc. Presentaremos algunos teoremas y lemas bien conocidos de la literatura a fin de completar la exposición. Sus respectivas pruebas pueden encontrarse en cualquier texto de complejidad computacional, por ejemplo [65], [46], [15].

2.2.1 Definición de Máquina de Turing

A continuación repasaremos el concepto de Máquina de Turing debido a que mas adelante necesitaremos recurrir al mismo con el objetivo de probar que algunos problema son intrínsecamente "más difíciles" que otros. Asimismo, enunciaremos un teorema que permite mostrar que esta clase de problemas, la de los difíciles, no es vacía ya que SAT pertence a la misma.

Una MT consiste de:

- Una cinta doblemente infinita dividida en celdas, que pueden considerarse numeradas como $\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$.
- Un alfabeto $\Sigma = \{0, 1, \lambda\}$.
- Una cabeza sobre la cinta capaz de leer un caracter simple de la cinta o bien escribir uno, y además puede moverse una celda en alguna de ambas direcciones.
- Una lista finita de estados, tal que en cada instante la máquina esta en uno y solo uno de ellos. Los estados posibles de MT son, ante todo, los estados regulares $q_1, \ldots, q_s$, y ademas, tres especiales $q_0 = START$, $q_Y = FINAL_{ACCEPTING}$, $q_N = FINAL_{REJECTING}$.
- Un programa que dirije a MT a través de los pasos de una computación en particular, toma un estado y un símbolo y retorna un nuevo estado, un nuevo símbolo y la especificación del movimiento de la cabeza lectora.

Formalmente definimos MT como:

Definición 2.1 *$T = (k, \Sigma, \Gamma, \alpha, \beta, \gamma)$ donde $k \geq 1$ es un número natural (arriba consideramos $k = 1$), que especifica la cantidad de cintas de MT. Σ y Γ conjuntos finitos. Σ es el conjunto de los símbolos disponibles en la cinta, con $\lambda \in \Sigma$. Γ es el conjunto de estados, $START, STOP \in \Gamma$, además*
$\alpha \mid \Gamma \times \Sigma^k \mapsto \Gamma$,
$\beta \mid \Gamma \times \Sigma^k \mapsto \Sigma^k$,
$\gamma \mid \Gamma \times \Sigma^k \mapsto \{-1, 0, 1\}^k$
son mapeos arbitrarios. Donde 'α' genera un nuevo estado, 'β' el símbolo a ser

impreso en la cinta y 'γ' determina cuanto es el movimiento de la cinta. 'Σ' esta formado por $\{0, 1, \lambda\}$.

En la definición de arriba hemos considerado que los alfabetos de entrada y salida son el mismo, siendo sencillo formalizar el caso en que esto no ocurre así. Si bien hemos definido algunos estados como "especiales" los mismos no son necesarios si MT es *transductora* pero si en el caso de ser *reconocedora*. Las definiciones de arriba, ver [65], son con el solo objeto de introducir los conceptos al lector, y han sido simplificados para nuestro propósito. Para un tratamiento clásico del tema ver [46].

2.2.2 Problemas de Decisión

Un problema de decisión, PD en adelante, es un problema donde la salida es solo *SI* o *NO*. Por ejemplo, dado un grafo $G = (V, E)$ un problema de decisión asociado al mismo es verificar si puede ser coloreado con k colores, o si existe un tour T con $longitud(T) \leq L$ donde toda arista de T pertenece a E. Todo PD puede ser asociado con un problema de optimización, PO en lo que sigue. Para los ejemplos de arriba:
Cual es el menor número cromático de $G = (V, E)$? , o cual es el tour T más corto para G?

Si podemos resolver "rápido" un PD, entonces, con un poco más de esfuerzo podremos resolver "rápido" el PO asociado. Por ejemplo, supongamos que contamos con un algoritmo polinomial para verificar si un grafo puede ser coloreado con k colores. Entonces, podremos extenderlo para buscar el menor k de la siguiente manera:

1. Se inicializa k, esto es, $k =\mid V \mid$. Así, hemos asignado a cada vértice de G un color distinto, lo que genera el coloreo trivial.

2. Verificamos en tiempo polinómico si G puede ser $k - coloreado$.

3. En caso afirmativo, guardamos $k(old_k = k)$, asignamos a k el valor $\lfloor k/2 \rfloor$, y volvemos al punto 2.

4. Caso contrario, si $k > old_k$ obtuvimos lo buscado y seguimos en 5, sino hacemos $k = k + \lfloor \frac{old_k - k}{2} \rfloor$, y vamos al punto 2.

5. Devolver old_k, siendo este el menor número cromático de G.

6. Fin.

El algoritmo de arriba trabaja en tiempo $O(log(\mid V \mid) * q(\mid G \mid))$, donde $q(\mid G \mid)$ es un polinomio en el tamaño del grafo. De esta manera fuimos capaces de resolver un PO con un poco más de tiempo de cómputo, $O(log(\mid V \mid))$ veces $q(\mid G \mid)$, a partir de la solución al PD asociado [1].

[1] Desafortunadamente tal solución al coloreo de grafos no existe

2.2.3 Lenguajes

Todo PD tiene una respuesta *SI* o *NO*, de tal forma que se puede considerar que un PD consiste en decidir si una dada palabra x sobre Σ pertenece a L. L es el conjunto de todas las palabras para las cuales la respuesta es *SI*. En el contexto de este capítulo, la palabra x es solo una "buena" codificación de alguna instancia I de un problema Π. Para el PD asociado con Π, la respuesta para I será *SI* cuando su codificación x pertenezca al lenguaje L.

2.2.4 Nociones Sobre las Clases P y NP

Un PD Π pertenece a P(polinomial) si existe un algoritmo A y una constante c tales que: $\forall I \in \Pi$, A es $O(B^c)$ donde B es el número de bits necesarios para representar la instancia I, $B = \mid x \mid$. En resumen, los PDs que pertenecen a P son "fáciles" de resolver.

Por otro lado, un PD pertence a la clase NP si existe un algoritmo A que verifica:

1. Asociado con cada palabra del lenguaje Q existe un certificado $C(I)$, y cuando $(I, C(I))$ es dado como entrada para A este reconoce que $I \in Q$.

2. Si I no esta en Q entonces no existe ningún $C(I)$, de forma que A no reconoce I.

3. A es un algoritmo de tiempo polinomial.

Podemos ver que P $\subset$ NP ya que $\forall \Pi \in P, \forall I \in \Pi, C(I) \neq \emptyset$. Una definición mas rigurosa puede encontrarse en [46].

Dado un problema Π_1, podemos decir que pertenece a la clase de los problemas NP si puede ser resuelto por un algoritmo **no determinístico** que realiza su trabajo en a lo sumo tiempo polinomial en el tamaño del problema. Es obvio que la clase P de los algoritmos polinomiales determinísticos esta incluida en NP, P $\subseteq$ NP. Seguramente, si la inclusión es propia es una de las preguntas más importantes de la teoría de la complejidad ($P = NP?$). Diremos que Π_1 es transformable polinomialmente a otro problema Π_2, $\Pi_1 \prec_p \Pi_2$, si existe una función f que mapea las instancias de Π_1 en instancias de Π_2, y se verifica lo siguiente:

- f es polinomial y determinística.

- Una solución a la instancia $f(I)$ de Π_2 puede ser convertida en una solución para la instancia I de $\Pi_1 \forall I$.

En otras palabras, esto nos dice que si contamos con el algoritmo mas rápido para resolver Π_2, entonces el tiempo necesario para alcanzar una solución para Π_1 es como mucho el tiempo necesario para solucionar Π_2 mas un término polinómico. Tenemos entonces que **complejidad(Π_1)** $\leq$ **complejidad(Π_2)** + **Polinomio**. En lo que sigue se entenderá que las transformaciones serán polinomiales y se usará $\prec$ en vez de $\prec_p$.

2.2.5 Completitud en NP

Un problema Π es NP-Hard si las siguientes condiciones se satisfacen:

- $\Pi\prime \prec \Pi$, $\forall \Pi\prime \in NP$.
- $\Pi \in P \Rightarrow P = NP$.

Si existe un algoritmo determinístico polinomial para Π entonces existe uno para cualquier otro problema en NP. Ahora podemos definir que es un problema NP-Completo:

Un problema Π es NP-Completo si pertenece a la clase de los NP y además es NP-Hard. Existen algunas relaciones importantes entre P y NP [15]. Como ya se dijo antes, un PD Π es NP-Completo si pertenece a NP y todo problema en NP es polinomialmente reducible a él. En otras palabras, un problema que es "mayor" que cualquier otro problema NP, esto es un problema al cual cualquier otro en NP puede ser reducido, se llama NP-Hard. Si un problema NP-Hard es además NP se dice NP-Completo. Entonces, todos los problemas NP-Completos son igualmente difíciles de resolver pues son reducibles entre sí.

2.2.6 Reductibilidad Entre Problemas

Enunciaremos ahora algunos hechos que ayudaran a comprender algunas relaciones entre la clase P y NP.

Teorema 2.1 *Si $\Pi \in NP$, entonces existe un polinomio p tal que Π puede ser resuelto por un algoritmo determinístico en tiempo $O(2^{p(n)})$.*

De esta manera, todo lo que podemos resolver con un algoritmo nodeterminístico también puede ser resuelto con uno determinístico, pero con bastante más trabajo.

El lema que aparece a continuación es solo a fin de completar algunos conceptos mencionados con anterioridad.

Lema 2.1 *Si $L_1 \prec_p L_2$, entonces $L_2 \in P \Rightarrow L_1 \in P$ (de la misma manera, $L_1 \notin P \Rightarrow L_2 \notin P$).*

2.2.7 Teorema de Cook

A continuación enunciamos el teorema de Cook, central a la teoría de la complejidad. La importancia de este teorema radica en que establece que la clase de problemas NP-Completos no son una mera invención matemática, sino que realmente existen problemas en esta clase. Mas aún, con el tiempo se descubrió que la gran mayoría de los problemas interesantes están en la clase de los NP-Completos.

El Problema de la Satisfactibilidad

El problema de la satisfactibilidad, SAT en adelante, fue el primer problema en probarse NP-Completo, ya que Cook lo utilizó para dar una reducción de cualquier problema $\Pi \in$ NP a él. Supongamos que tenemos un conjunto $U = \{u_1, \ldots, u_m\}$ de variable booleanas. Definimos una asignación de verdad como

la función $t : U \mapsto \{T, F\}$ de tal forma que si $f(u) = T$ decimos que u es verdadera y si $f(u) = F$ decimos que es falsa. Necesitamos también definir las cláusulas sobre U como un conjunto de literales de U, por ejemplo $\{u_1, \neg u_5, u_7\}$ que representa la disjuncion de literales. Si y solo si, al menos uno de estos literales es verdadero decimos que la cláusula es satisfecha por la asignación de verdad. SAT, entonces, queda formulado de esta manera:

Definición 2.2 *SAT:*

INSTANCIA: Un conjunto U de variables y una colección C de cláusulas sobre U.
PREGUNTA: Hay una asignación de verdad que satisfaga C?

Teorema 2.2 *SAT es NP-Completo*

La demostración de este teorema esta fuera del objetivo de este trabajo y es por eso que no se la incluye, sin embargo el lector interesado puede ver su hermosa demostración en por ejemplo [9] o [15]. A partir de este resultado de Cook, y de las reducciones polinomiales se han encontrado innumerable cantidad de problemas NP-Completos. Para ver una lista extensa de los mismos con sus clases de aproximación el lector puede referirse a [10].

2.3 Algoritmos de Aproximación y Heurísticas

Una gran cantidad de problemas que surgen en la industria, economía, logística, etc., son NP-Completos. Como ya se dijo antes, es imposible resolver este tipo de problemas eficientemente [2] a menos que $P = NP$, lo que difícilmente sea cierto. Sin embargo, aún así debemos encontrar soluciones a estos problemas. La teoría de la complejidad nos dice que no debemos esperar resolver este tipo de problemas con un algoritmo polinomial en el tamaño de la entrada en forma óptima o exacta. Entonces, mientras sea necesario resolverlos, deberemos relajar el problema o las condiciones pedidas a las soluciones. Quedamos pues, ante diversas alternativas:

- Heurísticas de tiempo superpolinomial:

  Se relaja la condición que el problema deba ser resuelto en forma polinomial. En algunos casos, algoritmos que son superpolinomiales se comportan bien en instancias de tamaño reducido. Sin embargo, cuando nos enfrentamos a problemas "de la vida real", es muy raro encontrar buen comportamiento y casi siempre se verifican tiempos exponenciales.

- Análisis probabilístico de heurísticas:

  Cuando se trabaja con este tipo de análisis, lo que se hace es considerar que solo se resolverán un subconjunto de las instancias del problema, para las cuales existen garantías de performance y límites. Y se acepta que el algoritmo utilizado no se comportará igual sobre todos las entradas posibles. El problema con este enfoque es que el análisis del comportamiento de algoritmos bajo ciertas distribuciones estadísticas es, muchas veces, intratable *per se*.

[2] Consideramos eficientes a los algoritmos polinómicos en el tamaño de su entrada

- Algoritmos de aproximación y heurísticas:

  La argumentación anterior nos lleva a la tercera posibilidad, que es la de relajar la condición de encontrar siempre la solucion óptima. En muchos casos se puede proveer un algoritmo que solucione el problema aproximadamente, es decir, que devuelva una resultado casi óptimo. En la mayoría de los casos es suficiente, e inclusive a veces indistinguible, de la solución óptima real. Entonces parecería que tiene sentido desarrollar tal tipo de sistemas que sean eficientes en resolver NPO [3] bajo la condición que las soluciones puedan ser un poco diferentes de las óptimas.

2.3.1 Órdenes de Magnitud

En esta sección definiremos la forma de cuantificar la tasa de crecimiento de diferentes funciones e introduciremos los símbolos que denotan los diferentes comportamientos. En base a estos símbolos es que se mide la complejidad espacial y temporal de los programas. En lo que sigue supongamos que $f(x)$ y $g(x)$ son funciones de x.

Definición 2.3 *Decimos que* $f(x) = o(g(x))(x \to \infty)$ *si* $lim_{x\to\infty}\frac{f(x)}{g(x)}$ *existe y es igual a cero.*

básicamente esto nos está diciendo que $g(x)$ crece más rápidamente que $f(x)$.

Definición 2.4 *Decimos que* $f(x) = O(g(x))(x \to \infty)$ *si* $\exists C, x_0$ *tales que* $\mid f(x) \mid < C * g(x)(\forall x > x_0)$.

en este caso $f(x)$ no crece más rápido que $g(x)$.

Definición 2.5 *Decimos que* $f(x) = \Theta(g(x))$ *si existen constantes* $c_1 \neq 0, c_2 \neq 0, x_0$ *tal que* $\forall x > x_0$ *se verifica* $c_1 * g(x) < f(x) < c_2 * g(x)$.

Con esta noción, más fuerte que las dos anteriores, podemos asegurar que $f(x)$ tiene la misma tasa de crecimiento, donde solo las constantes no están determinadas.

Definición 2.6 *Decimos que* $f(x) \sim g(x)$ *si* $lim_{x\to\infty}\frac{f(x)}{g(x)} = 1$.

aquí se afirma que $f(x)$ no solo tiene la misma tasa de crecimiento (como en la definición anterior) sino además el límite del cociente de ambas funciones se acerca a uno a medida que el argumento crece.

Definición 2.7 *Decimos que* $f(x) = \Omega(g(x))$ *si existe un* $\epsilon > 0$ *y una secuencia* $x_1, x_2, x_3, \ldots \to \infty$ *tal que* $\forall j :\mid f(x_j) \mid > \epsilon * g(x_j)$.

informalmente se puede interpretar esta definición como la negación de $o()$.

[3] ver definición más abajo

2.3.2 Problemas de Optimización NP

A pesar de haber hecho una breve presentación de la teoría subyacente a la NP-Completitud, aún no hemos definido formalmente que es un problema de optimización NP, NPO en adelante. A continuación definiremos algunos conceptos que nos serán de utilidad [69].

Definición 2.8 *Un problema de optimización, NPO,* Π *está caracterizado por tres componentes*

- *Instancias:*

  D | *un conjunto de instancias.*

- *Soluciones:*

  $S(I)$ | *el conjunto de soluciones factibles para cada instancia* $I \in D$.

- *Costos:*

  f | *una función que asigna cierto costo a una dada solución, por ejemplo alguna función de la forma* $f \mid S(I) \rightarrow \Re$

Definición 2.9 *Un problema de maximización* Π *es:*
dado $I \in D$, *encontrar una solución* $\Sigma^I_{opt} \in S(I)$ *tal que* $\forall \Sigma \in S(I)$, $f(\Sigma^I_{opt}) \geq f(\Sigma)$. *También nos referiremos al valor de la solucion óptima como* $OPT(I)$, *de forma que* $OPT(I) \equiv f(\Sigma^I_{opt})$.

Definición 2.10 *Un algoritmo de aproximación A, para un PO* Π, *es un algoritmo de tiempo polinomial tal que: dada como entrada una instancia I para* Π, *retornará algún* $\Sigma \in S(I)$. *Denotaremos* $A(I)$ *al valor de* $f(\Sigma)$ *de la solución obtenida por A.*

2.3.3 La clase PLS

A continuación definiremos una nueva clase de complejidad que será mencionada más adelante en relación a una heurística para el *ETSP* . En [17] se define la clase PLS, por *Polynomial-time Local Search*, aparentemente situada entre P y NP. Un problem Π en PLS verifica que:

- Para cada $I \in D$ tenemos el conjunto $S(I)$ tal que dados I y s, es fácil verificar si $s \in S(I)$.

- Dado I podemos producir en tiempo polinomial s_0, con $s_0 \in S(I)$.

- Teniendo I y $s \in S(I)$ podemos calcular en tiempo polinomial el costo de s mediante $f(s)$.

- Además, dados I y $s \in S(I)$, podemos verificar en tiempo polinomial si es un óptimo local, y si no es este el caso, producir una solución con mejor costo.

Π es entonces el siguiente problema computacional:
dado I, obtener una solución localmente óptima $s \in S(I)$.

Para probar que una heurística (algoritmo) pertenece a la clase de los PLS-Completos debemos proveer una reducción PLS. Una reducción PLS de Π_1 a Π_2 en PLS se define en término de dos funciones computables polinomialmente $g()$ y $h()$. Dada una instancia x de Π_1, $g()$ computa una instancia $g(x) \in \Pi_2$ tal que, para cada óptimo local s de $g(x)$, $h(x)$ es el óptimo local de x.

Estos conceptos son muy importantes a la hora de demostrar ciertas equivalencias computacionales entre diversos problemas al mirarlos desde la óptica de las heurísticas que los resuelven.

2.4 *TSP* : Complejidad y Algoritmos de Aproximación

La demostración de que $TSP \in NPC$ es bastante extensa y muy técnica, la misma puede ser encontrada en, por ejemplo,[15]. Sin embargo en esta sección mostraremos que el problema del *Circuito Hamiltoniano* puede ser transformado polinomialmente al *TSP* , y como $HC \in NPC$ [15], se sigue que $TSP \in NPC$.

Si bien el *TSP* es quizás el más estudiado de los PO, ha escapado, hasta hace muy poco tiempo, a todo intento por obtener buenos esquemas de aproximación. Ilustraremos también los primeros resultados de aproximación y los más recientes.

2.4.1 *TSP* es NP-Completo

Teorema 2.3 HC $\prec_p$ TSP.

Demostración:

Para demostrar el teorema lo que debemos hacer es proveer una función total computable f tal que $f : HC \mapsto TSP$ y verifique $\forall x \in \Sigma^*_{HC}, x \in L_{HC} \Leftrightarrow f(x) \in L_{TSP}$.

Supongamos que tenemos un grafo $G = (V, E)$ con $\mid V \mid = m$ y $G \in L_{HC}$. Construimos una instancia del *TSP* con el mismo conjunto de vertices V , llamado C, y $\forall v_i, v_j \in C$ definimos $dist(v_i, v_j) = 1$ si $(v_i, v_j) \in E$, o en caso contrario igual a 2. así, el límite en la longitud del tour será $B = m$. Demostrar que esta función es total computable y polinomial es trivial.

La parte esencial de la prueba es: $\forall x \in \Sigma^*_{HC}, x \in L_{HC} \Leftrightarrow f(x) \in L_{TSP}$.

Supongamos que $(v_1, \ldots, v_m)$ es un circuito hamiltoniano HC, de G, entonces también es un tour candidato en $f(G)$. Es más, este tour tiene exactamente longitud m porque está compuesto de aristas de E, cada una de las cuales tiene longitud 1. De la misma manera, supongamos que $(v_1, \ldots, v_m)$ es un tour de $f(G)$ con longitud no menor a B. Ya que hay exactamente m términos aditivos, y a que las distancias interciudades son de longitud 1 o 2 únicamente, se sigue que cada una de ellas contribuye con 1. De la definición de $f()$, se tiene que $(v_i, v_{i+1}) \in E, \forall 1 \leq i \leq m$ y lo mismo vale para (v_m, v_1), constituyendo un HC para G. Lo que prueba el teorema.

2.4.2 Definición del NPO *min ETSP*

El problema del viajante de comercio puede entonces definirse como un problema de optimización no determinísticamente polinómico como sigue:

Definición 2.11 *ETSP es una terna* $(D, S(I), f)$ *donde:*

- $D|$ *es un conjunto instancias. Cada instancia, a su vez, es un conjunto de ciudades* $(c_1, c_2, \ldots, c_N)$ *donde para cada una de ellas se conocen sus coordenadas en el plano Euclideo. Dichas coordenadas inducen una matriz de distancias* $Dist : N*N$; $Dist_{i,j} \in Z^+$ (*siendo* $Dist_{i,j}$ *la distancia entre la ciudad* c_i *y la ciudad* c_j).
- $S(I)|$ *es el conjunto de soluciones factibles para cada* $I \in D$, *en este caso, el conjunto de "tours" o permutaciones* $(c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(N)})$ *de las* N *ciudades en* I, *donde* $Tour(N)$ *se identifica con* $Tour(0)$ *de tal forma de cerrar un ciclo.*
- $f|$, *la función de costo tal que* $f|S(I) \mapsto \Re$ *y*

$$f(Tour) = \sum_{k=0}^{k=N-1} Dist_{Tour(k),Tour(k+1)} \tag{2.1}$$

2.4.3 Algoritmo de Rosenkrantz-Stearns-Lewis

Rosencrantz et. al. en [71] presentan un algoritmo de aproximación para el *TSP* descripto a continuación.

Teorema 2.4 *Existe un algoritmo de tiempo polinomial que resuelve el TSP brindando una solución con a los sumo dos veces la longitud del tour óptimo*

Demostración:
Supongamos que tenemos n ciudades, el algoritmo continúa como sigue...

1. Encontrar el MST [4] T de las n ciudades.
2. Duplicar cada arista de T, obteniendo T' en el cual, para cada par de vértices, existen 0 o 2 aristas.
3. Encontrar un tour Euleriano W de aristas en T'. Esto es factible de hacer porque cada vértice tiene un número par de aristas incidentes.
4. Elegir una ciudad y seguir W. Cuando llegamos a un vértice v ya visitado, ir de v directamente al próximo vértice del camino W que aún no haya sido visitado.

El tour obtenido en 4, llamemosle Z', contiene todas las ciudades y pasa por cada una sólo una vez. Sea Z el tour óptimo, entonces $longitud(Z') \leq 2 * longitud(Z)$. Sea e alguna arista de Z. Entonces $Z - e$ es un camino que visita todas las ciudades. Dado que un camino es un árbol, $Z - e$ es un árbol de recubrimiento, siendo $Z - e$ por lo menos tan largo como T, de esa forma

[4] MST son las iniciales de Minimum Spanning Tree, remitirse al capítulo de Grafos de Vecindades

Z es al menos tan costoso como T. Un paso de Z' que toma una arista de W tiene una longitud igual a la de dicha arista de W. Un paso de Z' que acorta camino salteando varias aristas de W tiene longitud a lo sumo igual a la suma de longitudes de las aristas de W que fueron evitadas. Si se suman estas desigualdades sobre todos los pasos de Z', encontramos que la longitud de Z' es a lo sumo igual a la longitud de W, que es el doble de T. Esto es

$longitud(Z) > longitud(Z - \mathrm{e}) \geq longitud(T) = \frac{1}{2} * longitud(W)$
$\geq \frac{1}{2} * longitud(Z')$,
lo que prueba el teorema.

2.4.4 Algoritmo de Christofides

En [12] Christofides presenta una mejora al algoritmo de Rosenkrantz et. al. basada en el hecho de que el mínimo matching euclideo puede ser computado en $O(N^3)$[54]. Este problema se define así:

Definición 2.12 *Mínimo Matching Euclideo:*
Dados $2 * N$ puntos en el plano, unirlos de a pares por segmentos, tales que la suma total de sus longitudes sea mínima.

Entonces se plantea el siguiente teorema cuya demostración es muy similar a la del teorema 2.4 y se omite,

Teorema 2.5 *Una aproximación al TSP cuya longitud es a lo sumo $\frac{3}{2}$ del óptimo puede ser obtenida en tiempo $O(N^3)$ si las distancias entre puntos verifican la desigualdad triangular.*

2.4.5 Algoritmos de Arora y Mitchell: PTAS para el *ETSP*

Como se dijo más arriba, varias décadas de esfuerzo debieron ser recorridas para llegar recién en 1996 a obtener un "Esquema de Aproximación de Tiempo Polinomial", PTAS en lo que sigue, para el *ETSP* . En [66] y [45] se describen sus respectivos esquemas, donde ambos comparten principios similares, aunque fueron desarrollados independientemete. Lo que es importante notar es que dichos PTAS hacen uso de programación dinámica para obtener esos resultados.

En [66] se enuncia lo siguiente,

Teorema 2.6 *Para todo entero positivo determinado m, existe un algoritmo $O(n^{20*m+5})$ que computa un tour aproximado para el TSP , cuya longitud euclideana esta dentro de un factor $(1 + \frac{2*\sqrt{2}}{m})$ del óptimo.*

Es menester remarcar el alto grado del polinomio, lo que hace muy poco viable el uso de este algoritmo en grandes instancias. Todavía habrá que esperar algún tiempo para obtener mejoras teóricas en estos factores e implementaciones eficientes de los algoritmos descriptos en [45] y [66]. Mas aún, Trevisan en [73] muestra que el *min TSP* es *Max SNP-hard*, es decir, *NP-hard* de aproximar dentro de una constante $r > 1$, inclusive en el caso de que todas las ciudades esten embebidas en un espacio geométrico $\Re^n$ (con n el número de ciudades) y las distancias computadas en base a L_1. Dichos resultados son también válidos para cualquier norma L_p o un espacio de $\Re^n$. Como consecuencia de lo anterior, Trevisan muestra que a menos que *NP* tenga algoritmos subexponenciales, el esquema de aproximación de Arora para el *TSP* geométrico en $\Re^d$ es doblemente exponencial en d.

2.5 Conclusiones

En este capítulo hemos repasado algunos conceptos básicos de *complejidad computacional*, *optimización combinatoria* y *teoría de algoritmos*. Se mostró la NP-Completitud del *TSP* , y algunos algoritmos de aproximación y PTAS para el mismo. Creemos que es fundamental para poder mejorar y desarrollar nuevas heurísticas para este problema conocer cual es el estado del arte y contra que tipo de enfoques hay que competir. Además, siempre que las instancias sean razonablemente pequeñas podemos usar los algoritmos de aproximación citados, pero cuando las mismas alcanzan tamaños grandes, se imponen las heurísticas con la consiguiente pérdida de garantía en las cotas de error.

2.6 Notas:

Capítulo 3

Geometría Computacional y Grafos de Proximidad

No se si volveremos en un ciclo segundo como vuelven las cifras de una fracción periódica. Pero sé que una oscura rotación pitagórica noche a noche me deja en un lugar del mundo.

La Noche Cíclica
J.L. Borges

En este trabajo, como se mencionó en la introducción, se desarrollarán algoritmos y heurísticas para resolver el *TSP* basadas en construcciones de la geometría computacional. Dicha disciplina cobró un auge notorio en los últimos 20 años debido al advenimiento de nuevas tecnologías computacionales que hicieron aplicable sus métodos.

Sin embargo, podemos remontarnos a muy larga data para encontrar que los antiguos geómetras también se preocuparon por la corrección, completitud y complejidad, tanto espacial como temporal, de sus métodos.

La geometría, computacional o no, estudia problemas que surgen tanto de "la práctica deficiente" como de "la sana teoría" [75]. Geodesia, arquitectura, ingeniería, minería, análisis, álgebra, y mas cercanos a nuestro campo del saber, los gráficos por computadora, reconocimiento de patrones, robótica, optimización combinatoria, métodos numéricos, bases de datos, etc., son algunos de los múltiples campos en los que florecieron conceptos, problemas y soluciones de la geometría como tal.

Los geómetras egipcios utilizaron su saber en la medición y división de tierras, también en la construcción de magníficas estructuras que hoy todavía podemos apreciar. De igual manera grandes hombres de geometría fueron los griegos.

Euclides introdujo el método axiomático, dando origen a lo que se llamo construcción euclideana. Una construcción euclideana es una sucesión de acciones, que es finita, correcta y no ambigua; construcción que es algoritmo y prueba a la vez. Ya los antiguos geómetras se preocuparon por conocer la completitud de las operaciones con regla y compás que utilizaban para resolver problemas. El

reductio ad absurdum durmió a la geometría por casi 2000 años, pues se podían resolver problemas más facilmente que con el uso de construcciones geométricas.

Recién con la "algebraización" de la geometría gracias a Descartes se pudo comprobar la incompletitud del método euclideano. Con la introducción del sistema cartesiano de coordenadas resurge el constructivismo y se otorga a la geometría un nuevo poder expresivo. Mas tarde surgen ilustres matemáticos como Leibnitz y Newton que, fertilizando el álgebra con la nueva geometría, inventarían, independientemente uno del otro, el cálculo.

Gauss haciendo uso del álgebra responde a problemas geométricos antes irresolubles. Los geómetras post euclideanos se preocuparon en simplificar sus construcciones. En 1902 Lemoine establece la *Geometrography* y codifica las primitivas euclideanas en:

- Apoyar el compás sobre un punto.
- Apoyar el compás sobre una línea.
- Hacer un círculo.
- Trazar una línea por un punto.
- Apoyar la regla en un punto.

y llamó "simplicidad" a la cantidad de estos pasos que se usan en la resolución de un problema. Asi como Lemoine se ocupo de tratar de medir la simplicidad de una construccion, Hilbert en 1899 estudia cuales son los mínimos pasos necesarios para llevar a cabo cierta construcción. De esta manera se acerca mas aún a nuestros conceptos de complejidad conputacional. En 1672 Mohr demuestra que cualquier construcción realizable con regla y compás, puede hacerse solo con este último.

La analogía llega mas lejos... hasta el concepto de complejidad espacial; en 1972 Eves estudia el espacio requerido para concretar cierta construcción.

Es así como la geometría estuvo siempre íntimamente relacionada con los modernos conceptos de completitud, correctitud, y complejidad, tanto temporal como espacial. Hasta sería dado imaginar que las raíces de los actuales, arriba mencionados, conceptos se remontan a la ciencia-arte de egipcios y griegos.

3.1 Problemas de Proximidad

En geometría computacional se plantean una serie de problemas, llamados problemas de proximidad, donde se trata de determinar ciertos objetos geométricos a partir de alguna propiedad específica que determina su "proximidad" a otros objetos geométricos. Estos problemas, que a primera vista podrían parecer distintos, son esencialmente similares, y transformando las soluciones de unos podemos resolver otros. En el gráfico 3.1 obtenido de [35] se muestra la relación que existe entre varios de ellos, y sus definiciones se dan a continuación:

Definición 3.1 *PAR MAS CERCANO(CP): Dados N puntos en el plano, encontrar dos cuya distancia mutua sea la mínima entre todos los pares.*

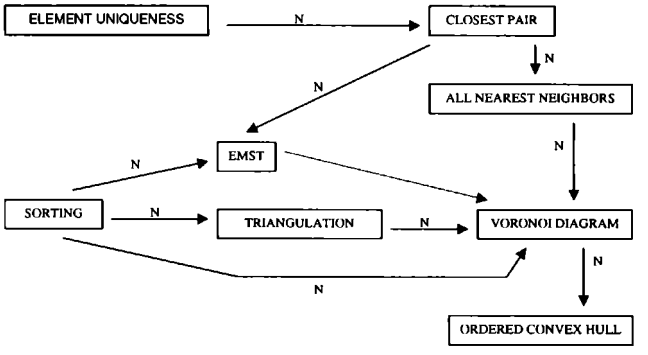

Figura 3.1: Relación entre algunos problemas de proximidad y los prototipos computacionales

Definición 3.2 *TODOS LOS VECINOS MÁS CERCANOS(ANN): Dados N puntos en el plano, encontrar el vecino más cercano de cada uno. Se define la relación vecino más cercano sobre un conjunto de puntos S como,*

$a, b \in S, a \rightarrow b \Leftrightarrow \delta(a,b) = min_{c \in S-\{a\}}\delta(a,c)$, *donde delta*$(a,b)$ *es la distancia de a a b. Se dice que b es el vecino más cercano de a. Otra manera de notar* $a \rightarrow b$ *es* $b = NN(a)$.

De las definiciones presentadas se deduce que si u y v son PARES MAS CERCANOS entonces $u = NN(v)$ y $v = NN(u)$. El problema de los vecinos más cercanos es encontrado inmerso en infinidad de otros problemas. En este trabajo veremos una heurística para el *ETSP* basada en este concepto y utilizaremos los llamados *KD-trees* [5] para, respondiendo el "range querie" apropiado, construir los grafos de vecinos relativos y de Gabriel.

Definición 3.3 *ARBOL EUCLIDEO DE RECUBRIMIENTO MÍNIMO(MST): dados N puntos en el plano construir un árbol que recubra todos los puntos y cuya longitud total sea mínima.*

Es posible encontrar este problema en innumerables situaciones prácticas, en general cuando se trata de problemas de tendido de redes eléctricas, gas, agua, etc. En el capítulo 5 se describen otras situaciones donde en forma natural puede recurrirse al *MST* para su modelización. En la figura 3.2 se muestra el árbol de recubrimiento mínimo de una instancia de TSPLIB.

Definición 3.4 *CÁSCARA CONVEXA(CH): dado un subconjunto arbitrario, L de puntos en* E^d, *la cáscara convexa,* $CH(L)$, *se define como el menor polígono convexo que contiene a L.*

Intuitivamente podemos visualizar el convex hull de un conjunto de puntos imaginando que las posiciones de dichos puntos representan las coordenadas donde se ubicarán unas clavijas (en $\Re^2$). Si, alrededor de las clavijas se suelta una banda elástica, esta se adaptará a los límites convexos de la superficie que determinan las clavijas. La forma que adoptara la banda será aquella de la cáscara convexa.

Figura 3.2: att532 - árbol de recubrimiento mínimo

Definición 3.5 *TRIANGULACIONES: dados N puntos en el plano, unir los mismos por líneas rectas que no se intersecten, de tal forma que cada región interna a la cáscara convexa sea un triángulo.*

Las triangulaciones tienen utilidad en problemas tan variados como los gráficos por computadora, método de los elementos finitos, interpolación de funciones, etc. Existen diferentes criterios para decidir la bondad de una triangulación. Sin embargo, la mayoría de las veces dichos criterios no garantizan realmente que la triangulacion resultante sea la óptima para la aplicación en cuestión. Se opta por uno u otro criterio por la facilidad con la cual se puedan derivar cotas teóricas a los errores numéricos de los algoritmos que las utilizan.

Definición 3.6 *BÚSQUEDA DEL VECINO MÁS CERCANO: dados N puntos en el plano, cuan rápido podemos determinar el vecino más cercano a un punto nuevo dado si se permite preprocesamiento?*

Definición 3.7 *BÚSQUEDA DE LOS k VECINOS MÁS CERCANOS: dados N puntos en el plano, cuan rápido pueden encontrarse los k puntos más cercanos a un nuevo punto dado?*

Los dos problemas anteriores se vinculan estrechamente con problemas de clasificación, reconocimiento de patrones, recuperación de la información, etc. [63]

Todos los problemas definidos más arriba han sido objeto de numerosos estudios y aún hoy quedan muchas conjeturas sin dilucidar. A lo largo de este trabajo describiremos los algoritmos para resolverlos, su complejidad computacional y las propiedades importantes de ellos.

3.2 Grafos de Proximidad

La mayoría de los problemas citados anteriormente pueden ser tratados desde una formulación más general, aquella de los grafos de proximidad, también llamados grafos de vecindario. Se definen como sigue:

Sea V un conjunto de puntos en $\Re^d$, con $d \in Z^+$. Cada par de puntos $(p,q) \in V * V$ se asocia a un vecindario $U_{p,q} \subset \Re^d$, *locus* en adelante. Sea P una propiedad definida sobre $U = \{U_{p,q} | (p,q) \in V * V\}$. Un grafo de vecindario $G_{U,P}(V,E)$ definido por la propiedad P, es un grafo con vértices V y aristas E tales que $(p,q) \in E \Leftrightarrow U_{p,q}$ verifica la propiedad P. Si $(p,q) \in E$ entonces decimos que p y q son vecinos el uno del otro. Hay que destacar que para algunos grafos de proximidad es mejor hablar de vecindarios de puntos y no de pares de puntos. El vecindario de una arista se define usualmente usando el concepto de distancia. Existen diferentes definiciones de distancia, $\delta(x,y)$, entre dos puntos $x, y \in \Re^d$. Por ejemplo:

- $\delta_p(x,y) = \sum_{i=1}^{i=d} |x_i - y_i|^{\frac{1}{p}}$ con $1 \leq p < \infty$ y $p \in Z$.
- $\delta_\infty = max_{1 \leq i \leq d} |x_i - y_i|$

Sin embargo otras distancias más generales pueden utilizarse. En general $\delta(x,y)$ se conocerá como la longitud de la arista pq.

Una esfera centrada en x, de radio r que no incluye su contorno, esto es abierta, se define como $B(x,r) = \{y | \delta(x,y) < r\}$. Si se considera el perímetro como parte de la misma se llama esfera cerrada y queda definida como $\overline{B}(x,r) = \{y | \delta(x,y) \leq r\}$.

3.2.1 Grafo de Vecinos Relativos

Sea

$$\Delta_{p,q} = B(p, \delta(p,q)) \cap B(q, \delta(p,q)) \tag{3.1}$$

$\Delta_{p,q}$ es llamada luna. El grafo de vecinos relativos sobre V, $RNG(V)$, se define como sigue:

$$(p,q) \in E \Leftrightarrow \Delta_{p,q} \cap V = \emptyset \tag{3.2}$$

Es interesante notar que el $RNG(V)$ para puntos en $\Re^d$ también se puede definir como el grafo con vértices V y aristas (p, q) tales que

$$\delta(p,q) \leq min_{v \in V-\{p,q\}} max(\delta(p,v), \delta(q,v)), \tag{3.3}$$

siendo $\delta()$ la distancia apropiada.

$\Delta_{p,q}$ definida como más arriba, recibe el nombre de *Vesica Piscis* y puede encontrarse en el diseño de algunas catedrales góticas.

Obsérvese también que una arista $(p, q) \in RNG(V)$ si no existe ningún triángulo $\Delta_{p,q,v}$ con $v \in V - \{p, q\}$ tal que (p, q) es su hipotenusa. En 3.3 se muestra el grafo de vecinos relativos de la instancia att532.tsp .

Figura 3.3: att532 - grafo de vecinos relativos

3.2.2 Grafos de Gabriel

El vecindario llamado esfera-diámetro se define como

$$\Gamma_{p,q} = B(\frac{p+q}{2}, \frac{\delta(p,q)}{2}). \tag{3.4}$$

siendo $\frac{p+q}{2}$ el punto medio del segmento $\overline{pq}$. El grafo de Gabriel [1] de un conjunto de vértices V $GG(V)$, esta dado por V y el conjunto de aristas que verifican $(p,q) \in E \Leftrightarrow \Gamma_{p,q} \cap V = \emptyset$. En el espacio euclídeo de dos dimensiones, (p,q) es una arista de $GG(V)$ si no existe ningún $\triangle_{p,q,v}, v \in V - \{p,q\}$ con $\angle_{p,q,v} > \frac{\pi}{2}$. Una definición alternativa es que una arista (p,q) pertenece a $GG(V)$ si

$$\delta_2(p,q) \leq min_{s \in V}\{\sqrt{\delta_2^2(p,s) + \delta_2^2(s,q)}\} \tag{3.5}$$

Para la misma instancia que en el caso anterior se presenta en 3.4 el grafo de Gabriel.

3.2.3 Diagramas de Voronoi

El diagrama de Voronoi resuelve el siguiente problema (problema del *locus*):

"Dado un conjunto S de N puntos en el plano, para cada punto p en S; cuál es la ubicación espacial, locus, de los puntos de coordenadas (x,y) que se encuentran más cercanos a p que a cualquier otro punto de S ?".

Dados p y q, el conjunto de puntos más cercanos a p que a q es el semiplano que contiene a p definido por la mediatriz a $\overline{pq}$. Denotemos dicho semiplano por $H(p,q)$. La celda de Voronoi asociada al punto p ($V_R(p)$), es una región poligonal que posee no más de $N-1$ lados (ya que es la intersección de a lo sumo $N-1$ semiplanos que quedan definidos por p y los otros $N-1$ puntos en S). Entonces,

$$V_R(p) = \bigcap_{p,q \in S, p \neq q} H(p,q) \tag{3.6}$$

es la celda de Voronoi de p. Si consideramos las celdas para cada punto de S obtenemos el diagrama de Voronoi.

En cierto sentido se podría argumentar que este grafo captura "toda" la información de proximidad de un conjunto de puntos. A continuación se da una lista de propiedades interesantes del DV [35].

- Para cada vértice v del DV de S, el círculo centrado en v no contiene ningún otro punto de S que no sea aquellos tres que lo definen.
- Cada vecino más cercano de $p \in S$ define una arista de la celda de Voronoi de p.
- La celda de Voronoi de p es ilimitada si p radica en el $CH(S)$.
- El dual de $DV(S)$ es $TD(S)$.

[1] Estos grafos, si bien de Gabriel, no son de Gabriel Baum[38].

Figura 3.4: att532 - grafo de gabriel

Con esta construcción geométrica se pueden responder en forma eficiente los problemas de:

- Todos los vecinos más cercanos, $O(N * Log(N))$.
- El vecino más cercano, $O(N * Log(N))$.
- Una triangulación con la propiedad que el circuncírculo de cada triángulo es vacío (TD), $\Theta(N * Log(N))$
- El convex hull del conjunto de puntos puede ser encontrado en $O(N)$.

donde $N = \mid S \mid$.

3.2.4 Triangulación de Delaunay

La triangulación de Delaunay, como es el caso de los grafos anteriores, puede ser caracterizada de diversas manera, pero la más frecuentemente encontrada en la bibliografía es la que la define como el grafo dual del diagrama de Voronoi de un

conjunto de vértices V. El diagrama de Voronoi de V es una descomposición del espacio en N celdas asociadas a cada uno de los vértices $v \in V$. Un punto x está en la celda asociada a $v \in V$ si $\forall w \in V - \{v\}, \delta(x, v) \leq \delta(x, w)$. Entonces dos vértices de $TD(V)$ están unidos por una arista si el perímetro de sus celdas de Voronoi se intersecta. Alternativamente podemos decir que dado V un conjunto de vértices, construimos $TD(V)$ con todas las aristas (p, q) tales que a través de ellas pasa un círculo que no contiene ningún otro $x \in V$. En 3.5 se muestra la triangulación de Delaunay de un conjunto de puntos. Nótese la inclusión en la misma de la cáscara convexa.

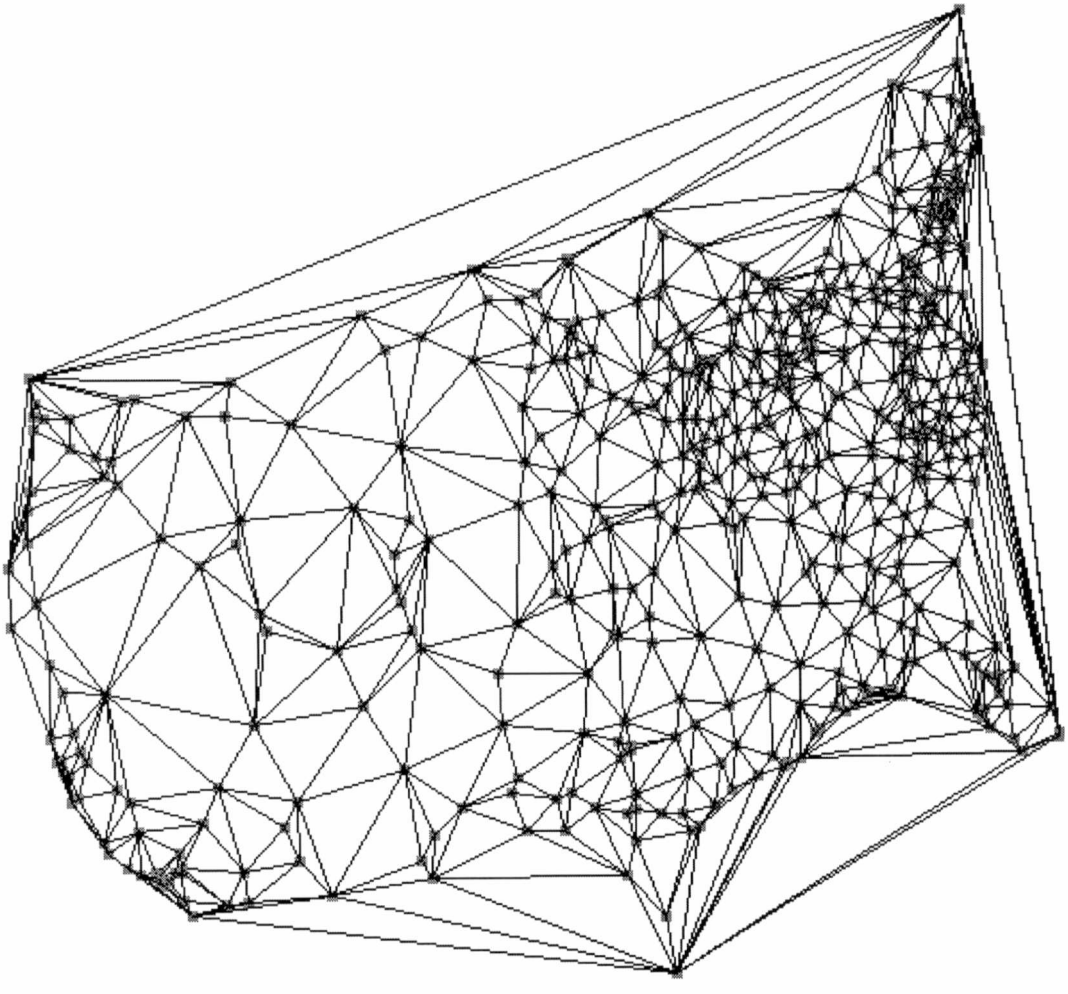

Figura 3.5: att532 - triangulación de Delaunay

3.3 Conclusiones

En este capítulo se presentaron los conceptos básicos de geometría computacional que serán utilizados a lo largo de todo el trabajo. Se definieron los problemas de proximidad más comunes y los grafos de vecinos relativos asociados a algunos de estos problemas. Presentamos una lista con propiedades interesantes de dichos objetos geométricos. A medida que sigamos haciendo uso de estos conceptos en capítulos próximos se introducirán nuevas propiedades y se mostrarán los algoritmos necesarios para calcular los diferentes grafos.

3.4 Notas:

Capítulo 4

Triangulación de Delaunay y Optimización Heurística

Un cronopio pequeñito buscaba la llave de la puerta de calle en la mesa de luz, la mesa de luz en el dormitorio, el dormitorio en la casa, la casa en la calle. Aquí se detenía el cronopio, pues para salir a la calle precisaba la llave de la puerta.

Historia
Historias de Cronopios y de Famas
J. Cortazar

En este capítulo se presentará la motivación de esta tesis y se discutirán algunos conceptos sobre búsqueda heurística.

En la introducción se comentó el trabajo de Reinelt [37] donde se realiza un estudio detallado de la utilización en tandem de heurísticas clásicas como $2-opt$,*nearest neighbor* y *Lin-Kernighan*, luego de obtener un tour factible a partir de un grafo "vecindario", GV_k. Dicho grafo se definía así:

Sea S un conjunto de ciudades a visitar, y $TD(S)$ la triangulación de Delaunay de dicho conjunto, entonces, GV_k tiene el mismo conjunto de vértices (ciudades) S y el conjunto de aristas se define por
$E = \{(u,v) | u,v \in S, \exists C_{TD(S)}(u,v,k)\}$, es decir u estará conectado a v en GV_k si existe en TD(S) un camino de longitud a lo sumo k desde u a v.

A diferencia del estudio realizado por Reinelt, investigaremos sobre el uso de algunos **subgrafos** de $TD(S)$ con propiedades geométricas conocidas. Si nuestras soluciones resultan comparables a los de la utilización de supergrafos de $TD(S)$, debiésemos entonces optar por los primeros por ser éstos:

- grafos con menor número de aristas: $|MST(S)| \leq |RNG(S)| \leq |GG(S)| \leq |TD(S)| \ll |GV_k(S)|$ y,
- construibles en tiempo $O(N * LogN)$ con $N = |S|$.

en caso contrario el enfoque de Reinelt resultaría beneficiado.

4.1 Búsqueda Heurística

En optimización combinatoria e inteligencia artificial la gran mayoría de los problemas pueden ser referidos a una búsqueda. Dicha búsqueda puede ser de dos tipos:[80]

- Encontrar un objeto que verifique ciertas propiedades. Donde el éxito o fracaso de la búsqueda dependerá de si dicho objeto fue hallado o no. En este caso no existe el concepto de proximidad entre una solución y otra.
- Encontrar un objeto tan bueno como sea posible con los recursos disponibles, ya sean estos, tiempo de cómputo, memoria u otro. A diferencia del caso anterior, aquí se cuenta con la noción de distancia o proximidad entre soluciones.

Rich en [84] sugiere que

> "Every search process can be viewed as a traversal of a directed graph in which each node represents a problem state and each arc represents a relationship between the states represented by the nodes it connects."

y efectivamente es este el enfoque que se dio a muchos de los problemas de optimización. Algunas distinciones merecen la pena ser tenidas en cuenta ya que se hará uso de ellas en este trabajo. El grafo donde se realizará la búsqueda puede ser construido y mantenido explícitamente mediante estructuras de datos adecuadas. Sobre dicho grafo, algún algoritmo o heurística describirá una trayectoria, que en esencia es el proceso de búsqueda, y arrojará cierto resultado. Obviamente, esto es factible solo cuando el espacio de configuraciones es chico, mientras que en la gran mayoría de los casos este grafo crece exponencialmente con el tamaño de la instancia en cuestión. La alternativa es que los algoritmos de búsqueda posean un conjunto de reglas a partir de las cuales generar la región del grafo de estados que será explorada. Esto evita mantener estructuras de datos complejas y extensas, para las cuales muchas veces grandes regiones de las mismas son ignoradas.

Remarquemos entonces que, en general, una heurística consta de dos partes, a saber, el grafo que mantiene el espacio de configuraciones y el generador de movimientos. En el grafo de estados, cada nodo representa una solución al problema y las aristas del mismo representan transiciones que llevan de un estado a otro. El generador de movimiento es una estrategia de navegación del grafo de estados, es decir, dado un vértice desde donde comienza la búsqueda, el control debe generar una camino en dicho grafo utilizando las aristas y vértices del mismo. Sirvan como ejemplos de mecanismos de control "deph first search" y "breadth first search". En I.A. se utilizan los términos espacio de estados y control [84][81], mientras que en optimización combinatoria es más frecuente referirse al "landscape" y "navigation strategy" [1] [80].

Es muy importante a la hora de estudiar heurísticas para problemas complejos entender las limitaciones de la estrategia de navegación que se está usando y cuál es el landscape implícito sobre el que dicha estrategia opera. Algunas

[1] Existen sin embargo algunas diferencias importantes entre landscapes y grafos de estados. El lector interesado remítase a la cita mencionada.

de las preguntas claves a responder cuando se nos presenta un nuevo algoritmo son:

- En qué orden y cómo el grafo es explorado.
- Cómo se generan las diferentes componentes conexas de dicho grafo.
- Dónde comienza la exploración.
- Cómo se modifica la estrategia a medida que avanza la búsqueda.
- Cuándo debe terminar.

En [83] Papadimitriou resume lo expuesto de esta manera:

> "A local search heuristic starts with a solution and repeatedly tries to find a better solution which is a *neighbor* of the first. If a better neighbor is found, a search starts for a better neighbor of that one, and so on. Since problems of this sort have finitely many solutions totally ordered by cost, this process always ends at a *local optimum*. The process may be repeated many times from initial solutions generated in some randomized way. Naturally, the most critical part in the design of such heuristic is deciding when two solutions are neighbors. A neighborhood shoul be easy to search, and at the same time rich enough to assure the quality of the local optima obtained."

4.2 Un Vecindario de Búsqueda Para el TSP

Mucho esfuerzo se focalizó en el desarrollo de heurísticas para el *TSP* , sin embargo, considerablemente menos énfasis se ha puesto en el estudio del landscape donde dichas heurísticas operaban o del cual obtenían información. Para comprender mejor a que se está haciendo referencia en las líneas de arriba introduzcamos algunas definiciones.

Sea un grafo $G = (V, E)$, entonces cualquier otro $G' = (V', E')$ con $V' = V$ y $E \subseteq E'$ es un supergrafo de G. Golumbic et. al. en [56] definen un "Sandwhich Problem" como sigue

> " Given two graphs $G^1 = (V, E^1)$ and $G^2 = (V, E^2)$ such as $E^1 \subseteq E^2$, is there a graph $G = (V, E)$ such as $E^1 \subseteq E \subseteq E^2$ which belongs to a specified graph family? "

Este es un marco general y útil en una gran variedad de problemas de reconocimiento como es el caso que nos ocupa. La pregunta sobre la que se trabajará es:

Siendo $G^1 = (V, \emptyset)$ con V el conjunto de ciudades, y sea $G^2 = (V, V \times V)$, existe un grafo $G = (V, E)$ con $\emptyset \subseteq E \subseteq V \times V$ que capture la información de vecindad más esencial del conjunto de ciudades?. Si se hallase ese grafo, las heurísticas que obren sobre el mismo, o sean guiadas por la información de G, deberían comportarse mejor que sus pares sobre $V \times V$. Los experimentos que desarrollaremos apuntarán a probar o refutar la siguiente hipótesis:

> Los grafos en la jerarquía $MST \subseteq RNG \subseteq GG \subseteq DT$ son buenos candidatos para G.

Nombre	$\notin$ TD	$\notin$ GG	$\notin$ RNG
ulysses16	1	3	6
ulysses22	1	4	11
att48	0	4	12
eil51	0	3	10
berlin52	0	3	9
st70	2	7	15
eil76	0	2	13
pr76	2	8	11
gr96	1	7	26
kroa100	1	6	20
kroc100	0	6	20
krod100	1	9	22
rd100	1	8	24
eil101	2	3	20
lin105	2	8	20
ch130	2	11	27
ch150	0	9	23
gr202	4	10	68
tsp225	0	0	22
a280	0	0	26
pcb442	3	8	49

Tabla 4.1: Cantidad de links óptimos que no estan en la triangulación de Delaunay, los grafos de Gabriel o vecinos relativos.

Si se comprueba esta hipótesis se contaría con una nueva familia de heurísticas eficientes para el *ETSP* , ya que todos estos grafos son construibles en tiempo $O(N * LogN)$, siendo N la cantidad de ciudades. En [18] se demuestra que en general la Triangulación de Delaunay no contiene al tour óptimo, porqué entonces insistir con usar la misma como landscape de búsqueda? Para contestar esta pregunta se diseñó un experimento muy simple consistente en superponer a la triagulación de Delaunay el tour óptimo de aquellas instancias para las cuales es conocido. En la tabla 4.1 se muestran dichas instancias y se cuenta para cada una de ellas el número de aristas del tour óptimo que no pertenecen a la triangulación de Delaunay y dos de sus subgrafos más importantes.

Este experimento arroja evidencias claras de que la triangulación de Delaunay sería un buen candidato para ser usada como vecindario de búsqueda para las heurísticas o bien como fuente de información adicional de la cual las mismas podrían obtener datos de control. El lector interesado en un análisis más profundo entre la relación del tour óptimo y la triangulación de Delaunay puede referirse a [39]. En las figuras 4.1, 4.2, 4.3, 4.4, 4.5 se muestran las triangulaciones correspondientes a algunas de dichas instancias. Es importante notar que, cada vez que un link del tour óptimo no esta en la triangulación de Delaunay esto se debe a que todas las ciudades vecinas en Delaunay ya han sido incorporadas al tour y por lo tanto no puede "cortar" camino por una arista de la triangulación. Los links más claros son aquellos del óptimo que no pertenecen a TD. A modo de comparación se incluyó en la tabla 4.1 cuantos

links del tour óptimo no están en el grafo de Gabriel y de vecinos relativos de dichas instancias.

4.3 Conclusiones

En este capítulo se planteó la problemática de la definición de un vecindario de búsqueda adecuado para el *ETSP* . Si bien se demostró en [18] que en general no se puede esperar encontrar al tour óptimo enbebido en TD, comprobamos como la gran mayoría de las aristas del mismo pertenecen a TD. El experimento de este capítulo se realizó solo sobre aquellas instancias cuyas soluciones óptimas están disponibles en TSPLIB.

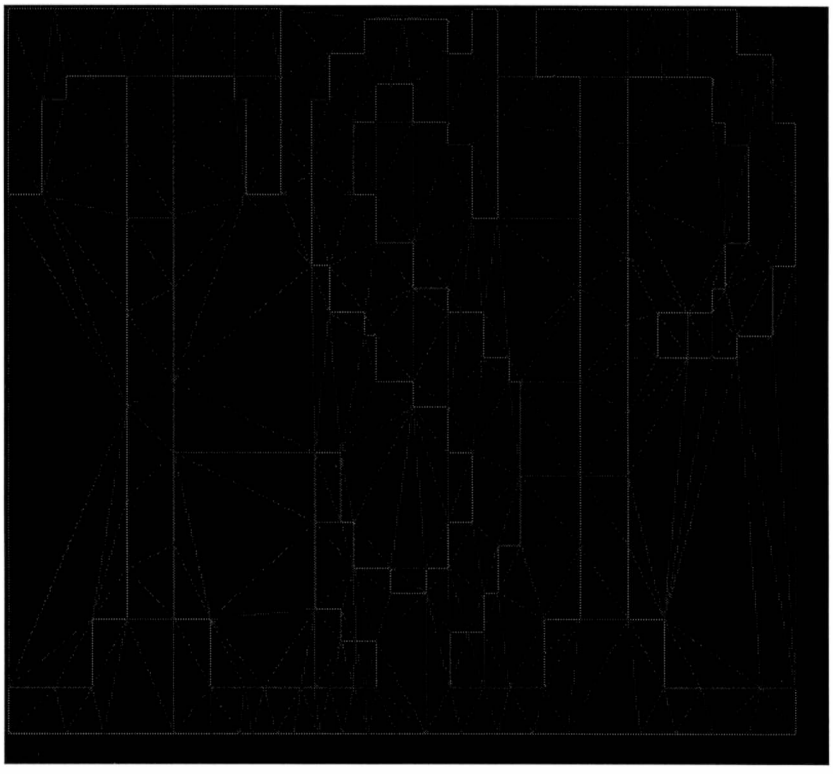

Figura 4.1: tsp225 - triangulación de Delaunay y tour óptimo

Figura 4.2: eil101 - triangulación de Delaunay y tour óptimo

4.4 Notas:

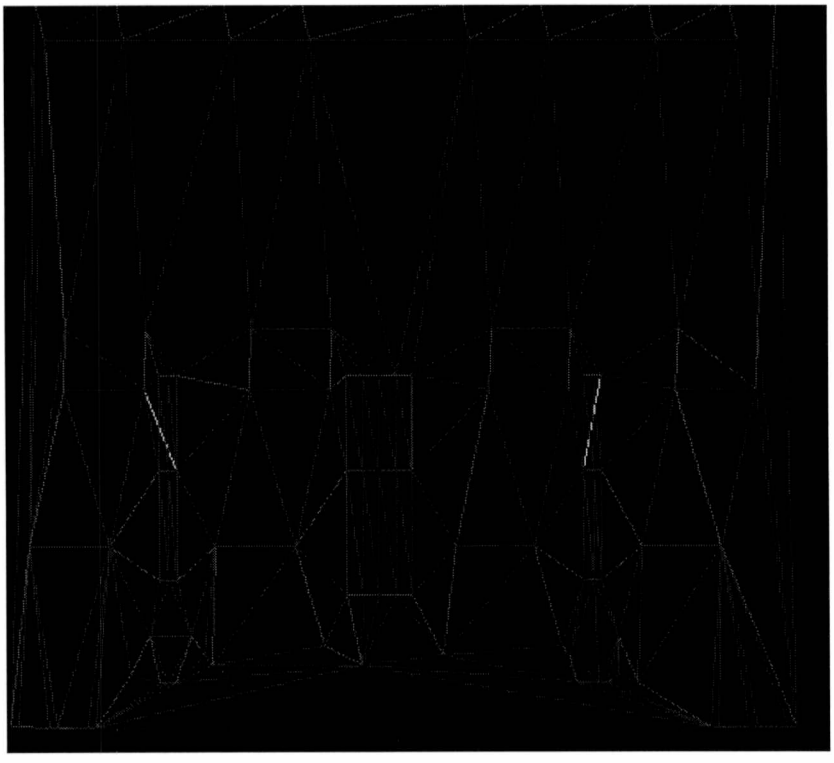

Figura 4.3: lin105 - triangulación de Delaunay y tour óptimo

Figura 4.4: ch130 - triangulación de Delaunay y tour óptimo

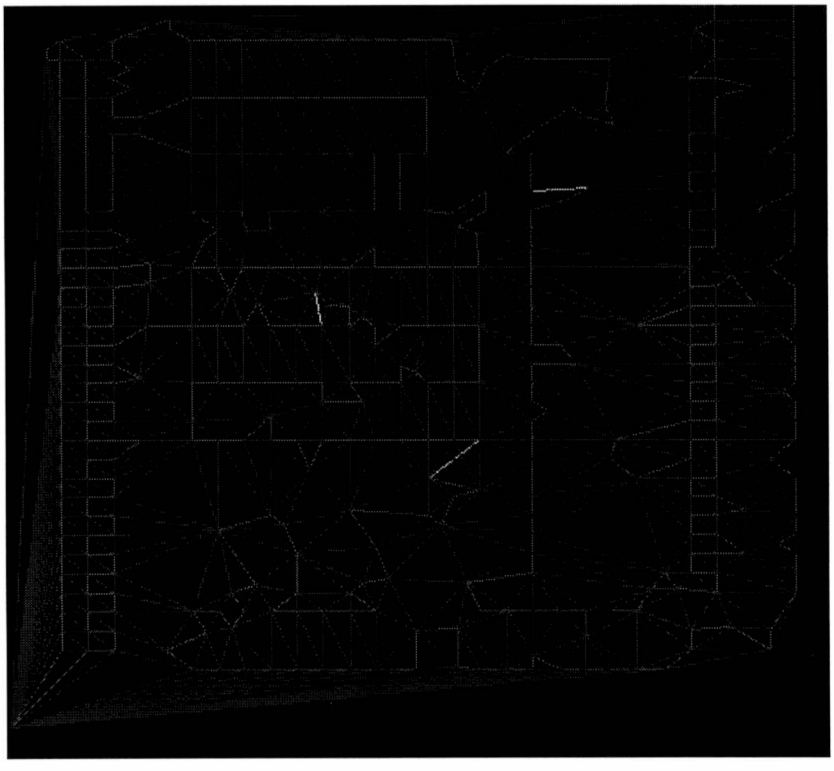

Figura 4.5: pcb442 - triangulación de Delaunay y tour óptimo

Capítulo 5

Estructuras y Códigos Para Problemas Geométricos

Nadie habrá dejado de observar que con frecuencia el suelo se pliega de manera tal que una parte sube en ángulo recto con el plano del suelo, y luego la parte siguiente se coloca paralela a este plano, para dar paso a una nueva perpendicular, conducta que se repite en espiral o en línea quebrada hasta alturas sumamente variables...

Instrucciones Para Subir Una Escalera
Historias de Cronopios y de Famas
J. Cortazar

Este capítulo trata sobre las estructuras de datos y los algoritmos que se utilizarán para calcular los objetos geométricos descriptos en el capítulo anterior. En particular estudiaremos las estructuras de datos básicas de la llamada *The Stanford GraphBase*[1] [25], SGB en adelante, como así también los *kd−trees* [5].

La SGB es una colección de programas y conjuntos de datos que generan una gran variedad de grafos y redes. Los programas en SGB están escritos en CWEB. CWEB es un sistema de programación literaria donde el programador, mientras escribe el código va generando el texto que lo explica. El código fuente cweb lleva extensión .w y es básicamente TeX enriquecido con comandos para expresar Lenguaje C. Una vez escrito un módulo *.w, debe compilarse en dos etapas. La primera es a través de un compilador llamado CTANGLE que toma un archivo *.w y genera un *.c. Luego, se pasa otra vez el archivo *.w por otro compilador llamado CWEAVE que genera el *.tex correspondiente. Si bien esta forma de desarrollar programas genera un producto terminado de muy alta calidad, pues tiene referencias cruzadas a definiciones de tipo, variables, funciones, etc., es sumamente engorroso de usar en proyectos grandes. Además de ilustrar los conceptos de programación literaria SGB tiene como propósito servir de plataforma de prueba a científicos del área de la optimización combinatoria.

[1] ftp://labrea.stanford.edu/pub/sgb/

Las estructuras de datos desarrolladas en SGB han tratado de ser independientes de la plataforma. El algoritmo principal para calcular la triangulación de Delaunay se basa en el modulo GB_PLANE del SGB. Si bien, en Internet ha sido facil encontrar innumerable cantidad de programas para calcular diagramas de Voronoi y/o triangulaciones de Delaunay, no sucede lo mismo con los grafos de vecinos relativos , ni los grafos de Gabriel. Para poder calcular estos últimos es que se utilizarán los $kd-trees$, que son estructuras de datos eficientes para manipular conjuntos semidinámicos de puntos. Estas estructuras de datos son básicamente árboles binarios de búsqueda que representan puntos en un espacio k dimensional. Los mismos son aptos para computar cierto tipo de búsquedas, como ser, correspondencia exacta o parcial, consultas de rango y pertenencia. Asímismo pueden manipular cierta forma limitada de borrados e inserciones. Se desarrolla también un algoritmo de complejidad temporal $O(N * LogN)$ para calcular el árbol de recubrimiento mínimo de un conjunto de N puntos en el plano euclideo. Es importante remarcar que el objetivo de este capítulo es mostrar **pseudocódigo** , no así código, para las diferentes estructuras de datos y algoritmos, entendiéndose que el estudio detallado de las diferentes implementaciones que se llevaron a cabo bien podría ser tema de extensos y numerosos capítulos. En tanto que en beneficio de claridad y brevedad en la exposición, numerosas estructuras de datos auxiliares han sido omitidas y reemplazadas aqui por código mas "legible" y por ende menos eficiente. Valga como ejemplo todos los testeos de pertenencia de un objeto a un dado conjunto. Bien es sabido que el problema de pertenencia es complejo en si mismo, necesitandose recurrir a funciónes eficientes de hashing o bien árboles balanceados para su implementación. En las porciones de pseudocódigo en que se necesita este tipo de cálculo se utilizan simplemente iteradores sobre listas.

5.1 La estructura de datos "Grafo"

Describiremos brevemente como representamos los grafos en este trabajo. Básicamente existen tres estructuras principales (tipo de dato **struct** en C):

- **Graph** *g;
- **Vertex** *u,*v;
- **Arc** *a;

si declaramos las variables g, u, v, a como arriba, entonces, g apuntará a un registro grafo, u y v serán vértices y a un arco entre algún par de vértices. Cada Grafo consta de una matriz de Vertex, donde cada Vertex apunta a una lista enlazada de arcos Arc. Si g es un grafo (declarada como arriba), entonces $g \rightarrow n$ nos indica la cantidad de vértices de g, que a su vez se acceden por $g \rightarrow vertices$. Dos vértices que son vecinos en g tendrán un arco dirigido entre uno y el otro. Los grafos se suponen dirigidos, entonces, para denotar un grafo no dirigido se debe contar con dos arcos, apuntando en direcciones opuestas. No se pueden representar multigrafos. Si v es definido como arriba, entonces $v \rightarrow arcs$ apunta al primer arco que emana de v. $v \rightarrow arcs \rightarrow next$ devolverá el siguiente y así sucesivamente. Si de v no sale ningún arco entonces $v \rightarrow arc = NULL$. Supongamos ahora que $a = v \rightarrow arcs$, entonces $a \rightarrow tip$ nos devuelve el vértice destino del arco a, y como se dijo anteriormente, $a \rightarrow next$ dará el siguiente

arco de v. Además, $v \rightarrow name$ es un puntero a una secuencia de caracteres, que sirven de nombre a v.

Para aclarar considérese el siguiente fragmento de código c:

```
for (v=g->vertices; v< g->vertices+g->n;v++)
{
 printf("%s:",v->name);
 for (a=v->arcs;a;a=a->next)
  printf(" %s",a->tip->name);
 printf("\n");
}
```

este código recorre todos los arcos de todos los nodos de un grafo imprimiendo los nombres. Además de los campos mencionados anteriormente, las estructuras Vertex y Arc tienen "utility fields". Estos campos son del tipo union en C y sirven para almacenar información externa al grafo, en particular permiten mantener estructuras de datos dinámicas ligadas al mismo. El campo $g \rightarrow m$, contiene la cantidad total de arcos de g, mientras que $g \rightarrow id$, hace las veces de $g \rightarrow name$ pero para g, $g \rightarrow data$ es un área de memoria donde son guardados (realmente) los vértices, arcos, etc, de g. Como los vértices y aristas, g tiene sus correspondientes "utility fields".

5.2 Pseudocódigo para Calcular la Triangulación de Delaunay

A continuación se desarrolla el pseudocódigo para calcular la triangulación de Delaunay de un conjunto de puntos. Como entrada al algoritmo se tiene un conjunto P de ciudades, la salida sera la triangulación de Delaunay del mismo.

```
Triangulacion de Delaunay(P)
Comenzar
 Sean p_1,p_2 y p_3 tres puntos en el plano tales que
 el triangulo que ellos forman contiene a P.
 Se inicializa T con el triangulo formado por p_1,p_2 y p_3.
 Se computa una permutacion random de las ciudades en P.
 Para r entre 1 y n hacer
 /* agregaremos p_r a la triangulacion T */
  Encontrar un triangulo p_i,p_j,p_k en T que contenga a p_r.
  Si p_r esta en el interior del triangulo p_i,p_j,p_k entonces
   Agregar aristas desde p_r a los tres vertices de p_i,p_j,p_k,
   esto es, el triangulo original se partio en 3.
   LegalizarArista(p_r,p_i p_j,T).
   LegalizarArista(p_r,p_j p_k,T).
   LegalizarArista(p_r,p_k p_i,T).
  Sino
   /* p_r esta sobre una arista de triangulo original
      supongamos p_i p_j */
   Agregar aristas desde p_r a p_k y a tercer vertice
   p_l del otro triangulo que es incidente a p_i p_j,
   partiendo de esta forma los dos tri\'angulos incidentes
   en p_i p_j en 4 tri\'angulos.
   LegalizarArista(p_r,p_i p_l,T);
```

```
   LegalizarArista(p_r,p_l p_j,T);
   LegalizarArista(p_r,p_j p_k,T);
   LegalizarArista(p_r,p_k p_i,T);
  Fin_Si
 Fin_Para
 Descartar p_1,p_2,p_3 y todas las aristas incidentes en los mismos.
 Retornar T.
Fin.
```

Ahora bien, la función LegalizarArista se encarga de mantener siempre una triangulación de Delaunay válida. Si encuentra que una de las aristas agregadas no es válida la cambia por otra que haga localmente Delaunay a la triangulación resultante.

```
LegalizarArista(p_r,p_i p_j,T)
Comenzar
 Si p_i p_j es una arista no valida entonces
  Sea p_i p_j p_k el triangulo adyacente a p_r p_i p_j sobre
  la arista p_ p_j.
  Reemplazar p_i p_j por p_r p_k.
  LegalizarArista(p_r,p_i p_k,T).
  LegalizarArista(p_r,p_k p_j,T).
 Sino
  Aplicar(G, p_i p_j).
 Fin_Si.
Fin.
```

Obsérvese que cuando finalmente se llega a una arista Delaunay válida, se aplica a dicha arista la función G. Esta es una función que definirá el usuario y servirá para determinar que hacer con dicha arista. En el caso de querer computar la triangulación de Delaunay lo único que se hará será incluirla en un grafo g mediante la función

```
void new_TSP_edge(Vertex* u,Vertex* v)
{
 long dx,dy;
 if(v&&u)
 {
  dx= (u->x_coord)-(v->x_coord);
  dy= (u->y_coord)-(v->y_coord);
  u->vecinos++;
  u->distancias+= int_sqrt(dx*dx+dy*dy);
  v->vecinos++;
  v->distancias+= int_sqrt(dx*dx+dy*dy);

  gb_new_edge(u,v,int_sqrt(dx*dx+dy*dy));
 }
}
```

Esta función simplemente recibe un par de vértices que determinan una arista de Delaunay válida y la incluyen en el grafo corriente (objeto global de tipo **Graph ***), incrementando la cantidad de vecinos Delaunay de cada vértice, la suma de las distancias a cada vecino y asigna a la arista en cuestión

su longitud. Los datos de cantidad de vecinos, longitud a los mismos, etc., se guardan en los ya mencionados "utility fields".

Para poder determinar en que triángulo se encuentra un determinado punto, a medida que se va construyendo la triangulación se mantiene un DAG, en donde cada nodo hoja representa uno de los últimos triángulos Delaunay encontrados. Si mas tarde el algoritmo procede a partir en dos o tres nuevos triángulos alguno de los representados por un nodo hoja, este se convierte en un nodo intermedio, y los dos o tres recientemente creados triángulos pasan a ser hojas. Debido a que el número máximo de aristas emergentes de cada nodo es 3, ubicar el triángulo al cual pertenece un punto es de order lineal en la cantidad de nodos de la trayectoria de búsqueda.

Destaquemos que el algoritmo que surge de implementar el pseudocódigo anterior tiene complejidad temporal $O(N * LogN)$ y espacial $O(N)$. Para un análisis mas detallado de las estructuras de datos y la complejidad temporal y espacial de estos algoritmos referirse por ejemplo a [7] o [25].

5.3 Los kd-Trees

Los *kd−trees* [2] son estructuras de datos especialmente aptas para manejar conjuntos de puntos semidinámicos. Por conjuntos semidinámicos entendemos a un conjunto de objetos conocidos inicialmente sobre los que se pueden aplicar operaciones de borrado y recuperación, es decir, borrados lógicos. Este es el caso que nos ocupa, pues nosotros conoceremos la disposición de las ciudades a recorrer de antemano. Como dijimos en la introducción, los *kd−trees* son árboles binarios de búsqueda que representan puntos en un espacio K dimensiones y sirven para constestar a cierto tipo de consultas que se formulan sobre los puntos. En particular soportan dos tipos distintos que son de interés en este trabajo:

- **nn** (consulta por el vecino más cercano): dado un punto retorna el vecino más cercano al mismo bajo una determinada métrica.
- **frnn** (consulta por los vecinos dentro de un radio fijo): dado un punto y un radio, reporta todos aquellos puntos incluidos en $B(p, r)$.

Existen dos tipos de nodos en los *kd−trees* los *internos* y los *externos*. Los internos particionan el espacio por un plano, "cut plane", definido por un valor en una de las k dimensiones. Los nodos externos o buckets, guardan los puntos en el hiperrectángulo resultado de la partición. Un nodo se representa por la siguiente estructura en C:

```
typedef struct kdnode
{       int     bucket;         /* booleano para diferenciar buckets de nodos internos */
        int     empty;          /* booleano para saber si el bucket esta vac\'{\i}o o no    */
        int     cutdim;         /* dimension del cut plane */
        long    cutval;         /* valor del cut plane */
        struct  kdnode  *loson; /* puntero a un kdnode con valores < cutval */
        struct  kdnode  *hison; /* puntero a un kdnode con valores > cutval */
```

[2] Esta sección es parte del trabajo [63] presentado en The VII Latin American Conference on Operational Research (C.L.A.I.O.), Santiago de Chile, Chile, Julio 4-9,1994

```
        struct  kdnode  *father;/* puntero al padre */
        KdItem  **permSeg;      /* si es un bucket apunta a perm  */
        int     len;
        long    bnds[DIMENSIONS][2];
        int     lopt;           /* parte baja de permSeg  */
        int     hipt;           /* parte alta permSeg */
} KdNode;
```

Donde la variable **bucket** es uno si el **kdnode** es un bucket y cero si es un nodo interno. En un nodo interno, **loson** y **hison** son punteros a subárboles con valores menores y mayores respectivamente a **cutval** en la dimensión **cutdim**. Un campo **empty** se le agrega a cada nodo en el árbol. Dicho campo se setea a 1 si todos los puntos en ese subárbol han sido eliminados, y 0 en otro caso. Los procedimientos de búsqueda tienen en cuenta este campo para retornar de un llamado ni bien comienza la evaluación de un subárbol que esta vacío. Mientras el árbol está siendo construido, dos punteros a **perm** representan un subconjunto de puntos. El árbol se construye con

```
for (i=0; i<n; i++)
 perm[i] = i;
root = build(0, n-1);
```

La función recursiva **build** es

```
kdnode *build(int l, int u)
{
 p = allocate_kdnode();
 if (u-l+1 <= cutoff)
 {
  p->bucket = 1;
  p->lopt = l;
  p->hipt = u;
 }
 else
 {
  p->bucket = 0;
  p->cutdim = findmaxspread(l,u);
  m=(l+u)/2;
  select(l, u, m, p->cutdim);
  p->cutval = px(m, p->cutdim);
  p->loson = build(l,m); p->hison = build(m+1,u);
 }
 return p;
}
```

La función **findmaxspread** retorna la dimensión en la cual los puntos están mas dispersos entre los representados por **perm**. La función **px** accede a la $j-$ésima coordenada de los puntos en **perm**. A su vez, **select** permuta **perm**$[l..u]$ de tal forma que **perm**$[m]$ contiene un punto que no es menor que ningún punto a su izquierda en la dimensión **p->cutdim**, ni tampoco mayor que los puntos a su derecha. La partición recursiva tiene lugar mientras $u - l + 1 \leq$ **cutoff**, esto es, cuando un nodo interno tiene menor o igual cantidad de puntos que los especificados por **cutoff** se convierte en un bucket, y termina la recursión. El procedimiento por el cual se construye el árbol, **build**, tiene complejidad temporal $O(K * N * logN)$ y espacial $O(N)$ [7].

5.3.1 Consultas en *kd−trees*

Si bien los *kd−trees* pueden ser utilizados para responder a varios tipos de consultas, nosotros estamos interesados en dos de ellas, a saber, el vecino más cercano a un punto dado(nn), y todos los vecinos que se encuentran dentro de un determinado radio(frnn). Ambas consultas se computan en tiempo logaritmico, sin embargo existen implementaciones donde la búsqueda puede empezar por un bucket, no por la raiz del árbol. Este tipo de búsquedas se llaman *bottom-up* y son particularmente útiles en el desarrollo de heurísticas rápidas para el *ETSP* [6]. El siguiente código responde a una consulta de vecino más cercano,

```
int nn( int aPoint)
{
 nntarget = aPoint;
 nndist = MAX_INT;
 rnn(root);
 return (nnptnum);
}

void rnn(KdNode * p)
{
    if (p->bucket)
    {
       for (i=p->lopt;i<=p->hipt;i++)
       {
          dist2=dist2(perm[i],nntarget);
          if (dist2<nndist2)
          {
           nndist2=dist2;
           nnptnum=perm[i];
          }
       }
    }
    else
    {
     val = p->cutval;
     thisx=x(nntarget,p->cutval);
     if (thisx<val)
     {
      rnn(p->loson);
      if(thisx+nndist2>val)
       rnn(p->hison);
     }
     else
     {
      rnn(p->hison);
      if (thisx-nndist2<val) rnn(p->loson);
     }
    }
}
```

para simplificar la exposición se han utilizado variables externas a fin de pasar información hacia y desde la función recursiva rnn la cual realmente realiza el trabajo. En un nodo bucket, rnn ejecuta una búsqueda secuencial por el

vecino más cercano al dado entre los pertenecientes al bucket. En un nodo interno, la búsqueda primero se realiza bajando por el hijo más cercano, y luego por el hijo lejano solo si es necesario. La función `x(i,j)` accede a la $j-$ésima dimensión del punto i. Estas funciones son correctas en cualquier métrica en la cual la diferencia entre las coordenadas de dos puntos en alguna de las k dimensiones nunca excede la distancia métrica [3].

La función para devolver todos los vecinos dentro de un dado radio, es muy similar y se omite. El lector interesado puede referirse a [5]. Existen ciertas optimizaciones a la hora de hacer los cálculos de distancias y la orientación de los planos de corte que mejoran la calidad del *kd−tree* y la performance, ver por ejemplo [63]. En la figura 5.1 se muestra el kdtree correspondiente a att532.tsp, cabe remarcar que el tamaño de cada bucket es 3.

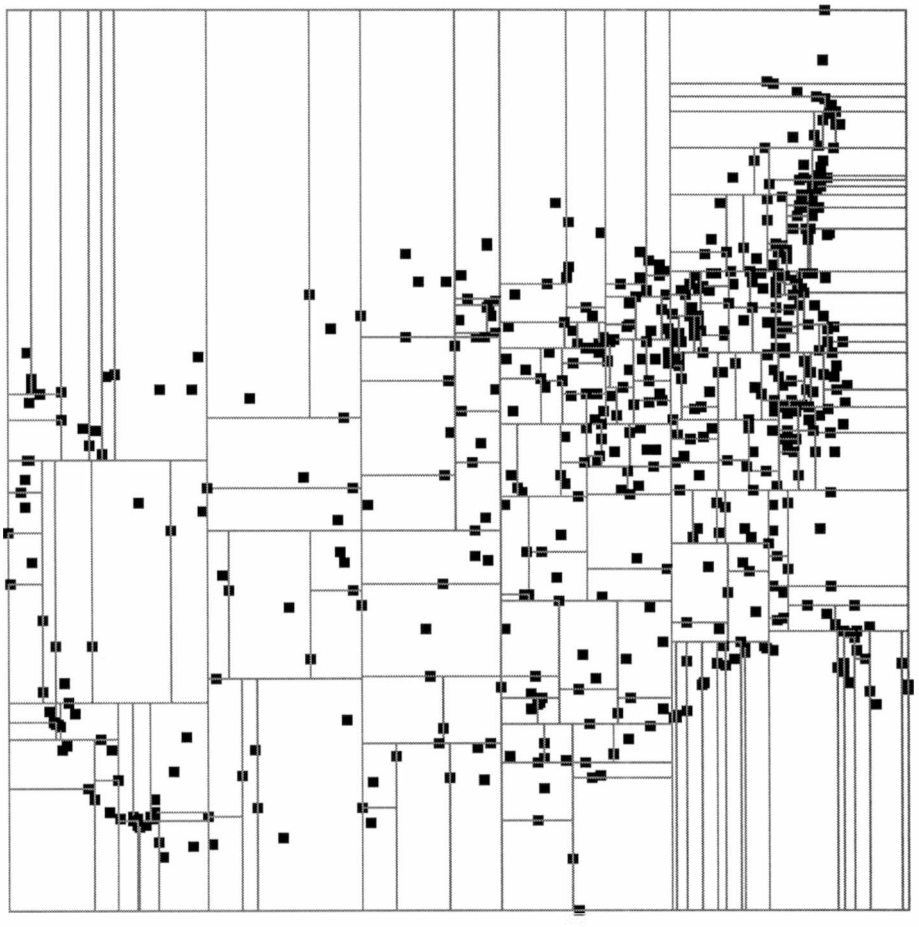

Figura 5.1: att532 - *kd−tree* , **cutoff**=3

[3]Las normas Minkowski $L_1, L_2, y L_\infty$ presentan dicha propiedad

5.3.2 Una implementación Eficiente de la Heurística "Nearest Neighbor"

Los *kd−trees* son estructuras de datos muy importantes a la hora de obtener implementaciones eficientes, estos es, con complejidad temporal de a lo sumo $O(N * LogN)$ de varias heurísticas para el *ETSP* . Sea como ejemplo la implementación de Nearest Neighbor. Básicamente es un algoritmo constructivo goloso que comenzando por una cualquiera de las ciudades a recorrer, elige en cada iteración la más cercana a la última ciudad incorporada al subtour.

```
void nntour(List * ciudades, int startCity, int *tour)
{
 /* carga perm a partir de ciudades,
    llama a la funci\'on build,
    y build construye un arbol
    que queda apuntado por tree */

 tour[0] = startCity;
 tree->delete(tour[0]);
 for (i=1;i<n;i++)
 {
  tour[i] = tree->nn(tour[i-1]);
  tree->delete(tour[i]);
 }
 free(tree);
}
```

Esta función incorpora de a una las n ciudades que componen un tour. Las ciudades originalmente están guardadas en `ciudades`. La función `nn` es invocada para que retorne el vecino más cercano a la ciudad que se le pasa como parámetro.

5.4 Pseudocódigo Para Calcular el Grafo de Vecinos Relativos

El cálculo de este objeto geométrico se basa en el código para desarrollar la triangulación de Delaunay y la utilización de los *kd−trees* . La diferencia esencial con respecto al algoritmo para calcular la triangulación de Delaunay esta en las siguientes funciones:

```
extern   KdTree *theTree;


char checkLuna(long u, long v, double radio)
{
 Lista listaU;
 Lista listaV;
 char  encontro;
 long  cDeU;
 long  cDeV;

 listaU=frnn(theTree,u,radio);
```

```
 /* obtiene todos las ciudades que distan menos que radio de la
    ciudad u */
 listaV=frnn(theTree,v,radio);
 /* obtiene todos las ciudades que distan menos que radio de la
    ciudad v */

 encontro=0;
 begin_iter(listaU);
 while((!end_iter(listaU)) && (!encontro))
 {
  cDeU = leer_iter(listaU)->city;
  begin_iter(listaV);
  while((!end_iter(listaV))&&(!encontro))
  {
   cDeV = leer_iter(listaV)->city;
   encontro= (cDeV==cDeU);
   next_iter(listaV);
  }
  next_iter(listaU);
 }

 vaciar(listaU);
 vaciar(listaV);
 return(encontro);

}


void new_TSP_edge(Vertex* u,Vertex* v)
{
 long dx,dy;
 double radio;

 radio = 0.0;
 dx    = 0;
 dy    = 0;
 if(v&&u)
 {
  dx= (u->x_coord)-(v->x_coord);
  dy= (u->y_coord)-(v->y_coord);

  radio = (double)(dx*dx+dy*dy);
  /* se pasa el radio al cuadrado */

  if(!checkLuna(u->z_coord,v->z_coord,radio))
   /* si no hay otra ciudad dentro de la luna agrega el link */
   /* utiliza el KdTree para verificar un frnn sobre u y v   */
  {
   u->vecinos++;
   u->distancias+= (long)sqrt(radio);
   v->vecinos++;
   v->distancias+= (long)sqrt(radio);
   gb_new_edge(u,v,(long)sqrt(radio));
  }
```

```
 }
}
```

Es decir, antes de agregar una nueva arista al grafo en cuestión se verifica que la luna determinada por los vértices que componen el arco (p, q), $\Delta_{p,q}$, no contenga a ningún otro punto del conjunto de ciudades. El costo extra de la verificación esta dado por las funciones **frnn**, cuya orden es $O(LogN)$.

5.5 Pseudocódigo Para Calcular el Grafo de Gabriel

Hemos dejado el pseudocódigo del grafo de Gabriel para despues de haber mostrado el del grafo de vecinos relativos porque es una variación del primero. Para que una arista pueda ser considerada parte del grafo de Gabriel no basta con verificar $(p, q) \in E \Leftrightarrow \Delta_{p,q} \cap V = \emptyset$, además no debe haber ninguna otra ciudad en el interior de $\Gamma_{p,q}$, excepto las que definen la arista. Luego, haciendo uso de la ecuación 3.5, modificamos checkluna de tal forma de obtener,

```
char checkEsferaDiametro(long u, long v, double radio)
{
 Lista listaU;
 Lista listaV;
 char  encontro;
 long  cDeU;
 long  cDeV;
 double minDistResto;
 double distResto;

 /* el radio viene siempre al cuadrado */
 listaU=frnn(theTree,u,radio);
 /* obtiene todos las ciudades que distan menos que radio de la
    ciudad u */

 listaV=frnn(theTree,v,radio);
 /* obtiene todos las ciudades que distan menos que radio de la
    ciudad v */


 encontro     = 0;
 minDistResto = 99999999.99999;
 distResto    = 0.0;
 begin_iter(listaU);

 while(!end_iter(listaU))
 {
  cDeU = leer_iter(listaU)->city;
  begin_iter(listaV);
  while(!end_iter(listaV))
  {
   cDeV = leer_iter(listaV)->city;
   if (cDeV==cDeU)
   {
    distResto = 0.0;
    distResto =(double)( distsq(u,cDeU)+distsq(v,cDeV));
```

```
    if (minDistResto> distResto)
       minDistResto = distResto;
   }
   next_iter(listaV);
  }
  next_iter(listaU);
 }
 encontro=radio>minDistResto;
 vaciar(listaU);
 vaciar(listaV);
 return(encontro);

}
```

La complejidad computacional de este algoritmo es igual al utilizado para el RNG, salvo constantes.

5.6 Pseudocódigo Para Calcular el Árbol de Recubrimiento Mínimo

El árbol de recubrimiento mínimo, MST en adelante, generalmente se encuentra clasificado bajo la categoría de problemas de "diseño de redes geométricas" cuyo objetivo es *conectar un conjunto de puntos mediante la construcción de una "buena" red.* Tómese como ejemplo válido la conexión de elementos VLSI por medio de cables tratando de minimizar el área de chip ocupada, el consumo de energía y la propagación de la señal en forma rápida. En tanto que problema surgido de la arquitectura de redes de telecomunicaciones, diseño de redes viales y análisis de imágenes médicas [1], generación de mallas y "robot motion planning" [13], el problema de la construcción del MST no es simple.

El estudio de algoritmos eficientes para calcular los árboles de recubrimiento mínimo se remonta a por lo menos 1926, cuando Otakar Boruvka describe el algoritmo "all nearest fragments". Los algoritmos clásicos para calcular el MST son debidos a Kruskal, Jarnik, Prim, y Boruvka. Implementaciones eficientes de los mismos pueden encontrarse por ejemplo en [25], donde se desarrollan las versiones de Tarjan, Cheriton y Karp. Básicamente los algoritmos para MST son variaciones sobre los siguientes conceptos:

1. unir dos fragmentos que sean más cercanos.
2. unir los vecinos más cercanos.
3. considerar todos los fragmentos más cercanos.

El algoritmo de Prim es el siguiente: el input es un grafo $G = (V, V * V)$ y la salida es un árbol T de recubrimiento mínimo.

```
T = empty;
Para i = 1 hasta n-2 hacer
 Elegir una arista de longitud minima, e, que sea adyacente a algun
 vertice en T.
 Si (no_hay_ciclos(T+e)) entonces
  T = T+e;
```

el algoritmo de Kruskal recibe el mismo input y genera la misma salida que el anterior:

```
T =  empty;
Para i = 1 hasta n-1 hacer
 Elegir una arista de longitud minima, e.
 Si (no_hay_ciclos(T+e)) entonces
  T = T+e;
```

Notar que la diferencia entre el algoritmo de Kruskal y el de Prim radica en que el último requiere que la arista de longitud mínima elegida sea incidente en el arbol parcial, T, que se va construyendo, y en Kruskal eso no es requisito. Es decir, Prim responde al segundo criterio mencionado, mientras que Kruskal al tercero. El costo de estos algoritmos, $O(|V*V|*Log|V*V|)$, viene del proceso de ordenamiento de las $V*V$ aristas del grafo de menor a mayor. Esto no es aceptable en nuestro caso ya que deseamos obtener heurísticas $O(N*LogN)$ en la cantidad de ciudades y no $O(N*N)$. Para solucionar este problema y obtener un algoritmo de generación del EMST con la complejidad deseada apelaremos al siguiente lema cuya demostración puede encontrarse en la página 220 de[35]:

Lema 5.1 *Sea S un conjunto de puntos en el plano, y $\Delta(p)$ el subconjunto de puntos de S adyacentes a p en la triangulación de Delaunay DT, con $p \in S$. Para toda partición $\{S_1, S_2\}$ de S, si (p,q) es el segmento más corto entre puntos de S_1 y S_2, entonces $q \in \Delta(p)$.*

Lo que este lema nos dice es que podemos utilizar la triangulación de Delaunay para trabajar sobre un conjunto de aristas más reducido (solo las aristas Delaunay) con alguno de los algoritmos de Prim o Kruskal. Como DT se computa en $O(N*LogN)$ y la cantidad de aristas de DT es $O(N)$, bajamos la complejidad de dichos algoritmos a la mencionada.

El algoritmo usando Kruskal queda:
dado S conjunto de vértices:

```
Calcular TD(V).
Ordenar aristas de menor a mayor.
MST = {}
Tam_MST = 1.
Mientras Tam_MST<|S|-1 hacer
 a = obtener la menor arista disponible.
 Si no hace-ciclo(a)
  agregar(a,MST).
  Tam_MST = Tam_MST + 1.
 Fin_Si
 eliminar a.
Fin_Mientras.
```

Este esquema es esencialmente similar al anterior. Las aristas se ordenan de menor a mayor y de a una por vez se las va considerando para ser incluidas en el MST. El algoritmo requiere una cuidadosa codificación de `hace-ciclo()` y `agregar()`. En [11] se presenta un algoritmo eficiente para llevar a cabo estas tareas. La idea central al implementar dichas funciones es ir guardando en un árbol balanceado el bosque que se va formando con las aristas que se agregan

en $agregar(a, MST)$, de tal forma de poder determinar en $Log(N)$ el árbol al que pertenece a.

Existe la posibilidad de utilizar Prim, Delaunay y los $kd-trees$ para obtener otro algoritmo diferente de complejidad temporal $O(N * LogN)$; en el cual la triangulación solo se utilizaría para obtener la arista más corta y luego los queries $kd-trees$ para unir los fragmentos.

5.7 Conclusiones

En esta sección se presentaron pseudocódigo y código para la obtención de objetos geométricos complejos en tiempo proporcional a $O(N * LogN)$, siendo N la cantidad puntos. Es importante notar que si bien existen varios algoritmos para computar la triangulación de Delaunay o los diagramas de Voronoi disponibles en Internet, encontrar alguno que sea numéricamente estable y portable a plataforma PC es muy difícil. Los algoritmos para MST disponibles via ftp son $O(N * LogN)$ en la cantidad de aristas, no así en los vértices como se requiere en este trabajo. Por ello se desarrollaron dos algoritmos $O(N * LogN)$ en la cantidad de ciudades haciendo uso de la triangulación y los $kd-trees$. Asimismo se implementaron algoritmos eficientes para computar RNG y GG. No fue posible encontrar hasta la fecha algorimos eficientes y portables a PC para generar estos grafos.

5.8 Notas:

Capítulo 6

Búsqueda Heurística: Algunas Consideraciones

Es un error capital teorizar antes de tener datos. Sin darse cuenta, uno empieza a deformar los hechos para que se adapten a las teorías, en lugar de adaptar las teorías a los hechos.

Escándalo en Bohemia
A.C. Doyle

En los capítulos anteriores se estudiaron conceptos de diversas disciplinas necesarios para poder formular, en los capítulos por venir, algunas nuevas heurísticas para el *ETSP* . Es ahora el momento de hacer algunas consideraciones respecto de la evaluación por medio de experimentos computacionales de nuevas teorías, hipótesis,conjeturas o algoritmos para la resolución de problemas. Si bien las heurísticas han formado parte de la cultura humana por milenios, y su utilización ha trascendido el quehacer diario del hombre desde mucho tiempo atrás, recien en los últimos tiempo sus versiones matemáticas han crecido en número, complejidad y aplicaciones. Las nuevas heurísticas hacen posible que hoy en día científicos de investigación de operaciones, ciencias de la computación e ingenieros puedan resolver problemas que hasta ayer eran intratables ya sea por su complejidad o tamaño. John Holland en su libro más reciente [60] nos dice lo siguiente :

> "Computer-based models nicely integrate the themes exemplified by games, numbers, and building blocks. To implement a model on a computer, we first determine the model's principal components - the model's building blocks. Then we implement these components as sets of instructions in the computer called subroutines. Finally, the subroutines are combined in the computer in a way that determines their interactions, yielding the overall program that defines the model. The result is a computer-based realization of the rules that define the model's behaviour."

Y es así como debiesen ser presentadas las heurísticas para su discusión en el ámbito científico, debe quedar claro cuales son los bloques constructivos de las mismas, que aporte se supone hará cada uno de ellos, y por sobre todas las cosas, cuál es el modelo del mundo que la heurística intenta capturar y aprovechar.

Holland continua diciendo,

> "Computer-based models are at once abstract and concrete. They are abstractly defined in terms of numbers, relations between numbers, and changes in numbers over time - a feature they share with mathematical models. At the same time, the numbers are actually *written down* in the computer's registers, rather than being represented symbolically. Moreover, the numbers are overtly manipulated by the computer's instructions, much as a grain mill produces flour. We can produce quite concret records of these manipulations. These records are closely related to the laboratory notebook records of a carefully run experiment. Computer-based models, then, partake of features of both theory and experiment. This combination of the abstract and the concrete offers advantages and also disadvantages."

De este modo las heurísticas, en tanto que modelos del mundo, deben ser cuidadosamente especificadas y descriptas tratando de diferenciar claramente la "idea" por la cual la misma opera, de la mecánica (una particular implementación) que emplea para cumplir su cometido. En la medida de lo posible se debería poder presentar cotas en la complejidad temporal y espacial de la idea de la heurística, además de un estudio de las particulares implementaciones. La complejidad de la "idea" será el factor determinante de la aplicabilidad de una dada heurística ya que nos habla de la complejidad intrínseca del modelo del mundo adoptado. Si es posible, es deseable también contar con un estudio empírico de los tiempos y memoria de los que la mecánica de la heurística hace uso. Debido a la gran cantidad de plataformas computacionales, sistemas operativos y lenguajes de desarrollo, esto último se torna un trabajo sumamente complejo, además de estéril. Imagínese el lector, comparar los tiempos de corrida de un programa en un DEC Alpha, contra la misma implementación portada a un sistema multiusuarios en un SUN Workstation o una PC. Un paso importante en este sentido es el dado por Donald Knuth en [25], cuando trata sobre algoritmos para calcular el *MST* :

> "When the code for MILES_SPAN was being written, there was no good way to predict which of the four methods would be fastest... Nor was it clear how to make fair comparison of the methods - a comparison that would be meaningful on more than one computer and in the future as well as today... Experiments have shown, however, that a remarkably simple idea provides an excellent way to compare combinatorial algorithms such as this, namely the technique of *mem* counting. Each algorithm's running time is reported by MILES_SPAN in terms of mems, meaning references to memory..."

Una de las desventajas que heredan las heurísticas por su condición de estar a medio camino entre un modelo matemático formal y un experimento en particular es que cada vez que una nueva heurística es presentada en la literatura

especializada, su contribución debe ser analizada con todo el rigor científico. Sin embargo esto no siempre sucede.

Debido a que un algoritmo es una abstracción, es evaluado indirectamente por medio de la experimentación con una dada implementación. En general podemos decir que un experimento es una serie de tests que se corren bajo condiciones controladas con cierto propósito, a saber, demostrar una verdad conocida, verificar la validez de una hipótesis o controlar la performance de algún algoritmo original. Los experimentos son ubicuos a la actividad científica y forman parte esencial del método científico. Es el mecanismo a través del cual la ciencia se corrige y verifica a si misma. Por el experimento se demuestran teorías, se releva conocimientos sobre algún campo oscuro y se miden los efectos de factores o fenómenos bajo control del científico. La ciencia de la computación es muy joven en relación a las ciencias de la vida, físicas o químicas, donde el experimento ha desempeñado un papel central en sus respectivos desarrollos. Es así que los científicos de la computación todavía no se sienten cómodos frente a la necesidad, impuesta por las circunstancias, de experimentar, situación que por otro lado esta cambiando rápidamente; prueba de ello es la creación del "Journal of Experimental Algorithms", "Journal of Heuristics", etc.

En general podemos decir que las pruebas experimentales de algoritmos se basan en resolver una serie de instancias de prueba usando cierta implementación computacional. Básicamente la exprimentación consiste en [2]:

1. Definir los objetivos del experimento.
2. Elegir las medidas de performance y factores a explorar.
3. Diseñar y ejecutar el experimento.
4. Analizar los datos y arrojar conclusiones.
5. Reportar los resultados de lo experimentado.

En el trabajo mencionado se reconoce que si bien no hay un estandar para la publicación de investigación en algorítmica, en general se considera que una heurística hace una contribución original si presenta alguna de las siguientes propiedades [2]:

- **Fast** - producing high-quality solutions quicker than other approaches.
- **Accurate** - identifying higher-quality solutions than other approaches.
- **Robust** - less sensitive to differences in problem characteristics, data quality, and tunning parameters than other approaches.
- **Simple** - easy to implement.
- **High-Impact**- solving a new or important problem faster and more accurately than other approaches.
- **Generalizable**- having application to a broad range of problems.
- **Innovative** - new and creative in its own right.

- **Revealing** - offering insight into genera heuristic design or the problem structure ; stablishing the reasons for its performance and explaining its behaviour.
- **Theoretical**- Providing theoretical insights, such as bounds on solution quality.

6.1 Heurísticas para el *ETSP*

En esta sección haremos un brevísimo repaso de las heurísticas constructivas e iterativas para el *ETSP* que se mencionarán en los próximos capítulos. Los algoritmos que repasaremos son:

- **OCI** - One City Insertion.
- **NN** - Nearest Neighbor.
- **2-Opt** - 2-Opt.
- **LK** - Lin-Kernighan.

Las cuatro heurísticas mencionadas han recibido muchisima atención en la literatura, el lector interesado en un estudio más detallado puede referirse a [6], [26], [28], [33], [40], [71], [83].

6.1.1 OCI

Al igual que NN, *One City Insertion* es quizás una de las más simples heurísticas para el *ETSP* . El procedimiento de construcción del tour comienza eligiendo una ciudad al azar. A partir de esta ciudad se incorpora cualquier ciudad aún no visitada en la posición relativa que minimice la longitud del nuevo tour. Evidencia empírica sugiere que la calidad de los tours calculados por este método es buena [71].

6.1.2 NN

Esta heurística, cuyo código se presentó en la sección 5.3.2 puede aparecer en dos versiones a saber, *Nearest Neighbor-MCS* y *Nearest Neighbor-FCS*. El algoritmo descripto en la sección 5.3.2 responde a la segunda versión consistente en comenzar por alguna ciudad elegida al azar y enlazar al final del tour parcial la ciudad más cercana a dicho extremo, FCS es la abreviación de "Fixed City Start". Por el contrario, NN-MCS, toma cada una de las ciudades y a partir de ellas crea un nearest neighbor tour devolviendo el más corto de ellos. Las siglas corresponden a "Moving City Start". La performance de estos algoritmos puede ser arbitrariamente mala.

6.1.3 2-Opt

Este algoritmo es una instancia del más general $\lambda - Opt$. Se comienza construyendo alguna solución factible, esto es, un tour que pase por todos las ciudades solo una vez y retorne a la primera ciudad visitada. Luego, mientras

se encuentren λ aristas a intercambiar de manera que minimicen la longitud del tour resultante se realiza el cambio y se intenta nuevamente. Cuando no exiten ningun conjunto de λ aristas a cambiar se dice que el tour resultado es $\lambda - Opt$. Cada intercambio de aristas se llama λ−swap. Por ejemplo, si tenemos un conjunto de cuatro puntos a recorrer, digamos $a, b,$ c y d, comenzamos armando el tour $a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$. Un λ−swap podría generar el tour $a \rightarrow d \rightarrow b \rightarrow c \rightarrow a$, donde se intercambiaron las aristas $d \rightarrow$ c y $b \rightarrow a$ por $d \rightarrow b$ y $c \rightarrow a$. La heurística λ-Opt devuelve tours de buena calidad.

6.1.4 LK

Papadimitriou en [83] reconoce:

> "Perhaps the most famous and succesful local search heuristic is the one proposed by Lin and Kernighan in 1973 for the Traveling Salesman Problem."

En esencia LK consiste en una inteligente combinación de λ−swaps haciendo una búsqueda primero en anchura y luego otra en profundidad. En [28] se presenta el siguiente pseudocódigo para LK:

"

1. Generate a random initial solution T.
2. (a) Set $i = 1$;
   (b) Select x_i and y_i as the most-out-of-place pair at the i^{th} step. This generally means that x_i and y_i are chosen to maximize the improvement when $x_1, \ldots, x_i$ are exchanged with $y_1, \ldots, y_i$. x_i is chosen from $T - \{x_1, \ldots, x_{i-1}\}$ and y_i from $S - T - \{y_1, \ldots, y_{i-1}\}$.
   (c) If it appears that no more gain can be made, according to an appropiate stopping rule, go to Step 3; otherwise, set $i = i + 1$ and go back to step 2.b .
3. If the best improvement is found for $i = k$, exchange $x_1, \ldots, x_k$ with $y_1, \ldots, y_k$, to give a new T, and go to Step 2; if no improvement is found, go to Step 4.
4. Repeat from Step 1 if desired.

"

En los primeros estudios sobre la clase PLS la gran ausente era la heurística de Lin y Kernighan para el viajante de comercio. Recien en [83] se demuestra que LK es PLS-Completa a través de una reducción de *2SATFLIP*. Este resultado tiene como consecuencia importante que se pueden construir instancias del *TSP* para las cuales LK necesite recorrer un camino exponencialmente largo antes de converger. Sin embargo en numerosos experimentos computacionales se observó su buen desempeño tanto en tiempo de ejecución como en la calidad de las soluciones encontradas.

6.2 Conclusiones

Cuando se analiza una nueva heurística, cualquiera sea la contribución de la misma, los experimentos computacionales serán necesarios a la hora de demostrar lo que los autores de la misma proponen. Además, en general se obsevarán dos tipos de experimentos, algunos para comparar la eficiencia/eficacia de diferentes algoritmos bajo un mismo tipo de instancias de prueba, y otros para caracterizar y describir el comportamiento de cierto particular algoritmo en forma aislada.

Desde una perspectiva más amplia [60],

> "It is usual to think of models as validated through the correctness of their predictions about the world, but we have seen at least two other roles for models in science. Models in these other roles have a different kind of validation. One role is to provide a rigorous demonstration that something is *possible*, as in von Neumann's demonstration that a machine is able to reproduce itself. Here validation occurs when a dynamic model works as claimed, as when one validates a patented device. Another role is served when the model suggest *ideas* about a complex situation, suggesting where to look for critical phenomena, points of control, and the like."

Adhiriendo a las ideas vertidas arriba quisás sean estos dos, los roles más importantes de las heurísticas. Por un lado demostrar que existen soluciones factibles, de buena calidad, construibles en tiempos razonables a partir de algún concepto que anteriormente había sido ignorado o insuficientemente estudiado. A la vez que se convierten en una fuente inagotable de caminos a explorar, experimentar e ideas a corroborar, fertilizando campos del conocimiento humano que bién hubiesen podido permancer aislados.

En este capítulo describimos brevemente las heurísticas a partir de las cuales se trabajará en el resto del trabajo. Debemos remarcar que sobre estas heurísticas se han hecho muchas variaciones y combinaciones. En innumerable cantidad de publicaciones se menciona el hecho de que cambiando el orden de selección de las ciudades a la hora de elegir la siguiente con la cual optimizar, los resultados de los algoritmos serían bien diferentes. En ninguno de ellos hemos encontrado un análisis detallado de este aspecto de las heurísticas descriptas. Parte del objetivo de este trabajo es comenzar con el análisis del correcto orden de consideración de las ciudades para OCI.

6.3 Notas:

Capítulo 7

OCIG, Una Nueva Heurística Híbrida

It's either this or that way
it's one way or the other
it should be one direction
it could be on reflexion
the turn I have just taken
the turn that I was making
I might be just beginning
I might be near the end.

Memory of Trees
Enya

Este capítulo presentará un conjunto de nuevas heurísticas basadas en un híbrido de la triangulación de Delaunay o alguno de sus subgrafos y "One City Insertion" como método constructivo. Estas nuevas heurísticas híbridas de baja complejidad temporal, a saber $O(N * LogN)$ siendo N la cantidad de ciudades, son aptas [1] para instancias del *ETSP* de gran porte en el plano. El método se basa en la construcción de un grafo G a partir del conjunto de ciudades a recorrer. G puede ser la triangulación de Delaunay o alguno de sus subgrafos. En tanto que subgrafos de la triangulación, las posibilidades son: el grafo de Gabriel, el de vecinos relativos o el árbol de recubrimiento mínimo. Muchas heurísticas para el *ETSP* generalmente estan compuestas de dos fases: un método de construcción y un paso de mejoría, hasta que se llega a un óptimo local. Consideramos que el esquema presentado en este capítulo es un híbrido ya que después que una ciudad es introducida, un período de búsqueda local nos lleva a otro mínimo local, antes que una nueva ciudad sea considerada para ser incorporada al tour.

[1] La parte relativa a la triangulación de Delaunay de este capítulo fue presentado originalmente en el congreso de la Sociedad Brasilera de Informática y Pesquisa Operacional, Victoria, Brasil, 1995 [64]

7.1 OCIG, Presentación

Mas de una década atrás, se probó que el tour óptimo de un conjunto de puntos en el plano no esta necesariamente incluido en TD [18], cerrando con esto una conjetura pendiente. Sin embargo, como sugieren los experimentos realizados en el capítulo 4, un porcentaje muy alto de los links de los tours óptimos disponibles en TSPLIB pertenecen a TD,GG o RNG. El uso de conjuntos candidatos es una técnica estandard para mejorar la eficiencia de heurísticas de tipo *Hill Climbers* [80] y metaheurísticas basadas en Búsqueda Tabu [31]. En el contexto de TD, los conjuntos candidatos han sido utilizados en [37]. Nuestro interés en G también deriva de la consideración de otra estrategia, FAMCH [33], que es a su vez una variación de una heurística estudiada por D.S. Johnson en [16]:

> "Farthest Addition from Minimal Convex Hull (FAMCH): Start from the (possibly incomplete) tour comprising the minimal convex hull (that is the convex hull from which have been removed all vertices to which adjacent edges are at an angle of 180 degrees). Repeatedly choose the non-tour city with the maximal distance to its nearest neighbor amongst the tour cities, breaking ties randomly, and insert it between the two consecutive cities in the subtour for which such an insertion causes the minimum increase in total tour length, breaking ties randomly."

En [33] se aplico FAMCH a 4 instancias *fractales* del *ETSP* , cada una de las cuales tiene diferente box-counting dimension, resolviendo a optimalidad la instancia KochTour. Esta última es la que tiene menor dimensión fractal de las estudiadas en dicho paper. A su vez, falla en devolver el óptimo en dos instancias con dimension fractal 2 y en DavidTour, una instancia con regiones vacías en cualquier escala.

La pregunta sobre el orden apropiado de inserción es la que nos hace considerar diferentes variantes del procedimiento de selección mencionado arriba:

> "...Repeatedly choose the non-tour city with the maximal distance to its nearest neighbor amongst the tour cities, breaking ties randomly..."

La TD fue entonces un buen candidato para dar una definición precisa de vecindad de una ciudad en un dado conjunto, es así que se asocia a G con TD. Una vez instanciado OCIG con DT, llamamos a la nueva heurística OCIDT. Como primer criterio de selección se eligió el grado de una ciudad en TD, prefiriéndose dar prioridad a aquellas con mayor grado en el grafo. Un punto interesante a considerar sobre esta estrategia es el siguiente: se puede asociar una superficie poliédrica a cada triangulación, en particular a TD, forzando que todos los links sean de longitud unidad. Es claro que dicha superficie, compuesta por triángulos equiláteros, no puede ser chata, excepto cuando una ciudad tiene 6 vecinos en TD. Cinco vecinos tendrían curvatura positiva, mientras que siete formarían una ensilladura de curvatura negativa [29]. Esto es, tenemos como primer criterio de selección el grado de una ciudad en TD, dicho grado dará el ranking de cada ciudad al ser considerada para inserción. Este criterio puede ser reinterpretado, o mejor aún "refinado", en futuros estudios sobre la base del concepto de curvatura y otros afines que se discuten en [29].

Ahora bien, como es de esperar, muchas ciudades en TD tendrán el mismo grado. Por esto, necesitaremos otro criterio de decisión. Por el momento llamaremos a este criterio $H\#$ y haremos explícita la estructura básica de OCIDT. OCIDT deriva de las iniciales *"One-City-Insertion from DT"*.

OCIDT básico
*Crear TD, la Triangulación de Delaunay del conjunto de ciudades S.
*$G = TD$.

```
Repetir hasta que todas las ciudades hayan sido incorporadas al tour:
 Encontrar la ciudad todavia no incorporada al subtour con el mayor grado en G.
 Decidir los empates, si los hubiese, segun H\#
 Insertar la ciudad elegida entre dos ciudades consecutivas en el subtour para
  las cuales dichas insercion cause el minimo incremento en la longitud
  total del mismo.
 Decidir los empates al azar.
 Disparar un paso de mejora iteractiva OCI.
```

Remarquemos el hecho que la mejora interactiva se realiza cada vez que una ciudad es incorporada, reduciendo así dramáticamente las inserciones a verificar. Es interesante ver como evoluciona la heurística; aveces una cascada de eventos cambia significativamente la forma global del subtour. OCIDT tiene baja complejidad temporal, el paso más costoso es el cálculo de la TD, es decir, $O(N * LogN)$, siendo N la cantidad de ciudades. Se utilizó la movida *one-city-insertion*, que consiste en verificar la inserción de una ciudad entre otras dos, debido a su simplicidad y baja complejidad computacional. Otras movidas como el *2-change* del *2-Opt*, más poderosa que el OCI, mejorarían el esquema OCIDT. Sin embargo como lo que se pretende mostrar es la viabilidad de la utilización de TD como fuente de información a heurísticas para el *ETSP* , preferimos no enmascarar los resultados al usar una búsqueda local mas poderosa.

De igual forma como se hizo al asociar G con TD, obtenemos las siguientes 3 heurísticas, OCIGG, OCIRNG y OCIMST, cuando consideramos $G \in \{GG, RNG, MST\}$. Los correspondientes y similares pseudocódigos se obtienen cuando reemplazamos las líneas con * en **OCIDT básico** por

*Crear GG, el grafo de Gabriel del conjunto de ciudades S.
*$G = GG$.

*Crear RNG, el grafo de vecinos relativos del conjunto de ciudades S.
*$G = RNG$.

*Crear MST, el árbol de recubrimiento mínimo del conjunto de ciudades S.
*$G = MST$.

para los tres casos respectivamente.

7.2 Experimentos Computacionales

En la literatura del *ETSP* , las heurísticas constructivas e iterativas son aplicadas, generalmentem en "tandem", ya que el paso iterativo es usualmente aplicado a un tour completo. En nuestro caso, la hipótesis de trabajo del método

híbrido es que podemos explotar la correlación de mínimos locales de dos tours parciales que solo difieren en una ciudad. La importancia de la correlación de mínimos locales para algunas metaheurísticas ha sido discutido en [31]; F. Glover también se refiere a esto como *"Proximate Optimality Principle"*. En forma natural, nos enfrentamos entonces al problema (ya mencionado antes) de un orden apropiado de inserción. En este capítulo reportaremos resultados computacionales con cuatro criterios de selección diferentes (H1,H2,H3 y H4) definiendo así cuatro heurísticas. Cada H_i se define como un par $(\pm grado, criterio)$. La primera componente nos da el grado de una ciudad en G. Por su parte *criterio* puede ser, por ejemplo, la suma de longitudes de las aristas a los vecinos en G de una dada ciudad, anl(all nearest length), o la distancia al vecino más próximo, nnl (nearest neighbor length). Así queda definido H# en sus 8 versiones:

1. $H1 = (grado, anl)$.
2. $H2 = (grado, -anl)$.
3. $H3 = (grado, -nnl)$.
4. $H4 = (grado, nnl)$.
5. $H5 = (-grado, anl)$.
6. $H6 = (-grado, -anl)$.
7. $H7 = (-grado, -nnl)$.
8. $H8 = (-grado, nnl)$.

En los casos H1 y H2 computamos para cada ciudad la suma de las longitudes de todos las aristas en G que tienen a dicha ciudad como uno de sus vértices. Usando esta suma, H1 rankea las ciudades en orden decreciente mientras que H2 lo hace en orden creciente. Cuando varias ciudades pueden ser seleccionadas, debido al hecho que tienen el mismo número de vecinos, H1 las inserta en orden decreciente, mientras que H2 lo hace al revés. Para H3 y H4 el criterio asignado es el de desempatar mediante la distancia de cada ciudad a su vecino más próximo. De esta forma, H3 inserta primero aquellas ciudades que están cerca de su primer vecino, cuando H4 da preferencia a aquellas alejadas de su primer vecino. Se utilizan los dos criterios, H1 y H2, para testear la consistencia de la suma de las longitudes de las aristas, y de igual forma sucede con H3 y H4. Debemos hacer este análisis para verificar que dicha suma, respectivamente la distancia al vecino más próximo, sea una medida segura de desrandomización del criterio de selección basado solo en el grado de las ciudades en G. Si consideramos $H5 \ldots H8$, las segundas componentes del par cumplen la función detallada en las líneas precedentes, sin embargo debido al cambio de signo en la primera componente se privilegia a aquellas ciudad con menor número de vecinos.

7.3 Resultados Experimentales

En las tablas que aparecen en los apéndices se listan las instancias por sus nombres y las longitudes de los tours obtenidos con las diferentes heurísticas. El nombre de cada instancia consta de letras y números, donde los números indican

la dimensionalidad del problema. Sea como ejemplo att532, esta instancia del *TSP* es de 532 ciudades, y se puede encontrar en TSPLIB como att532.tsp .

En el apéndice A aparecen el nombre de cada instancia y su correspondiente óptimo o mejores valores conocidos. Las columnas "límites" como su nombre indica muestra cotas conocidas para la longitud del tour óptimo, que vienen de una variedad de fuentes, ver los agradecimientos de [37].

En el apéndice B se muestran las mismas instancias que en A, para las cuales se corrió OCIG en sus 4 versiones. Se observa una columna con el nombre de la instancia, otra con la longitud obtenida, y la última con el porcentaje sobre el óptimo.

A modo de comparación hemos considerado ilustrativo incluir los resultados de G. Reinelt, quien utiliza la TD de una manera diferente. En sus trabajos usa la TD y DG para definir un subgrafo del grafo completo sobre todas las ciudades, pero supergrafo de TD, y desarrolla una variante de la clásica heurística del vecino más cercano. Recordemos que el grafo de Delaunay, DG, se crea a partir de TD excluyendo aquellas aristas pq para las cuales $\mid V_R(p) \bigcap V_R(q) \mid = 1$, esto es, sus celdas de Voronoi se intersectan en un punto. En contraste con TD, DG ya no es una triangulación pero sigue siendo un grafo planar implicando[37] que $\mid DG \mid = O(N)$. Los resultados se muestran en dos grupos. En el apéndice B presentamos los valores obtenidos con nuestras heurísticas sobre la mayoría de las instancias usadas por Reinelt en [37]. Ademas se hicieron análisis similares para muchas otras instancias geométricas de TSPLIB, resultados que también se exponen. Los valores de Reinelt son aquellos que pertenecen a la tercera variante de las heurísticas presentadas por Reinelt en [37], quien la indica como la mejor ya que la calidad promedio para el 2-opt y Lin-Kernighan es de 7.31% y 3.59%, mientras que el valor promedio para NN es 18.69%. Una variante distinta que usa como grado límite 3 en la definición de vecindario en [37] lleva a resultados inferiores, 7.83% para 2-opt, 4.05% en Lin-Kernighan y 20.73% con NN. Incluimos los mejores resultados de [37] que aparecen en la tabla V de dicho trabajo. Las columnas $H1, \ldots, H8$ contienen la longitud del tour encontrado por nuestras heurísticas y el porcentaje sobre el óptimo usando el límite inferior o el valor óptimo según corresponda.

7.4 Conclusiones

En este capítulo se definió una familia completa de nuevas heurísticas a saber, OCIDT, OCIGG, OCIRNG y OCIMST. Cada una de ellas fué instanciada en 8 versiones, obteniendose un total 32 nuevos algoritmos. Las 32 heurísticas tienen un tiempo de corrida esperado de orden $O(N * Log\ N)$ para el *ETSP* simétrico en $2D$. Los resultados de las simulaciones resultan comparables en calidad a métodos más complejos y de mayor complejidad computacional como pueden ser 2−Opt y Lin-Kernighan aplicados en tandem al método de vecindarios de Reinelt. Los promedios obtenidos con nuestras heurísticas son mejores que aquellos de Reinelt. Si bien no mostramos tablas con los tiempos de corridas (por motivos expuestos con anterioridad), mencionemos que los de las heurísticas presentadas aquí son mejores que aquellos de [37]. Téngase en cuenta además, que todas las corridas fueron hechas en una PC 383 con 16MB de memoria RAM.

7.5 Notas:

Capítulo 8

Conclusiones Generales

El peligro de la subjetividad y el prejuicio ha estado claro desde el principio de la historia.

El Mundo y sus Demonios
C. Sagan

La primera ley es que el historiador no debe osar jamás escribir lo que es falso; la segunda, que no osará jamás ocultar la verdad; la tercera, que no debe haber sospecha en su obra de favoritismo o prejuicio.

Cicerón

En este capítulo resumiremos los principales resultados y conclusiones de este trabajo.

8.1 Lo que pudimos conocer

Para llevar a buen término nuestra investigación se han repasado algunos conceptos básicos de *complejidad computacional*, *optimización combinatoria* y *teoría de algoritmos*. Resumimos algunos aspectos de la NP-Completitud del *TSP* , y algunos algoritmos de aproximación y PTAS para el mismo. Es fundamental para poder mejorar y desarrollar nuevas heurísticas para un problema dado, conocer cuál es el estado del arte y contra qué tipo de enfoques hay que competir.

Asímismo se presentaron los conceptos básicos de geometría computacional que fueron utilizados a lo largo de todo el trabajo. Se definieron los problemas de proximidad más comunes y los grafos de vecinos relativos asociados a algunos de estos problemas. Se presentó una lista con propiedades interesantes de dichos objetos geométricos.

En el capítulo 4 se planteó la problemática de la definición de un vecidnario de búsqueda adecuado para el *ETSP* . Si bien se demostró en [18] que en general no se puede esperar encontrar al tour óptimo incluido en TD, comprobamos empíricamente que la gran mayoría de las aristas de los óptimos pertenecía

a TD, abriendo así una puerta a nuevas heurísticas, a la vez que plantea el problema de identificar el motivo por el cuál algunas aristas óptimas no están en TD.

Se desarrolló el pseudocódigo y código para la obtención de objetos geométricos complejos en tiempo $O(N * LogN)$, siendo N la cantidad puntos\ciudades. Es importante notar que si bien existen en la literatura, e inclusive accesibles por Internet, muchos algoritmos para computar la triangulación de Delaunay o los diagramas de Voronoi, encontrar alguno que sea numéricamente estable y portable a plataforma PC es muy difícil. Los algoritmos para MST disponibles vía `ftp` son $O(N * LogN)$ en la cantidad de aristas, no así en los vértices como se requería en este trabajo. Por ello se desarrollaron dos algoritmos $O(N * LogN)$ en la cantidad de ciudades haciendo uso de la triangulación y los $kd-trees$. Asimismo se implementaron algoritmos eficientes para computar RNG y GG. No fue posible encontrar hasta la fecha algorimos eficientes y portables a PC para generar estos últimos dos grafos.

Al abordar el análisis empírico de heurísticas es muy importante seguir ciertas pautas. Cualquiera sea la contribución de un nuevo algoritmo, los experimentos computacionales serán necesarios a la hora de demostrar lo que los autores proponen. En general se pueden encontrar dos tipos de experimentos, algunos para comparar la eficiencia/eficacia de diferentes algoritmos bajo un mismo tipo de instancias de prueba, y otros para caracterizar y describir el comportamiento de cierto particular algoritmo en forma aislada. Es importante que una nueva heurística pueda:

- mostrar que existen soluciones factibles, de buena calidad y construibles en tiempos razonables, a partir de algún concepto que anteriormente había sido insuficientemente estudiado, o peor aún, ignorado.

- convertirse en una fuente de ideas a explorar y experimentar, fertilizando campos del conocimiento humano que bien hubiesen podido permancer aislados el uno del otro.

Es con este espíritu que se desarrollaron los algoritmos y experimentos que se expusieron en esta tesis.

El trabajo de Reinelt [37] muestra un estudio detallado de la utilización en tandem de heurísticas clásicas como $2-opt$, *nearest neighbor* y *Lin-Kernighan*, luego de obtener un tour factible a partir de un grafo "vecindario", GV. Es en el trabajo de Reinelt donde encontramos las raices de nuestras motivaciones. GV se definía como un **supergrafo** de la triangulación de Delaunay o del grafo de Delaunay. Sea S un conjunto de ciudades a visitar, y $TD(S)$ la triangulacion de Delaunay de dicho conjunto, entonces, GV_k tiene el mismo conjunto de vértices (ciudades) S y el conjunto de aristas se define por $E = \{(u,v)|u,v \in S, \exists C_{TD(S)}(u,v,k)\}$, es decir u estará conectado a v en GV_k si existe en TD(S) un camino C de longitud a lo sumo k desde u a v. Luego, se construye a partir de GV_k un tour factible que más tarde sera optimizado con alguna heurística tradicional de las mencionadas anteriormente.

A diferencia del estudio realizado por Reinelt, se investigó en el uso de algunos **subgrafos** de $TD(S)$ con propiedades geométricas conocidas. Si los resultados resultaban comparables a los de Reinelt, eso abriría un gran interrogante sobre la validez del uso de supergrafos de TD, dejando libre el camino a la in-

vestigación de estos subgrafos como fuente de información para las heurísticas clásicas.

De las tablas mostradas en el apéndice B se puede observar que todas las heurísticas derivadas de OCIG, se comparan muy bien con la técnica 2-opt, según la implementación de Reinelt. En dicha implementación los tours de inicialización son la salida del método NN, que como reconoce el autor

> "...are usually locally not bad and contain only a few severe global errors that can be corrected easily (and quickly)."

Es importante remarcar que nuestros tours no son 2-Optimos ni tampoco óptimos respecto de L-K, lo que nos brinda un amplio margen de mejora de nuestras heurística usando tan solo estos dos procedimientos en tandem (a la Reinelt). Recordemos que los segundos criterios de H1 y H2 rankeaban las ciudades de acuerdo a la suma de las aristas hacia sus vecinos en G, haciendolo H1 en forma decreciente y H2 en sentido contrario. Mientras que H3 y H4 daban prioridad a aquellas ciudades cuya distancia al vecino más cercano fuese menor y mayor respectivamente. Es entonces cuando cobra importancia notar que luego de un examen atento de las tablas y a pesar de esperarse que H1 fuese mucho mejor que H2 y H3 que H4, las brechas no son tan grandes. Esto deja abierta la conjetura de que probablemente la suma de las longitudes de las aristas en G y las distancias a los vecinos más cercanos, no sea un patrón "esencial" a ser usado como criterio de desempate. Probablemente otra característica de G podría ser más apta para nuestras heurísticas.

Los grafos que Reinelt define en [37] son "sandwich graphs" entre $TD = (V, E)$ y $(V, V * V)$, es decir, son más costosos de generar en tiempo y espacio que los nuestros ya que incluyen a TD como subgrafo y se construyen a partir de ella.

Si observamos las tablas de performance del apéncide C y consideramos los mínimos alcanzados por cada una de las cuatro versiones de OCIG a saber, OCIMST, OCIRNG, OCIGG y OCIDT, podemos ver que OCIRNG es el que devolvió los peores resultados. Sorprendentemente MST y TD (los extremos de la jerarquia de inclusión 4.2) fueron los grafos que brindaron mejores soluciones, siendo OCIDT más estable que OCIMST; este último tiene el mejor promedio. Así, podemos conjeturar que el MST mantiene información esencial de la topología del conjunto de ciudades susceptibles de ser utilizadas por los algoritmos de búsqueda y optimización. La triangulación de Delaunay parecería tener buena información pero más difícil de recuperar, mientras que el grafo de Gabriel y el de vecinos relativos aparentemente al estar en el medio de la jerarquia de inclusión $MST \subseteq RNG \subseteq GG \subseteq TD$, perderían la información que hace buena a la TD mientras que incluiría "ruido" a la escasa pero esencial información del MST.

De las mismas tablas advertimos que excepto en el caso del MST , los otros grafos son mejor explotados cuando se utilizan los criterios definidos para $H5, \ldots, H8$; MST devuelve sus mejores resultados bajo $H1, \ldots, H4$. Recordemos que los primeros cuatro criterios dán prioridad de inserción a aquellas ciudades con mayor número de vecinos en el grafo considerado, mientras que los últimos cuatro priorizan aquellas con menos vecinos. Creemos que OCIMST se desempeña mejor con $H1, \ldots, H4$ porque es el grafo en 4.2 con menor grado promedio en sus vértices. De esta manera cobra mayor relevancia como criterio

de desición el número de vecinos a un vértice respecto de otros aspecto de la estructura del grafo. Cuando OCI es instanciado con RNG, GG o TD, optimiza más al dar prioridad a aquellas ciudades con pocos vecinos, de esta manera estaría minimizanco las posibilidades que una ciudad quede "frustrada" [33]. Es interesante notar también que si bien las tres estrategias son de $O(N * Log\ N)$, las constantes en OCIDT son menores que en OCIGG, OCIRNG y OCIMST.

Como se dijo algunos párrafos líneas arriba y en la introducción de este trabajo, se pretendía demostrar la viabilidad de utilizar, en vez de supergrafos de TD, TD y sus subgrafos importantes como fuente de información para algoritmos clásicos del *ETSP* en el plano. Efectivamente fué esto lo que se encontró a partir de los experimentos del capítulo 6 y el apéndice B. Sorprendentemente, los resultados obtenidos con la TD y sus subgrafos eran equiparables a los obtenidos con supergrafos y luego una aplicación en tandem de heurísticas como el 2−opt o *nearest neighbor*, no así con *Lin-Kernighan*. Mas aún, pudimos verificar que, excepto en el caso de TD y MST el uso de GG y RNG no reporta beneficios con respecto a una inicialización random de OCI. Además y contra toda expectativa, los mejores resultados fueron obtenidos a partir de OCIMST. Esta última heurística se destacó a pesar de que el árbol de recubrimiento mínimo es el subgrafo de TD con menos links, por ende, con mas probabilidades de que las aristas constitutivas del tour óptimo no pertenezcan al mismo.

8.2 Palabras Finales

Al comparar nuestras heurísticas con aquellas de Reinelt en un conjunto reducido de instancias (ver tablas del apéndice C) observamos que nuestros resultados son mejores que su versión de *nearest neighbor* y comparables a su implementación de *2-Opt*. No logramos sin embargo alcanzar los valores obtenidos con *Lin-Kernighan*. Si extendemos el análisis a las instancias en TSPLIB, las 32 versiones de OCIG [1] superan a *nearest neighbor* y *2-Opt*, pero siguen sin alcanzar a *Lin-Kernighan*. Recordemos lo que dice Reinelt en la página 214 de [37]:

> "Since we ran both heuristics(2-Opt and Lin-Kernighan) until no more improvements could be found and because we have real-world problems CPU times are not stable. For practical purposes it might be a good idea to limit in advance the number of possible exchange steps to be considered. This leads to deterministic running times..."

Es decir, sus heurísticas no tienen tiempo de corrida determinístico mientras que las nuestras si.

Si se quiere aplicar el *TSP* en situaciones reales se deben tener en cuenta tres factores:

- La calidad esperada de las soluciones.
- Los recursos computacionales disponibles.
- El esfuerzo en la implementación.

[1] Ocho por cada uno de los cuatro posibles grafos en 4.2

Consideramos que la familia de heurísticas presentadas en este trabajo son aptas para instancias del *ETSP* de gran porte en el plano que deban ser resueltas bajo severas restricciones de recursos computacionales (tiempo y memoria). Sostenemos esto ya que OCIG en todas sus versiones logra combinar muy favorablemente los tres puntos mencionados arriba.

El camino recorrido ha sido largo y laborioso, aún así, solo se han explorado los inicios, de lo que consideramos, una fructífera línea de investigación en heurísticas rápidas para el *ETSP* en dos dimensiones. Tal vez, el principal mérito de un trabajo de investigación radique no en las respuestas que logra encontrar sino en las preguntas que deja planteadas. En lo que resta del trabajo mencionaremos las líneas de investigación que serán continuadas.

8.3 Notas:

Capítulo 9

Trabajos Futuros

Si de algún modo existo, si no soy una de tus repeticiones y erratas, existo como autor de "Los Enemigos". Para llevar a término ese drama, que puede justificarme y justificarte, requiero un año mas. Otórgame esos dias, tu de quien son los siglos y el tiempo.

El Milagro Secreto
J.L. Borges

Ya que no hay lugar para milagros en la ciencia y a que tampoco tengo un año mas, esta tesis termina aquí. Al futuro proyecto las siguientes ideas...

Las posibilidades de extensión de esta tesis son notablemente variadas. Por un lado se debería profundizar en la utilización de la jerarquia $MST \subseteq RNG \subseteq GG \subseteq TD$ bajo criterios diferentes a H1, H2, H3, H4, H5, H6, H7 y H8. Además, así como se hizo con OCIG, se evaluarán en el futuro próximo las heurísticas 2-OptG y Lin-KernighanG en sus 32 respectivas versiones.

Por otro lado, el uso que se les dió a los grafos MST, RNG, GG y MST es estático pues la información se extraía de los mismos solo una vez al comienzo de la construcción de un tour. Investigar en esquemas más dinámicos puede ser un campo rico en nuevas variantes. Por ejemplo la utilización de G como guía para las heurísticas $k-$opt, *nearest neighbor* y *Lin-Kernighan* en forma dinámica parece prometedora.

Desde el punto de vista de los algoritmos evolutivos para el *ETSP* , como ser, recocido simulado, búsqueda Tabú, algoritmos genéticos, redes neuronales [67], etc., podría ser beneficioso usar como vecindario de búsqueda a la TD o a alguno de sus subgrafos.

Asimismo, urge entender porque los grafos del extremo de la jerarquía, *MST* y DT, fueron los que reportaron mejores resultados. Sorprendente e importante a la vez, al haber sido el *MST* el grafo que mejor se comportó en los experimentos, nos sugiere que los métodos desarrollados en esta tésis serían susceptibles de ser generalizados a D dimensiones ($D > 2$) y cualquier métrica ya que el *MST* no hace uso de conceptos geométricos para su construcción, a diferen-

cia de TD, RNG y GG que dependen de ellos para su definición\construcción. Además cuando se pasa a D dimensiones algunas propiedades en la complejidad de dichos grafos varían sustancialmente. De todos modos, bajo las métricas Manhattan y Máxima la TD conserva las mismas propiedades que en la métrica Euclidea.

El mismo tipo de análisis que se hizo con OCIG debería llevarse a cabo con el algoritmo de aproximación de C. Papadimitriou y M. Yannakakis [82] sobre instancias del *TSP* con distancias 1 y 2. En el mencionado trabajo se presenta un algoritmo de aproximación cuya cota en el peor caso es 7/6. Sería interesante asignar a las aristas de G, ya sea MST,RNG, GG o TD, distancias 1 y a las del complemento 2. Luego se correría el algoritmo de aproximación para distancias 1 y 2 obteniéndose un tour factible. Finalmente se calcularía a partir de ese tour la verdadera longitud usando las distancias dadas por las coordenadas de los puntos en cuestión. Es lícito suponer que los grafos de la jerarquía 4.2 serán un buen vecindario de búsqueda para el algoritmo de Papadimitriou y Yannakakis. Si los resultados son favorables, se necesitaría entonces tratar de obtener cotas formales sobre dicho algoritmo.

Con un espíritu similar al anterior, creemos que se podría mejorar la cota del algoritmo de aproximación que utiliza el MST (descripto en el capítulo 2) cuya cota es 3/2, al considerar no solo al MST sino a su supergrafo, la TD.

Capítulo 10

Apendice A

Tablas con instancias de Reinelt y TSPLIB junto a sus valores óptimos

Instancias de comparacion con Heinelt	Optimos - Limite Inf.	Optimos - Limite Sup.
d198.tsp	15780	
pcb442.tsp	50778	
u574.tsp	36905	
p654.tsp	34643	
rat783.tsp	8806	
pcb1173.tsp	56892	
d1291.tsp	50606	50864,00
u1432.tsp	152920	
fl1577.tsp	22134	22249,00
d1655.tsp	62128	
d2103.tsp	79687	80259,00
pr2392.tsp	378032	
pcb3038.tsp	137617	137694,00

Instancias de comparacion sobre TSPLIB	
eil76.tsp	538
pr76.tsp	108159
rat99.tsp	1211
kroa100.tsp	21282
krob100.tsp	22141
kroc100.tsp	20749
krod100.tsp	21294
kroe100.tsp	22068
rd100.tsp	7910
eil101.tsp	629
lin105.tsp	14379
pr107.tsp	44303
pr124.tsp	59030
bier127.tsp	118282
pr136.tsp	96772
pr144.tsp	58537
kroa150.tsp	26524
krob150.tsp	26130
pr152.tsp	73682
u159.tsp	42080
rat195.tsp	2323
d198.tsp	15780
kroa200.tsp	29368
krob200.tsp	29437
ts225.tsp	126643
pr226.tsp	80369
gil262.tsp	2378
pr264.tsp	49135
pr299.tsp	48191
lin318.tsp	42029
fl417.tsp	11861
pr439.tsp	107217
d493.tsp	35002
att532.tsp	27686
u574.tsp	36905
rat575.tsp	6773
d657.tsp	48912
u724.tsp	41910
rat783.tsp	8806
nrw1379.tsp	56638
u1432.tsp	152970
d1655.tsp	62128

Capítulo 11

Apendice B

Tablas de resultados de las heurísticas desarrolladas

Instancias de comparacion con Reinelt	ocidth1	% sobre el óptimo	ocidth2	% sobre el óptimo
d198.tsp	16038	1,63	16132	2,23
pcb442.tsp	55057	8,43	55020	8,35
u574.tsp	39256	6,37	39513	7,07
p654.tsp	35782	3,29	35372	2,10
rat783.tsp	9436	7,15	9525	8,16
pcb1173.tsp	63107	10,92	62688	10,19
d1291.tsp	59455	17,49	55972	10,60
u1432.tsp	164029	7,26	165100	7,96
fl1577.tsp	25192	13,82	24811	12,09
d1655.tsp	69220	11,42	68418	10,12
d2103.tsp	94244	18,27	94268	18,30
pr2392.tsp	410619	8,62	414957	9,77
pcb3038.tsp	149907	8,93	148735	8,08
Promedio		9,51		8,85
Desviación estandar		4,90		4,10

Instancias de comparacion sobre TSPLIB				
eil76.tsp	566	5,20	568	5,58
pr76.tsp	113083	4,55	117502	8,64
rat99.tsp	1317	8,75	1263	4,29
kroa100.tsp	22103	3,86	22186	4,25
krob100.tsp	23462	5,97	23059	4,15
kroc100.tsp	21497	3,60	22074	6,39
krod100.tsp	22761	6,89	23076	8,37
kroe100.tsp	22427	1,63	23130	4,81
rd100.tsp	8275	4,61	8145	2,97
eil101.tsp	638	1,43	665	5,72
lin105.tsp	14931	3,84	15612	8,58
pr107.tsp	44804	1,13	45012	1,60
pr124.tsp	63659	7,84	62943	6,63
bier127.tsp	129129	9,17	122856	3,87
pr136.tsp	101501	4,89	99531	2,85
pr144.tsp	60179	2,81	61510	5,08
kroa150.tsp	27889	5,15	28200	6,32
krob150.tsp	28001	7,16	27650	5,82
pr152.tsp	75184	2,04	74928	1,69
u159.tsp	43292	2,88	44987	6,91
rat195.tsp	2436	4,86	2515	8,27
d198.tsp	16038	1,63	16132	2,23
kroa200.tsp	30173	2,74	30362	3,38
krob200.tsp	30930	5,07	31178	5,91
ts225.tsp	132162	4,36	133601	5,49
pr226.tsp	83154	3,47	82930	3,19
gil262.tsp	2521	6,01	2497	5,00
pr264.tsp	52528	6,91	50753	3,29
pr299.tsp	50269	4,31	50510	4,81
lin318.tsp	44057	4,83	44050	4,81
fl417.tsp	12265	3,41	12024	1,37
pr439.tsp	115457	7,69	115107	7,36
d493.tsp	36551	4,43	37207	6,30
att532.tsp	29264	5,70	28995	4,73
u574.tsp	39256	6,37	39513	7,07
rat575.tsp	7231	6,76	7239	6,88
d657.tsp	52098	6,51	52354	7,04
u724.tsp	44296	5,69	44437	6,03
rat783.tsp	9436	7,15	9525	8,16
nrw1379.tsp	60536	6,88	60247	6,37
u1432.tsp	164029	7,23	165100	7,93
d1655.tsp	69220	11,42	68418	10,12
Promedio		**5,16**		**5,48**
Desviación estandar		2,25		2,11

Instancias de comparacion con Reinelt	ocidth3	% sobre el óptimo	ocidth4	% sobre el óptimo
d198.tsp	16002	1,41	16125	2,19
pcb442.tsp	54932	8,18	55564	9,43
u574.tsp	39066	5,86	39633	7,39
p654.tsp	35360	2,07	35194	1,59
rat783.tsp	9496	7,84	9406	6,81
pcb1173.tsp	62646	10,11	62421	9,72
d1291.tsp	56290	11,23	57117	12,87
u1432.tsp	164727	7,72	166101	8,62
fl1577.tsp	25047	13,16	25067	13,25
d1655.tsp	68594	10,41	69642	12,09
d2103.tsp	90135	13,11	91823	15,23
pr2392.tsp	409137	8,23	412175	9,03
pcb3038.tsp	149672	8,76	149759	8,82
Promedio		**8,31**		**9,00**
Desviación estandar		3,61		3,99

Instancias de comparacion sobre TSPLIB				
eil76.tsp	560	4,09	566	5,20
pr76.tsp	118788	9,83	114111	5,50
rat99.tsp	1305	7,76	1281	5,78
kroa100.tsp	21931	3,05	22194	4,29
krob100.tsp	23088	4,28	23259	5,05
kroc100.tsp	21708	4,62	21135	1,86
krod100.tsp	22751	6,84	22324	4,84
kroe100.tsp	23101	4,68	23093	4,64
rd100.tsp	8311	5,07	8241	4,18
eil101.tsp	663	5,41	658	4,61
lin105.tsp	15774	9,70	15664	8,94
pr107.tsp	44749	1,01	44749	1,01
pr124.tsp	60667	2,77	63121	6,93
bier127.tsp	123113	4,08	126207	6,70
pr136.tsp	100972	4,34	100605	3,96
pr144.tsp	60315	3,04	60996	4,20
kroa150.tsp	27200	2,55	28038	5,71
krob150.tsp	27680	5,93	26928	3,05
pr152.tsp	74274	0,80	76510	3,84
u159.tsp	42813	1,74	46896	11,44
rat195.tsp	2460	5,90	2511	8,09
d198.tsp	16002	1,41	16125	2,19
kroa200.tsp	30197	2,82	30363	3,39
krob200.tsp	30787	4,59	30356	3,12
ts225.tsp	133601	5,49	133601	5,49
pr226.tsp	82990	3,26	83035	3,32
gil262.tsp	2551	7,28	2515	5,76
pr264.tsp	51564	4,94	51142	4,08
pr299.tsp	51219	6,28	49499	2,71
lin318.tsp	43987	4,66	44256	5,30
fl417.tsp	12213	2,97	12029	1,42
pr439.tsp	116028	8,22	114315	6,62
d493.tsp	36895	5,41	37406	6,87
att532.tsp	28985	4,69	29168	5,35
u574.tsp	39066	5,86	39633	7,39
rat575.tsp	7249	7,03	7210	6,45
d657.tsp	52467	7,27	51645	5,59
u724.tsp	44885	7,10	44690	6,63
rat783.tsp	9496	7,84	9406	6,81
nrw1379.tsp	60484	6,79	59996	5,93
u1432.tsp	164727	7,69	166101	8,58
d1655.tsp	68594	10,41	69642	12,09
Promedio		**5,23**		**5,36**
Desviación estandar		2,35		2,34

Instancias de comparacion con Reinelt	ocidth5	% sobre el óptimo	ocidth6	% sobre el óptimo
[illegible]98.tsp	16357	3,66	16462	4,32
[illegible]b442.tsp	53590	5,54	54680	7,68
u574.tsp	39341	6,60	39170	6,14
[illegible]54.tsp	36287	4,75	36805	6,24
[illegible]783.tsp	9497	7,85	9462	7,45
pcb1173.tsp	61513	8,12	61725	8,50
[illegible]291.tsp	56227	11,11	55987	10,63
[illegible]1432.tsp	165218	8,04	163337	6,81
fl1577.tsp	24321	9,88	24422	10,34
[illegible]655.tsp	67959	9,39	68696	10,57
d2103.tsp	92629	16,24	93317	17,10
pr2392.tsp	408622	8,09	406252	7,46
[illegible]b3038.tsp	148964	8,25	148540	7,94
[illegible]omedio		**8,27**		**8,55**
[illegible]esviación estandar		3,16		3,17

Instancias de comparacion sobre TSPLIB				
[illegible]il76.tsp	556	3,35	567	5,39
pr76.tsp	112342	3,87	112143	3,68
[illegible]t99.tsp	1293	6,77	1293	6,77
krpa100.tsp	23458	10,22	21391	0,51
krob100.tsp	22576	1,96	22614	2,14
[illegible]pc100.tsp	21675	4,46	21328	2,79
krod100.tsp	21826	2,50	22106	3,81
[illegible]rpe100.tsp	22745	3,07	22509	2,00
[illegible]u100.tsp	8240	4,17	8256	4,37
eil101.tsp	655	4,13	653	3,82
[illegible]105.tsp	15142	5,31	15628	8,69
[illegible]x107.tsp	44577	0,62	44577	0,62
pr124.tsp	60091	1,80	59458	0,73
[illegible]er127.tsp	125663	6,24	122423	3,50
pr136.tsp	102257	5,67	99262	2,57
pr144.tsp	59576	1,77	60422	3,22
[illegible]pa150.tsp	27355	3,13	27971	5,46
krob150.tsp	26481	1,34	26832	2,69
[illegible]u152.tsp	75018	1,81	75359	2,28
[illegible]159.tsp	44067	4,72	44129	4,87
rat195.tsp	2485	6,97	2534	9,08
[illegible]198.tsp	16357	3,66	16462	4,32
krpa200.tsp	30706	4,56	31379	6,85
krob200.tsp	30771	4,53	30666	4,18
[illegible]225.tsp	138449	9,32	138532	9,39
pr226.tsp	81724	1,69	81704	1,66
gil262.tsp	2547	7,11	2516	5,80
[illegible]264.tsp	52083	6,00	52763	7,38
pr299.tsp	50672	5,15	51209	6,26
lin318.tsp	44665	6,27	43625	3,80
[illegible]417.tsp	12281	3,54	12468	5,12
pr439.tsp	112453	4,88	113490	5,85
[illegible]493.tsp	37129	6,08	36641	4,68
[illegible]u532.tsp	29097	5,10	29386	6,14
u574.tsp	39341	6,60	39170	6,14
[illegible]u575.tsp	7272	7,37	7237	6,85
[illegible]657.tsp	51303	4,89	51581	5,46
[illegible]u724.tsp	44626	6,48	44573	6,35
[illegible]783.tsp	9497	7,85	9462	7,45
[illegible]w1379.tsp	59666	5,35	60450	6,73
u1432.tsp	165218	8,01	163337	6,78
[illegible]655.tsp	67959	9,39	68696	10,57
[illegible]omedio		**4,95**		**4,92**
[illegible]esviación estandar		2,30		2,41

Instancias de comparacion con Reinelt	ocidth7	% sobre el óptimo	ocidth8	% sobre el óptimo
d198.tsp	16361	3,68	16285	3,20
fnb442.tsp	54228	6,79	54194	6,73
u574.tsp	39231	6,30	39167	6,13
u724.tsp	36790	6,20	36626	5,72
rat783.tsp	9509	7,98	9551	8,46
pcb1173.tsp	61887	8,78	61975	8,93
d1291.tsp	55081	8,84	56499	11,64
u1432.tsp	165964	8,53	165934	8,51
fl1577.tsp	24999	12,94	24898	12,49
d1655.tsp	68681	10,55	68307	9,95
d2103.tsp	88531	11,10	92298	15,83
pr2392.tsp	407680	7,84	405143	7,17
fnb3038.tsp	149700	8,78	148996	8,27
promedio		**8,33**		**8,69**
desviación estandar		2,37		3,26

Instancias de comparacion sobre TSPLIB				
eil76.tsp	564	4,83	565	5,02
pr76.tsp	112581	4,09	111473	3,06
kr99.tsp	1303	7,60	1291	6,61
krpa100.tsp	21513	1,09	21879	2,81
krob100.tsp	22704	2,54	22511	1,67
krpc100.tsp	21914	5,61	21204	2,19
krod100.tsp	21880	2,75	21812	2,43
krbe100.tsp	22565	2,25	23243	5,32
eil100.tsp	8419	6,43	8670	9,61
eil101.tsp	647	2,86	648	3,02
lin105.tsp	15579	8,35	14867	3,39
pr107.tsp	44577	0,62	44577	0,62
pr124.tsp	59767	1,25	61433	4,07
ber127.tsp	124565	5,31	124390	5,16
pr136.tsp	101439	4,82	102208	5,62
pr144.tsp	60422	3,22	59181	1,10
krpa150.tsp	27212	2,59	27724	4,52
krob150.tsp	27438	5,01	26650	1,99
pr152.tsp	75355	2,27	76303	3,56
u159.tsp	44086	4,77	43930	4,40
rat195.tsp	2474	6,50	2477	6,63
d198.tsp	16361	3,68	16285	3,20
krpa200.tsp	30704	4,55	31020	5,63
krob200.tsp	30553	3,79	31477	6,93
ts225.tsp	137894	8,88	137894	8,88
pr226.tsp	81655	1,60	82731	2,94
gil262.tsp	2486	4,54	2510	5,55
a280.tsp	52482	6,81	52637	7,13
pr299.tsp	52284	8,49	50216	4,20
lin318.tsp	43821	4,26	43713	4,01
fl417.tsp	12281	3,54	12250	3,28
pr439.tsp	116078	8,26	113712	6,06
pcb493.tsp	36737	4,96	36563	4,46
att532.tsp	29610	6,95	29335	5,96
u574.tsp	39231	6,30	39167	6,13
rat575.tsp	7219	6,58	7226	6,69
p657.tsp	51240	4,76	51920	6,15
u724.tsp	44836	6,98	44428	6,01
rat783.tsp	9509	7,98	9551	8,46
vm1379.tsp	60205	6,30	60251	6,38
u1432.tsp	165964	8,49	165934	8,47
d1655.tsp	68681	10,55	68307	9,95
promedio		**5,07**		**4,98**
desviación estandar		2,40		2,26

Instancias de comparacion con ReInelt	oclrngh1	% sobre el óptimo	oclrngh2	% sobre el óptimo
d198.tsp	16313	3,38	16332	3,50
pcb442.tsp	54567	7,46	55271	8,85
u574.tsp	39639	7,41	39049	5,81
p654.tsp	36650	5,79	35916	3,67
rat783.tsp	9518	8,09	9490	7,77
pcb1173.tsp	62850	10,47	62224	9,37
d1291.tsp	59846	18,26	59614	17,80
u1432.tsp	167151	9,31	164765	7,75
fl1577.tsp	23981	8,34	24655	11,39
d1655.tsp	68929	10,95	69018	11,09
d2103.tsp	93616	17,48	91491	14,81
pr2392.tsp	418729	10,77	415760	9,98
pcb3038.tsp	151279	9,93	150826	9,60
Promedio		9,82		9,34
Desviación estandar		4,15		4,01

Instancias de comparacion sobre TSPLIB				
eil76.tsp	564	4,83	566	5,20
pr76.tsp	114788	6,13	115376	6,67
rat99.tsp	1293	6,77	1269	4,79
kroa100.tsp	21404	0,57	21363	0,38
krob100.tsp	23398	5,68	22888	3,37
kroc100.tsp	21064	1,52	21456	3,41
krod100.tsp	22572	6,00	22592	6,10
kroe100.tsp	23293	5,55	22915	3,84
rd100.tsp	8660	9,48	8198	3,64
eil101.tsp	665	5,72	652	3,66
lin105.tsp	15015	4,42	15556	8,19
pr107.tsp	44431	0,29	44402	0,22
pr124.tsp	60603	2,66	60335	2,21
bier127.tsp	128136	8,33	131252	10,97
pr136.tsp	104424	7,91	98360	1,64
pr144.tsp	61278	4,68	60583	3,50
kroa150.tsp	27760	4,66	27639	4,20
krob150.tsp	26693	2,15	27681	5,94
pr152.tsp	75036	1,84	77931	5,77
u159.tsp	43792	4,07	45154	7,31
rat195.tsp	2549	9,73	2502	7,71
d198.tsp	16313	3,38	16332	3,50
kroa200.tsp	31001	5,56	31071	5,80
krob200.tsp	31294	6,31	31045	5,46
ts225.tsp	141357	11,62	141357	11,62
pr226.tsp	81677	1,63	82176	2,25
gil262.tsp	2529	6,35	2539	6,77
pr264.tsp	53420	8,72	51183	4,17
pr299.tsp	52479	8,90	50408	4,60
lin318.tsp	44538	5,97	45430	8,09
fl417.tsp	12479	5,21	12292	3,63
pr439.tsp	114878	7,15	118714	10,72
d493.tsp	37115	6,04	36776	5,07
att532.tsp	29118	5,17	29461	6,41
u574.tsp	39639	7,41	39049	5,81
rat575.tsp	7249	7,03	7199	6,29
d657.tsp	52380	7,09	52146	6,61
u724.tsp	45286	8,06	44742	6,76
rat783.tsp	9518	8,09	9490	7,77
nrw1379.tsp	60436	6,71	60382	6,61
u1432.tsp	167151	9,27	164765	7,71
d1655.tsp	68929	10,95	69018	11,09
Promedio		5,94		5,61
Desviación estandar		2,69		2,67

Instancias de comparacion con Reinelt	ocirngh3	% sobre el óptimo	ocirngh4	% sobre el óptimo
d198.tsp	16293	3,25	16241	2,92
pcb442.tsp	55233	8,77	54757	7,84
u574.tsp	39181	6,17	39329	6,57
p654.tsp	36583	5,60	36739	6,05
rat783.tsp	9505	7,94	9610	9,13
pcb1173.tsp	63589	11,77	62745	10,29
d1291.tsp	58400	15,40	58192	14,99
u1432.tsp	165652	8,33	166157	8,66
fl1577.tsp	24168	9,19	24044	8,63
d1655.tsp	68344	10,01	70029	12,72
d2103.tsp	88068	10,52	93018	16,73
pr2392.tsp	418922	10,82	411715	8,91
pcb3038.tsp	149465	8,61	150755	9,55
Promedio		8,95		9,46
Desviación estandar		3,02		3,67

Instancias de comparacion sobre TSPLIB				
eil76.tsp	567	5,39	569	5,76
pr76.tsp	116047	7,29	113240	4,70
rat99.tsp	1263	4,29	1290	6,52
kroa100.tsp	21363	0,38	21906	2,93
krob100.tsp	23771	7,36	23492	6,10
kroc100.tsp	22318	7,56	20915	0,80
krod100.tsp	22410	5,24	22020	3,41
kroe100.tsp	23272	5,46	23293	5,55
rd100.tsp	8165	3,22	8503	7,50
eil101.tsp	661	5,09	660	4,93
lin105.tsp	15500	7,80	15372	6,91
pr107.tsp	44621	0,72	44442	0,31
pr124.tsp	60177	1,94	60146	1,89
bier127.tsp	132691	12,18	124280	5,07
pr136.tsp	100723	4,08	100541	3,89
pr144.tsp	60269	2,96	62201	6,26
kroa150.tsp	27851	5,00	27761	4,66
krob150.tsp	27167	3,97	27526	5,34
pr152.tsp	76999	4,50	76472	3,79
u159.tsp	44862	6,61	44765	6,38
rat195.tsp	2521	8,52	2530	8,91
d198.tsp	16293	3,25	16241	2,92
kroa200.tsp	31260	6,44	30601	4,20
krob200.tsp	31179	5,92	30901	4,97
ts225.tsp	141357	11,62	141357	11,62
pr226.tsp	82454	2,59	84169	4,73
gil262.tsp	2517	5,85	2519	5,93
pr264.tsp	51850	5,53	54135	10,18
pr299.tsp	53743	11,52	53428	10,87
lin318.tsp	44839	6,69	44359	5,54
fl417.tsp	12500	5,39	12321	3,88
pr439.tsp	117257	9,36	115090	7,34
d493.tsp	36946	5,55	37406	6,87
att532.tsp	29721	7,35	29062	4,97
u574.tsp	39181	6,17	39329	6,57
rat575.tsp	7305	7,85	7320	8,08
d657.tsp	52275	6,88	53099	8,56
u724.tsp	44709	6,68	45308	8,11
rat783.tsp	9505	7,94	9610	9,13
nrw1379.tsp	60665	7,11	60016	5,96
u1432.tsp	165652	8,29	166157	8,62
d1655.tsp	68344	10,01	70029	12,72
Promedio		6,13		6,03
Desviación estandar		2,66		2,68

Instancias de comparacion con Reinelt	oclrngh5	% sobre el óptimo	oclrngh6	% sobre el óptimo
d198.tsp	16327	3,47	16139	2,28
fnb442.tsp	54530	7,39	55381	9,06
u574.tsp	39059	5,84	40090	8,63
p654.tsp	35028	1,11	35021	1,09
rat783.tsp	9363	6,33	9403	6,78
pcb1173.tsp	62925	10,60	63254	11,18
d1291.tsp	58547	15,69	57044	12,72
fl1432.tsp	164755	7,74	166526	8,90
fl1577.tsp	25838	16,73	23637	6,79
d1655.tsp	69223	11,42	68133	9,67
d2103.tsp	91438	14,75	91642	15,00
pr2392.tsp	411745	8,92	415237	9,84
pcb3038.tsp	151173	9,85	150376	9,27
promedio		9,22		8,55
desviación estandar		4,66		3,78

Instancias de comparacion sobre TSPLIB				
eil76.tsp	570	5,95	549,00	2,04
pr76.tsp	108234	0,07	115108,00	6,42
gr99.tsp	1261	4,13	1242,00	2,56
kroa100.tsp	21793	2,40	21343,00	0,29
krob100.tsp	23338	5,41	23413,00	5,74
kroc100.tsp	21537	3,80	21612,00	4,16
krod100.tsp	22378	5,09	21964,00	3,15
kroe100.tsp	23412	6,09	22535,00	2,12
rd100.tsp	8254	4,35	8301,00	4,94
eil101.tsp	663	5,41	653,00	3,82
lin105.tsp	15117	5,13	14568,00	1,31
pr107.tsp	44851	1,24	44533,00	0,52
pr124.tsp	59599	0,96	62298,00	5,54
bier127.tsp	122813	3,83	125579,00	6,17
pr136.tsp	103702	7,16	105270,00	8,78
pr144.tsp	60278	2,97	59970,00	2,45
kroa150.tsp	27442	3,46	27513,00	3,73
krob150.tsp	26490	1,38	26655	2,01
pr152.tsp	75600	2,60	75760	2,82
u159.tsp	44499	5,75	44560,00	5,89
rat195.tsp	2528	8,82	2474,00	6,50
d198.tsp	16327	3,47	16139,00	2,28
kroa200.tsp	30534	3,97	31285,00	6,53
krob200.tsp	30988	5,27	30149,00	2,42
ts225.tsp	138565	9,41	138565,00	9,41
pr226.tsp	82246	2,34	82036,00	2,07
gil262.tsp	2480	4,29	2505,00	5,34
a280.tsp	53404	8,69	53533,00	8,95
pr299.tsp	52932	9,84	51781,00	7,45
lin318.tsp	44094	4,91	44563,00	6,03
fl417.tsp	12845	8,30	12308,00	3,77
pr439.tsp	116137	8,32	112763,00	5,17
pcb493.tsp	36647	4,70	37169,00	6,19
d493.tsp	28841	4,17	29151,00	5,29
u574.tsp	39059	5,84	40090,00	8,63
rat575.tsp	7209	6,44	7284,00	7,54
p654.tsp	52508	7,35	52136	6,59
u724.tsp	44976	7,32	44523,00	6,23
rat783.tsp	9363	6,33	9403,00	6,78
vm1379.tsp	60390	6,62	60717,00	7,20
fl1432.tsp	164755	7,70	166526,00	8,86
d1655.tsp	69223	11,42	68133,00	9,67
promedio		5,30		5,08
desviación estandar		2,55		2,55

Instancias de comparacion con Reinelt	ocirngh7	% sobre el óptimo	ocirngh8	% sobre el óptimo
98.tsp	16482	4,45	16570	5,01
b442.tsp	54726	7,78	55347	9,00
u574.tsp	40469	9,66	39367	6,67
54.tsp	35733	3,15	35137	1,43
t783.tsp	9379	6,51	9404	6,79
pcb1173.tsp	62357	9,61	62544	9,93
291.tsp	56119	10,89	57139	12,91
u1432.tsp	165417	8,17	165707	8,36
1577.tsp	24040	8,61	26021	17,56
655.tsp	68602	10,42	68049	9,53
d2103.tsp	90096	13,06	93805	17,72
2392.tsp	420119	11,13	411793	8,93
b3038.tsp	149720	8,79	150266	9,19
Promedio		8,63		9,46
Desviación estandar		2,73		4,54

Instancias de comparacion sobre TSPLIB				
eil76.tsp	575,00	6,88	560,00	4,09
76.tsp	113189,00	4,65	110423,00	2,09
99.tsp	1241,00	2,48	1246,00	2,89
kroa100.tsp	21343,00	0,29	21578,00	1,39
ob100.tsp	23519,00	6,22	22998,00	3,87
oc100.tsp	22066,00	6,35	21511,00	3,67
krod100.tsp	21501,00	0,97	22348,00	4,95
be100.tsp	22676,00	2,76	23004,00	4,24
100.tsp	8128,00	2,76	8183,00	3,45
eil101.tsp	651,00	3,50	657,00	4,45
105.tsp	14573,00	1,35	14943,00	3,92
pr107.tsp	45162,00	1,94	45348,00	2,36
pr124.tsp	59385,00	0,60	61990,00	5,01
er127.tsp	122601,00	3,65	123267,00	4,21
pr136.tsp	103370,00	6,82	102585,00	6,01
144.tsp	64383,00	9,99	62886,00	7,43
oa150.tsp	27387,00	3,25	28067,00	5,82
krob150.tsp	26509	1,45	26785	2,51
152.tsp	75703	2,74	74978	1,76
159.tsp	45571,00	8,30	43912,00	4,35
rat195.tsp	2499,00	7,58	2494,00	7,36
98.tsp	16482,00	4,45	16570,00	5,01
kroa200.tsp	30770,00	4,77	30692,00	4,51
krob200.tsp	30882,00	4,91	31381,00	6,60
225.tsp	138565,00	9,41	138565,00	9,41
pr226.tsp	81203,00	1,04	81629,00	1,57
262.tsp	2547,00	7,11	2475,00	4,08
264.tsp	53164,00	8,20	54398,00	10,71
pr299.tsp	51487,00	6,84	51171,00	6,18
318.tsp	43893,00	4,44	44650,00	6,24
417.tsp	12604,00	6,26	12387,00	4,43
pr439.tsp	113525,00	5,88	113826,00	6,16
493.tsp	36827,00	5,21	36582,00	4,51
532.tsp	29292,00	5,80	29166,00	5,35
u574.tsp	40469,00	9,66	39367,00	6,67
575.tsp	7237,00	6,85	7198,00	6,27
657.tsp	52393	7,12	52407	7,15
u724.tsp	45029,00	7,44	44758,00	6,80
783.tsp	9379,00	6,51	9404,00	6,79
rw1379.tsp	60471,00	6,77	59894,00	5,75
u1432.tsp	165417,00	8,14	165707,00	8,33
655.tsp	68602,00	10,42	68049,00	9,53
Promedio		5,28		5,19
Desviación estandar		2,74		2,16

Instancias de comparacion con Reinelt	ociggh1	% sobre el óptimo	ociggh2	% sobre el óptimo
d198.tsp	16367	3,72	16513	4,65
pcb442.tsp	55503	9,31	56001	10,29
u574.tsp	39403	6,77	39575	7,23
p654.tsp	35613	2,80	36495	5,35
rat783.tsp	9583	8,82	9619	9,23
pcb1173.tsp	63636	11,85	63288	11,24
d1291.tsp	57462	13,55	57844	14,30
u1432.tsp	164614	7,65	164713	7,71
fl1577.tsp	24416	10,31	25647	15,87
d1655.tsp	69770	12,30	68864	10,84
d2103.tsp	94510	18,60	92820	16,48
pr2392.tsp	422006	11,63	422516	11,77
pcb3038.tsp	151328	9,96	151192	9,86
Promedio		9,79		**10,37**
Desviación estandar		4,15		3,68

Instancias de comparacion sobre TSPLIB				
eil76.tsp	580	7,81	582	8,18
pr76.tsp	112512	4,02	115057	6,38
rat99.tsp	1326	9,50	1290	6,52
kroa100.tsp	22952	7,85	22004	3,39
krob100.tsp	23050	4,11	23416	5,76
kroc100.tsp	22663	9,22	21833	5,22
krod100.tsp	22629	6,27	22597	6,12
kroe100.tsp	23500	6,49	22941	3,96
rd100.tsp	8190	3,54	8851	11,90
eil101.tsp	682	8,43	660	4,93
lin105.tsp	16015	11,38	16001	11,28
pr107.tsp	44566	0,59	44829	1,19
pr124.tsp	61950	4,95	65160	10,38
bier127.tsp	121003	2,30	129124	9,17
pr136.tsp	100268	3,61	102048	5,45
pr144.tsp	62226	6,30	59713	2,01
kroa150.tsp	28040	5,72	28026	5,66
krob150.tsp	27002	3,34	26870	2,83
pr152.tsp	74274	0,80	76325	3,59
u159.tsp	46895	11,44	44878	6,65
rat195.tsp	2522	8,57	2553	9,90
d198.tsp	16367	3,72	16513	4,65
kroa200.tsp	30929	5,32	30885	5,17
krob200.tsp	30564	3,83	31047	5,47
ts225.tsp	143637	13,42	136303	7,63
pr226.tsp	85877	6,85	81781	1,76
gil262.tsp	2530	6,39	2563	7,78
pr264.tsp	51013	3,82	50958	3,71
pr299.tsp	52723	9,40	53567	11,16
lin318.tsp	45610	8,52	45143	7,41
fl417.tsp	11915	0,46	12486	5,27
pr439.tsp	116836	8,97	117477	9,57
d493.tsp	37812	8,03	36967	5,61
att532.tsp	29514	6,60	29496	6,54
u574.tsp	39403	6,77	39575	7,23
rat575.tsp	7290	7,63	7310	7,93
d657.tsp	52674	7,69	53551	9,48
u724.tsp	45925	9,58	44736	6,74
rat783.tsp	9583	8,82	9619	9,23
nrw1379.tsp	60760	7,28	61005	7,71
u1432.tsp	164614	7,61	164713	7,68
d1655.tsp	69770	12,30	68864	10,84
Promedio		6,65		**6,64**
Desviación estandar		3,06		2,68

Instancias de comparacion con Reinelt	ociggh3	% sobre el óptimo	ociggh4	% sobre el óptimo
d198.tsp	16361	3,68	16581	5,08
pcb442.tsp	56101	10,48	55538	9,37
u574.tsp	39409	6,78	39227	6,29
p654.tsp	36489	5,33	35978	3,85
rat783.tsp	9430	7,09	9588	8,88
pcb1173.tsp	63930	12,37	64030	12,55
d1291.tsp	56854	12,35	57641	13,90
u1432.tsp	163661	7,02	166142	8,65
fl1577.tsp	25449	14,98	24138	9,05
d1655.tsp	69252	11,47	69831	12,40
d2103.tsp	88964	11,64	94259	18,29
pr2392.tsp	421815	11,58	422372	11,73
pcb3038.tsp	150890	9,64	150920	9,67
Promedio		9,57		9,98
Desviación estandar		3,30		3,87

Instancias de comparacion sobre TSPLIB				
eil76.tsp	574	6,69	583	8,36
pr76.tsp	115376	6,67	112817	4,31
rat99.tsp	1292	6,69	1259	3,96
kroa100.tsp	21939	3,09	22597	6,18
krob100.tsp	22704	2,54	23341	5,42
kroc100.tsp	21850	5,31	21131	1,84
krod100.tsp	22361	5,01	22813	7,13
kroe100.tsp	23793	7,82	23468	6,34
rd100.tsp	8647	9,32	8453	6,86
eil101.tsp	671	6,68	655	4,13
lin105.tsp	15787	9,79	15892	10,52
pr107.tsp	44772	1,06	44577	0,62
pr124.tsp	61734	4,58	60598	2,66
bier127.tsp	129668	9,63	124417	5,19
pr136.tsp	102480	5,90	101447	4,83
pr144.tsp	61786	5,55	63180	7,93
kroa150.tsp	27663	4,29	28366	6,94
krob150.tsp	27913	6,82	26827	2,67
pr152.tsp	76165	3,37	76229	3,46
u159.tsp	44615	6,02	46832	11,29
rat195.tsp	2536	9,17	2497	7,49
d198.tsp	16361	3,68	16581	5,08
kroa200.tsp	30605	4,21	31403	6,93
krob200.tsp	31729	7,79	30640	4,09
ts225.tsp	141067	11,39	141067	11,39
pr226.tsp	81546	1,46	81336	1,20
gil262.tsp	2580	8,49	2542	6,90
pr264.tsp	50565	2,91	50856	3,50
pr299.tsp	52676	9,31	52257	8,44
lin318.tsp	46011	9,47	45472	8,19
fl417.tsp	12498	5,37	11933	0,61
pr439.tsp	117452	9,55	116891	9,02
d493.tsp	37242	6,40	37211	6,31
att532.tsp	29215	5,52	29480	6,48
u574.tsp	39409	6,78	39227	6,29
rat575.tsp	7303	7,83	7203	6,35
d657.tsp	53463	9,30	52569	7,48
u724.tsp	45358	8,23	45657	8,94
rat783.tsp	9430	7,09	9588	8,88
nrw1379.tsp	60925	7,57	61225	8,10
u1432.tsp	163661	6,99	166142	8,61
d1655.tsp	69252	11,47	69831	12,40
Promedio		6,59		6,27
Desviación estandar		2,55		2,85

Instancias de comparacion con Reinelt	ocigah5	% sobre el óptimo	ocigah6	% sobre el óptimo
d198.tsp	16077	1,88	16191	2,60
fnb442.tsp	53853	6,06	53630	5,62
lu574.tsp	39131	6,03	39322	6,55
p654.tsp	35200	1,61	36601	5,65
rat783.tsp	9446	7,27	9475	7,60
pcb1173.tsp	61987	8,96	62487	9,83
d1291.tsp	58709	16,01	57888	14,39
fl1432.tsp	163943	7,21	163987	7,24
fl1577.tsp	25664	15,95	24415	10,31
d1655.tsp	68282	9,91	68612	10,44
vm2103.tsp	93501	17,34	92181	15,68
pr2392.tsp	409900	8,43	405601	7,29
pcb3038.tsp	149892	8,92	150642	9,46
Promedio		**8,89**		**8,67**
Desviación estandar		4,97		3,59

Instancias de comparacion sobre TSPLIB				
eil76.tsp	557	3,53	569	5,76
pr76.tsp	109531	1,27	112789	4,28
gt99.tsp	1258	3,88	1264	4,38
krpa100.tsp	21292	0,05	22268	4,63
krob100.tsp	22604	2,09	23291	5,19
kroc100.tsp	21405	3,16	21030	1,35
krod100.tsp	22030	3,46	21500	0,97
kroe100.tsp	22544	2,16	22411	1,55
rd100.tsp	8198	3,64	8742	10,52
eil101.tsp	660	4,93	658	4,61
lin105.tsp	15303	6,43	14913	3,71
pr107.tsp	44533	0,52	44751	1,01
pr124.tsp	61818	4,72	61144	3,58
bier127.tsp	125901	6,44	124095	4,91
ch136.tsp	103108	6,55	100572	3,93
pr144.tsp	60492	3,34	62184	6,23
kroa150.tsp	28069	5,82	28252	6,51
krob150.tsp	27219	4,17	26838	2,71
pr152.tsp	77034	4,55	75908	3,02
u159.tsp	47646	13,23	45244	7,52
rat195.tsp	2448	5,38	2446	5,29
d198.tsp	16077	1,88	16191	2,60
kroa200.tsp	31042	5,70	31096	5,88
krob200.tsp	30772	4,54	30542	3,75
ts225.tsp	129046	1,90	137634	8,68
pr226.tsp	81746	1,71	81679	1,63
gil262.tsp	2550	7,23	2537	6,69
pr264.tsp	53826	9,55	53023	7,91
a280.tsp	49680	3,09	50183	4,13
lin318.tsp	45220	7,59	44486	5,85
fl417.tsp	12216	2,99	12006	1,22
pr439.tsp	116431	8,59	113283	5,66
pcb442.tsp	36541	4,40	36820	5,19
d493.tsp	29425	6,28	29562	6,78
lu574.tsp	39131	6,03	39322	6,55
rat575.tsp	7232	6,78	7325	8,15
p657.tsp	53164	8,69	52789	7,93
lu724.tsp	44678	6,60	45138	7,70
rat783.tsp	9446	7,27	9475	7,60
vm1379.tsp	60578	6,96	60158	6,21
fl1432.tsp	163943	7,17	163987	7,20
d1655.tsp	68282	9,91	68612	10,44
Promedio		5,10		**5,22**
Desviación estandar		2,74		2,45

Instancias de comparacion con Reinelt	ociggh7	% sobre el óptimo	ociggh8	% sobre el óptimo
d198.tsp	16162	2,42	16047	1,69
eb442.tsp	53670	5,70	55839	9,97
r574.tsp	39654	7,45	39168	6,13
p654.tsp	35923	3,69	38200	10,27
783.tsp	9414	6,90	9424	7,02
pcb1173.tsp	62181	9,30	61817	8,66
1291.tsp	57683	13,98	58049	14,71
432.tsp	163571	6,97	164126	7,33
fl1577.tsp	24430	10,37	24532	10,83
1655.tsp	68507	10,27	68343	10,00
2103.tsp	86748	8,86	90978	14,17
pr2392.tsp	411779	8,93	406189	7,45
eb3038.tsp	149910	8,93	150431	9,31
Promedio		7,98		9,04
esviación estandar		3,00		3,38

Instancias de comparacion sobre TSPLIB				
76.tsp	561	4,28	556	3,35
pr76.tsp	114742	6,09	110316	1,99
t99.tsp	1270	4,87	1297	7,10
oa100.tsp	21292	0,05	21315	0,16
krob100.tsp	23200	4,78	22199	0,26
oc100.tsp	21605	4,13	21569	3,95
krod100.tsp	22255	4,51	22200	4,25
kroe100.tsp	22411	1,55	22406	1,53
100.tsp	8508	7,56	8218	3,89
eil101.tsp	654	3,97	659	4,77
lin105.tsp	14905	3,66	15241	5,99
107.tsp	44751	1,01	44482	0,40
pr124.tsp	61144	3,58	61065	3,45
ier127.tsp	123592	4,49	125395	6,01
136.tsp	100320	3,67	106233	9,78
pr144.tsp	60296	3,00	60665	3,64
oa150.tsp	27653	4,26	28160	6,17
rob150.tsp	26835	2,70	26885	2,89
pr152.tsp	75904	3,02	76212	3,43
59.tsp	45272	7,59	46265	9,95
rat195.tsp	2444	5,21	2438	4,95
d198.tsp	16162	2,42	16047	1,69
oa200.tsp	31184	6,18	30543	4,00
krob200.tsp	30669	4,19	31148	5,81
225.tsp	136700	7,94	136700	7,94
226.tsp	81486	1,39	81358	1,23
eil262.tsp	2575	8,28	2520	5,97
264.tsp	53425	8,73	54161	10,23
299.tsp	51066	5,97	49518	2,75
lin318.tsp	44307	5,42	45179	7,49
417.tsp	11927	0,56	12915	8,89
439.tsp	111434	3,93	112680	5,10
d493.tsp	36771	5,05	36314	3,75
532.tsp	29382	6,13	29305	5,85
574.tsp	39654	7,45	39168	6,13
rat575.tsp	7239	6,88	7272	7,37
657.tsp	51868	6,04	52211	6,74
724.tsp	44926	7,20	45006	7,39
rat783.tsp	9414	6,90	9424	7,02
w1379.tsp	60656	7,09	59872	5,71
u1432.tsp	163571	6,93	164126	7,29
1655.tsp	68507	10,27	68343	10,00
Promedio		4,97		5,15
esviación estandar		2,33		2,71

Instancias de comparacion con Reinelt	ocimsth1	% sobre el óptimo	ocimsth2	% sobre el óptimo
d198.tsp	16460	4,31	16282	3,18
pcb442.tsp	55038	8,39	55248	8,80
u574.tsp	38886	5,37	38629	4,67
p654.tsp	36533	5,46	35618	2,81
rat783.tsp	9415	6,92	9589	8,89
pcb1173.tsp	62250	9,42	61730	8,50
d1291.tsp	55979	10,62	56826	12,29
u1432.tsp	162779	6,45	164042	7,27
fl1577.tsp	24144	9,08	24925	12,61
d1655.tsp	68747	10,65	67772	9,08
d2103.tsp	93265	17,04	89892	12,81
pr2392.tsp	407634	7,83	413405	9,36
pcb3038.tsp	149258	8,46	149700	8,78
Promedio		8,46		8,39
Desviación estandar		3,25		3,26

Instancias de comparacion sobre TSPLIB				
eil76.tsp	557	3,53	565	5,02
pr76.tsp	112506	4,02	113089	4,56
rat99.tsp	1294	6,85	1256	3,72
kroa100.tsp	21583	1,41	21292	0,05
krob100.tsp	22565	1,91	23453	5,93
kroc100.tsp	21313	2,72	21431	3,29
krod100.tsp	22578	6,03	21770	2,24
kroe100.tsp	22636	2,57	22585	2,34
rd100.tsp	8298	4,91	8156	3,11
eil101.tsp	659	4,77	660	4,93
lin105.tsp	14583	1,42	15457	7,50
pr107.tsp	45180	1,98	44628	0,73
pr124.tsp	59800	1,30	60177	1,94
bier127.tsp	123138	4,11	122477	3,55
pr136.tsp	99048	2,35	101000	4,37
pr144.tsp	60159	2,77	61309	4,74
kroa150.tsp	28071	5,83	27474	3,58
krob150.tsp	26765	2,43	26689	2,14
pr152.tsp	75479	2,44	76685	4,08
u159.tsp	45620	8,41	44778	6,41
rat195.tsp	2527	8,78	2512	8,14
d198.tsp	16460	4,31	16282	3,18
kroa200.tsp	30792	4,85	31522	7,33
krob200.tsp	30837	4,76	30305	2,95
ts225.tsp	139699	10,31	139699	10,31
pr226.tsp	82276	2,37	81410	1,30
gil262.tsp	2569	8,03	2523	6,10
pr264.tsp	51354	4,52	50119	2,00
pr299.tsp	51724	7,33	51636	7,15
lin318.tsp	44358	5,54	43786	4,18
fl417.tsp	12226	3,08	12367	4,27
pr439.tsp	114740	7,02	115328	7,57
d493.tsp	36483	4,23	37604	7,43
att532.tsp	29227	5,57	29202	5,48
u574.tsp	38886	5,37	38629	4,67
rat575.tsp	7185	6,08	7221	6,61
d657.tsp	52001	6,32	52711	7,77
u724.tsp	44173	5,40	45038	7,46
rat783.tsp	9415	6,92	9589	8,89
nrw1379.tsp	59939	5,83	60259	6,39
u1432.tsp	162779	6,41	164042	7,24
d1655.tsp	68747	10,65	67772	9,08
Promedio		4,89		4,99
Desviación estandar		2,37		2,45

Instancias de comparacion con Reinelt	oclmsth3	% sobre el óptimo	oclmsth4	% sobre el óptimo
d198.tsp	16247	2,96	16315	3,39
pcb442.tsp	55088	8,49	53380	5,12
u574.tsp	39064	5,85	39170	6,14
p654.tsp	36531	5,45	36213	4,53
rat783.tsp	9481	7,67	9520	8,11
pcb1173.tsp	62654	10,13	62068	9,10
d1291.tsp	54746	8,18	56406	11,46
u1432.tsp	164114	7,32	164019	7,26
fl1577.tsp	24422	10,34	25071	13,27
d1655.tsp	68371	10,05	69015	11,09
d2103.tsp	88967	11,65	92139	15,63
pr2392.tsp	413167	9,29	406492	7,53
pcb3038.tsp	148441	7,87	149008	8,28
Promedio		8,09		8,53
Desviación estandar		2,35		3,55

Instancias de comparacion sobre TSPLIB				
eil76.tsp	564	4,83	552	2,60
pr76.tsp	112493	4,01	110167	1,86
rat99.tsp	1227	1,32	1301	7,43
kroa100.tsp	21292	0,05	21967	3,22
krob100.tsp	23255	5,03	23437	5,85
kroc100.tsp	21586	4,03	22081	6,42
krod100.tsp	21812	2,43	22788	7,02
kroe100.tsp	22810	3,36	22908	3,81
rd100.tsp	8156	3,11	8152	3,06
eil101.tsp	649	3,18	663	5,41
lin105.tsp	15257	6,11	15303	6,43
pr107.tsp	45180	1,98	44961	1,49
pr124.tsp	61890	4,84	61480	4,15
bier127.tsp	126397	6,86	129757	9,70
pr136.tsp	99367	2,68	98171	1,45
pr144.tsp	60422	3,22	63417	8,34
kroa150.tsp	27505	3,70	27585	4,00
krob150.tsp	26993	3,30	27039	3,48
pr152.tsp	75840	2,93	77984	5,84
u159.tsp	45176	7,36	46500	10,50
rat195.tsp	2480	6,76	2603	12,05
d198.tsp	16247	2,96	16315	3,39
kroa200.tsp	30622	4,27	30697	4,53
krob200.tsp	30488	3,57	30781	4,57
ts225.tsp	139699	10,31	139699	10,31
pr226.tsp	82751	2,96	82376	2,50
gil262.tsp	2476	4,12	2553	7,36
pr264.tsp	51906	5,64	52052	5,94
pr299.tsp	51094	6,02	50963	5,75
lin318.tsp	44655	6,25	44283	5,36
fl417.tsp	12625	6,44	12269	3,44
pr439.tsp	117366	9,47	114880	7,15
d493.tsp	37374	6,78	36457	4,16
att532.tsp	29453	6,38	29258	5,68
u574.tsp	39064	5,85	39170	6,14
rat575.tsp	7194	6,22	7229	6,73
d657.tsp	51723	5,75	51771	5,85
u724.tsp	44774	6,83	44445	6,05
rat783.tsp	9481	7,67	9520	8,11
nrw1379.tsp	60325	6,51	60419	6,68
u1432.tsp	164114	7,29	164019	7,22
d1655.tsp	68371	10,05	69015	11,09
Promedio		5,06		5,76
Desviación estandar		2,27		2,57

Instancias de comparacion con Reinelt	ocimsth5	% sobre el óptimo	ocimsth6	% sobre el óptimo
d198.tsp	16002	1,41	16389	3,86
fnb442.tsp	55538	9,37	54147	6,63
lin574.tsp	39965	8,29	39565	7,21
p654.tsp	35554	2,63	37798	9,11
rat783.tsp	9492	7,79	9425	7,03
pcb1173.tsp	62232	9,39	63452	11,53
d1291.tsp	59477	17,53	59097	16,78
fl1432.tsp	164447	7,54	164623	7,65
fl1577.tsp	24299	9,78	25681	16,03
vm1655.tsp	70338	13,21	68848	10,82
d2103.tsp	93349	17,14	90076	13,04
pr2392.tsp	415585	9,93	418170	10,62
fnb3038.tsp	149921	8,94	150890	9,64
Promedio		9,46		10,00
Desviación estandar		4,64		3,73

Instancias de comparacion sobre TSPLIB				
eil76.tsp	561	4,28	557	3,53
pr76.tsp	112429	3,95	112328	3,85
rat99.tsp	1273	5,12	1244	2,73
kroa100.tsp	21929	3,04	21320	0,18
krob100.tsp	23314	5,30	22758	2,79
kroc100.tsp	21010	1,26	22562	8,74
krod100.tsp	21874	2,72	22099	3,78
kroe100.tsp	22510	2,00	22678	2,76
rd100.tsp	8416	6,40	8329	5,30
eil101.tsp	655	4,13	662	5,25
lin105.tsp	15431	7,32	15049	4,66
pr107.tsp	44577	0,62	45340	2,34
pr124.tsp	61142	3,58	59781	1,27
bier127.tsp	122846	3,86	128734	8,84
ch136.tsp	102484	5,90	102261	5,67
pr144.tsp	60278	2,97	59499	1,64
kroa150.tsp	27359	3,15	27138	2,31
krob150.tsp	27836	6,53	26477	1,33
pr152.tsp	74123	0,60	75525	2,50
u159.tsp	43424	3,19	44818	6,51
rat195.tsp	2473	6,46	2512	8,14
d198.tsp	16002	1,41	16389	3,86
kroa200.tsp	31113	5,94	31643	7,75
krob200.tsp	31305	6,35	31089	5,61
ts225.tsp	137083	8,24	137083	8,24
pr226.tsp	81859	1,85	87903	9,37
gil262.tsp	2524	6,14	2598	9,25
a264.tsp	53814	9,52	52691	7,24
pr299.tsp	51521	6,91	52214	8,35
lin318.tsp	44767	6,51	44082	4,88
fl417.tsp	12474	5,17	12823	8,11
pr439.tsp	112760	5,17	118808	10,81
pcb493.tsp	36833	5,23	36969	5,62
d1532.tsp	29147	5,28	29027	4,84
lin574.tsp	39965	8,29	39565	7,21
rat575.tsp	7294	7,69	7274	7,40
p657.tsp	52467	7,27	51678	5,66
rat724.tsp	44988	7,34	45315	8,12
rat783.tsp	9492	7,79	9425	7,03
vsw1379.tsp	60492	6,80	60051	6,03
flu1432.tsp	164447	7,50	164623	7,62
vm1655.tsp	70338	13,21	68848	10,82
Promedio		5,29		5,66
Desviación estandar		2,58		2,75

Instancias de comparacion con Reinelt	ocimsth7	% sobre el óptimo	ocimsth8	% sobre el óptimo
u198.tsp	16290	3,23	16028	1,57
pcb442.tsp	54643	7,61	53909	6,17
u574.tsp	39344	6,61	39636	7,40
p654.tsp	35523	2,54	35730	3,14
rat783.tsp	9520	8,11	9449	7,30
pcb1173.tsp	63358	11,37	61974	8,93
d1291.tsp	57864	14,34	56758	12,16
u1432.tsp	164492	7,57	166415	8,82
fl1577.tsp	24710	11,64	24319	9,87
d1655.tsp	67690	8,95	68107	9,62
vm2103.tsp	88864	11,52	92722	16,36
pr2392.tsp	416072	10,06	416351	10,14
pcb3038.tsp	150864	9,63	150293	9,21
Promedio		8,71		8,51
Desviación estandar		3,33		3,73

Instancias de comparacion sobre TSPLIB				
eil76.tsp	565	5,02	562	4,46
pr76.tsp	112588	4,09	109541	1,28
gr99.tsp	1288	6,36	1246	2,89
kroa100.tsp	21320	0,18	22263	4,61
krob100.tsp	23438	5,86	23477	6,03
kroc100.tsp	21356	2,93	22450	8,20
krod100.tsp	22581	6,04	22321	4,82
kroe100.tsp	23129	4,81	22423	1,61
rd100.tsp	8543	8,00	8125	2,72
eil101.tsp	682	8,43	652	3,66
lin105.tsp	15049	4,66	15666	8,95
pr107.tsp	44482	0,40	44482	0,40
pr124.tsp	59630	1,02	60800	3,00
ber127.tsp	124551	5,30	121757	2,94
pr136.tsp	104156	7,63	103609	7,07
pr144.tsp	60268	2,96	63306	8,15
kroa150.tsp	27108	2,20	27947	5,36
krob150.tsp	26543	1,58	27986	7,10
pr152.tsp	76670	4,06	75141	1,98
u159.tsp	45217	7,45	44412	5,54
rat195.tsp	2507	7,92	2519	8,44
d198.tsp	16290	3,23	16028	1,57
kroa200.tsp	31485	7,21	30992	5,53
krob200.tsp	30820	4,70	31751	7,86
ts225.tsp	137083	8,24	137083	8,24
pr226.tsp	82627	2,81	82040	2,08
gil262.tsp	2622	10,26	2539	6,77
a280.tsp	53620	9,13	54401	10,72
lin299.tsp	52339	8,61	52270	8,46
fl318.tsp	44957	6,97	45086	7,27
pr317.tsp	12770	7,66	12341	4,05
pr439.tsp	120311	12,21	110903	3,44
rat493.tsp	36807	5,16	37145	6,12
u532.tsp	29097	5,10	29561	6,77
u574.tsp	39344	6,61	39636	7,40
rat575.tsp	7272	7,37	7295	7,71
p657.tsp	51342	4,97	52222	6,77
u724.tsp	44867	7,06	45070	7,54
rat783.tsp	9520	8,11	9449	7,30
vm1379.tsp	60320	6,50	60464	6,76
u1432.tsp	164492	7,53	166415	8,79
d1655.tsp	67690	8,95	68107	9,62
Promedio		5,84		5,71
Desviación estandar		2,69		2,61

Capítulo 12

Apendice C

Tablas de comparación de OCIMST, OCIRNG, OCIGG, OCIDT y Reinelt

OCI				
	MST	RNG	GG	TD
Orden por promedios sobre los óptimos (instancias de comparacion con Reinelt)	H3(8.09)	H6(8.55)	H7(7.98)	H5(8.27)
	H2(8.39)	H7(8.63)	H6(8.67)	H3(8.31)
	H1(8.46)	H3(8.95)	H5(8.89)	H7(8.33)
	H8(8.51)	H5(9.22)	H8(9.04)	H6(8.55)
	H4(8.53)	H2(9.34)	H3(9.57)	H8(8.69)
	H7(8.71)	H4(9.46), H8(9.46)	H1(9.79)	H2(8.85)
	H5(9.46)	H1(9.82)	H4(9.98)	H4(9.00)
	H6(10.00)		H2(10.37)	H1(9.51)

OCI				
	MST	RNG	GG	TD
Orden por promedios sobre los óptimos (instancias de TSPLIB)	H1(4.89)	H6(5.08)	H7(4.97)	H6(4.92)
	H2(4.99)	H8(5.19)	H5(5.10)	H5(4.95)
	H3(5.06)	H7(5.28)	H8(5.15)	H8(4.98)
	H5(5.29)	H5(5.30)	H6(5.22)	H7(5.07)
	H6(5.66)	H2(5.61)	H4(6.27)	H1(5.16)
	H8(5.71)	H1(5.94)	H3(6.59)	H3(5.23)
	H4(5.76)	H4(6.03)	H2(6.64)	H4(5.36)
	H7(5.84)	H3(6.13)	H1(6.65)	H2(5.48)

Mejores resultados de Reinelt	Nearest Neighbor % sobre el óptimo	2-Opt % sobre el óptimo	Lin-Kernighan %sobre el óptimo
d198.tsp	17,00	3,86	7,52
pcb442.tsp	11,99	5,26	2,50
u574.tsp	18,16	8,00	4,26
p654.tsp	32,94	3,40	1,97
rat783.tsp	19,50	6,72	2,96
pcb1173.tsp	15,21	6,22	2,73
d1291.tsp	14,95	6,43	1,96
u1432.tsp	17,71	8,33	3,13
fl1577.tsp	15,57	9,12	2,94
d1655.tsp	18,50	7,08	3,86
d2103.tsp	9,30	3,91	3,02
pr2392.tsp	20,58	7,70	4,19
pcb3038.tsp	19,95	8,15	3,27
Promedio	**17,80**	**6,48**	**3,41**
Desviación estandar	5,56	1,87	1,43

Bibliografía

[1] Active Geometry Group, Johns Hpkins University, Extracting the Geometry of the Vascular Tree, `http://blaze.cs.jhu.edu/grad/lundberg/agg/projects/vasc.html`

[2] R. S. Barr, B. L. Golden, J. P. Kelly, M.G.C. Resende and W. R. Stewart, Designing and Reporting on Computational Experiments with Heuristic Methods, Journal of heuristics, 1 :9-32 (1995), Kluwer Academic Publisher.

[3] J. Beardwood, J.H. Halton and J.M. Hammersley, The Shortest Path Through Many Points, Proc. Cambridge Philos. Soc. **55** (1959) 299-327.

[4] J.L. Bentley, Approximation Algorithms for Convex Hulls Communications of the ACM No.1, Vol. 25, Jan 1982

[5] J.L. Bentley, *kd−trees* for semidynamics point sets, Sixth Annual Symposium on Computational Geometry, 187-197. Berkeley CA, June 1990.

[6] J. L. Bentley, Fast Algorithms for Geometric Traveling Salesman Problems, Operational Research Society of America, Vol 4, Nro. 4, Fall 1992.

[7] M. de Berg, M. van Krevel, M. Overmars and O. Schwarzkopf, Computational Geometry, Algorithms and Applications, 1991, Springer

[8] E. Bonomi and J.L. Lutton, The N-city Travelling Salesman Problem and the Metropolis Algorithm, SIAM Review **26** (1984) 551-568.

[9] D.P. Bovet and P. Crescenzi, Introduction to the Theory of Complexity, C.A.R. Hoare Series Editor, Prentice Hall International Limited, 1994.

[10] P. Crescenzi and V. Kann, A Compendium of NP Optimization Problems `http://www.nada.kth.se/theory/problemlist.html`

[11] D. Cheriton and R.E. Tarjan, Finding Minimum Spanning Trees, SIAM J. Comput., 5(4), 724-742, Dec. 1976.

[12] N. Christofides, Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem, Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburg, PA, (1976).

[13] C. O'Dunlaing and C.K. Yap, A "retraction" method for planning the motion of a disc, Journal of Algorithms, vol. 6, 1985.

[14] L. Few, The Shortest Path and the Shortest Road Through N Points, Mathematika, **2** (1955) 141-144.

[15] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, (Freeman, San Francisco, 1979).

[16] D.S. Johnson, Local Optimization and the Traveling Salesman Problem, Proceedings of the 17th Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, V. 443, Springer-Verlag, Berlin, 1990, pag. 446-461

[17] D.S. Johnson, C.H. Papadimitriou and M. Yannakakis, How easy is local search?, Proceedings of Foundations of Computer Science, 1985.

[18] V. Kantabutra, Traveling salesman cycles are not always subgraphs of Voronoi duals, Inform. Process. Lett., V. 16, 1983, 11-12.

[19] R. Karp, F. Alizadeh, L. newberg and D. Weisser. Phisical Mapping of Chromosomes: A Combinatorial Problem in Molecular Biology. TR-92-066, September 29, 1992

[20] R.M. Karp, Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problem in the Plane, Math. Oper. Res. **2** (1977) 209-224.

[21] R.M. Karp, A Patching Algorithm for the Nonsymmetric Traveling-Salesman Problem, SIAM J. Comput. **8** (1979) 561-573.

[22] Krasnogor N., Lyardet F.D., On The Hardness Of Navigational Path Prediction On Large Hypermedias Systems, LIFIA internal report, `http://www-lifia.info.unlp.edu.ar/~natk/`

[23] N. Krasnogor, F.D. Lyardet Hypermedia Exploration and Link Discovery: Have I Seen You Before? LIFIA internal report, `http:/www-lifia.info.unlp.edu.ar/~natk/`

[24] P.D. Krolak, W. Felts and G. Marble, A Man-Machine Approach Toward Solving the Traveling Salesman Problem, Comm. ACM **14** (1971) 327-334.

[25] D.E. Knuth, The Stanford GraphBase, A Platform For Combinatorial Computing, Addison-Wesley, 1993

[26] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization, (Wiley-Interscience, Chichester, 1985).

[27] S. Lin, Computer Solutions of the Traveling Salesman Problem, Bell System Tech. Journal **44** (1965) 2245-2269.

[28] S. Lin and B.W. Kernighan, An Effective Heuristic Algorithm for the Traveling Salesman Problem, Oper. Res. **21** (1973) 498-516.

[29] M.O. Magnasco, Correlations in Cellular Patterns, Philosophical Magazine B, V. 69, N. 3, 1994, pag. 397-429

[30] P. Moscato and M.G. Norman, A "Memetic" Approach for the Traveling Salesman Problem. Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems, in: M. Valero *et al.* eds., Proceedings of the International Conference on Parallel Computing and Transputer Applications, Barcelona, Spain, 1992, (IOS Press, Amsterdam, 1992) 177-186.

[31] P. Moscato, An Introduction to Population Approaches for Optimization and Hierarchical Objective Functions: The role of Tabu Search, Annals of Operations Research, V. 41, n. 1-4, 1993, pag. 85-121.

[32] M.G. Norman and P. Moscato, The Euclidean Traveling Salesman Problem and a Space-Filling Curve, Chaos, Solitons and Fractals, Vol. 6, pages 389-397, 1995.

[33] P. Moscato and M.G. Norman, An Analysis of the Performance of Traveling Salesman Heuristics on Infinite-Size Fractal Instances in the Euclidean Plane, CeTAD - Universidad Nacional de La Plata, 1994, submitted - Oct. 1994

[34] M. Padberg and G. Rinaldi, Optimization of 532-City Symmetric TSP, Operations Research Letters, **6** (1987) 1-7.

[35] F.P. Preparata and M.I. Shamos, Computational Geometry, an Introduction, Springer-Verlag (1985).

[36] G. Reinelt, TSPLIB - A Traveling Salesman Problem Library, European Journal of Operations Research, 52 (1991) 125.

[37] G. Reinelt, Fast Heuristics for Large Geometric Traveling Salesman Problems, ORSA Journal on Computing, V. 4, N. 2, 1992, pag. 206-217

[38] K. R. Gabriel and R.R. Sokal, A New Statistical Approach to Geographic Variation Analysis, Systematic Zoology 18, 259 - 278 (1969)

[39] W.R. Jr. Stewart, Euclidean Traveling Salesman Problems and Voronoi Diagrams, presentado en la conferencia de ORSA-CSTS, Williamsburg, VA, 1992.

[40] P. A. Moscato, TSPBIB The Travelling Salesman Problem Home Page, `http://www.densis.fee.unicamp.br/~moscato/TSPBIB_home.html`

[41] `http://www.mathsoft.com/asolve/constant/sales/sales.html`

[42] Y.S. Abu-Mostafa, Hints and the VC dimension, submitted to Neural Computation, preprint, July 1992 version.

[43] Y.S. Abu-Mostafa, Finantial market applications of learning from hints, in, Neural Networks in the capital market, Ed. by A.N. Refenes, (Wiley, England, 1994).

[44] E. Amaldi, On the complexity of training perceptrons, Proceedings of the 1991 International Conference on Artificial Neural Networks, T. Kohonen, K. Makisara, O. Simula and J. Kangas, eds. North Holland, Amsterdam, 1991. pp 55-60.

[45] S. Arora, Polynomial time approximation schemes for Euclidean TSP and other geometric problems, FOCS 96, 1996

[46] G. Baum, M. Sagastume, Problemas, Lenguajes y Algoritmos, Campinas 1986, Editora Da Unicamp.

[47] E.B. Baum, What can Back Propagation and k-nearest neighbor learn with feasible sized sets of examples ?, in Neural Networks, EURASIP Workshop 1990 Proc., pp. 2-25.

[48] E.B. Baum, Neural net algorithms that learn in polynomial time from examples and queries, IEEE Transactions on Neural Networks, **2**, No. 1, Jan. 1991.

[49] A. Blum, R.L. Rivest, Training a 3 node neural networks is *NP-Complete*, Advances in Neural Information Processing Systems, 1, D.S. Touretsky, Ed. San Mateo, CA, Morgan Kaufmann, 1988, pp. 494-501.

[50] K.S. Booth, G. S. Lueker. *Testing For The Consecutive Ones Property, Interval Graphs, And Planarity Using PQ-tree Algorithms*, J. Comput. Sys. Sci., 13:335-379,1976.

[51] G. Cornuéjols and G.L. Nemhauser, Tight bounds for Christofides' Traveling Salesman Heuristic, Math. Programming **14** (1978) 116-121.

[52] G.A. Croes, A Method for Solving Traveling Salesman Problems, Oper. Res. **6** (1958) 791-812.

[53] W.J.A. Fyfe, Invariance hints and the VC dimension, PhD Thesis, Computer Science Department, Caltech, (1992) (Caltech-CS-TR-92-20).

[54] H. Gabow, An efficient implementation of Edmond's maximun matching algorithm, Technical report 31. Computer Science department, Stanford University, 1972.

[55] P.W. Goldberg, M.C. Golumbic, H. Kaplan, R. Shamir, *Four Strikes Against Physical Mapping Of DNA*, Technical Report 287/93, December 1993, The Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University.

[56] M. C. Golumbic, H. Kaplan and R. Shamir Algorithms and Complexity of Sandwhich problems in Graphs

[57] J. S. Judd, Complexity of connectionist learning with various node functions, Technical Report 87-60, University of massachusetts, Amherts, MA, 1987.

[58] J. S. Judd, Neural networks design and the complexity of learning, MIT press, Cambridge MA, 1990.

[59] H. Kaplan, R. Shamir, M. Golumbic, On The Complexity of DNA Physical Mapping, Advances In Applied Mathematics, 15 Pag. 251-261 (1994).

[60] J. H. Holland. Emergence, from Chaos to Order, Adisson-Wesley Publishing Company, Inc. 1997.

[61] N. Krasnogor, P. Moscato, M.G. Norman, Delaunay and Voronoi Helps Trucks and Salesmens, Anales del II Encuentro De Investigadores Del Grupo Montevideo, 1995.

[62] P. Moscato, N. Krasnogor Learning, Hierarchies and Hints In preparation.

[63] P. Moscato and N. Krasnogor, Semidynamic Point Sets for Polynomial-Time Learning, Presentado en la Conferencia Latino Americana de Investigación Operativa - CLAIIO, Santiago de Chile, Chile, 1994.

[64] N. Krasnogor and P. Moscato, A new hybrid heuristic for large geometric travelling salesman problems based on the delaunay triangulation, Presentado en Victoria, Brasil , Encuentro de la Sociedad Brasilera de Investigación Operativa, Aceptado para los anales de SoBraPos, 1995.

[65] Laszlo Lovasz, Peter Gacs, Computation complexity 1994.

[66] J. S. B. Mitchell, Guillotine subdivisions approximate polygonal subdivisions: Part II - A simpe polynomial-tyme approximation scheme for the geometric K-MST, TSP, and related problems, To appear in SIAM Journal of computing.

[67] A. Díaz, F. Glover, H. M. Ghaziri, J.S. Gonzáles Velarde, M. Laguna, P.A. Moscato and F.T. Tsen, Optimizacin Heurstica y Redes Neuronales, Editorial Paraninfo.

[68] P. Moscato, On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Caltech Concurrent Computation Program Report 826, CalTech, Pasadena CA, (1989).

[69] Rajeev Motwani, Lectures Notes On Approximation Algorithms - Vol. 1, Department of Computer Science - Stanford University.

[70] N.J. Radcliffe, Forma Analysis and Random Respectful Recombination, in: R.K. Belew and L.B. Booker, eds., Proceedings of the Fourth International Conference on Genetic Algorithms, (Morgan Kauffman, San Mateo, 1991) 222-229.

[71] D.J. Rosenkrantz, R.E. Stearns and P.M. Lewis, An Analysis of Several Heuristics for the Traveling Salesman Problem, SIAM J. Comput **6** (1977) 563.

[72] S. Suddarth and A. Holden, Symbolic neural systems and the use of hints for developing complex systems, International Journal of Machine Studies, **35** (1991) 291.

[73] L. Trevisan, When Hamming Meets Euclid: the approximability of geometric TSP and MST. Proceedings of the twenty-ninth annual ACM symposium on theory of computing, 1997, ACM press.

[74] L.G. Valiant, A theory of the learnable, Communications of the ACM, **27** (1984) 1134-1142.

[75] J.L. Borges, A Leopoldo Lugones, poema.

[76] N. Korte and R.H. Mohring. An Incremental Linear Time Algorithm for Recognizing Interval graphs. SIAM J. Comp., 18, 68-81, 1989.

[77] L.G. Valiant, A theory of the learnable, Communications of the ACM, **27** (1984) 1134-1142.

[78] W. Grosky, F. Fotouhi, I. Sethi Using Metadata for the Intelligent Browsing of Structured Media Objects SIGMOD RECORD, Vol 23, No 4, December 1994.

[79] J. Jaromczyc and G. Toussaint, Relative Neighborhood Graphs and Their Relatives, Proceedings of the IEEE, Vol 80 Nr 9, September 1992, Pag. 1502-1517

[80] Terry Jones, Crossover, Macromutation, and Population based Search, Proceedings of the Sixth International Conference on Genetic Algotihms, 73-80, Morgan Kauffman, 1995.

[81] G.F. Luger and W. A. Stubblefield, Artificial intelligence - Structures and Strategies for Complex Problem Solving, The Benjamin/Cummings Publishing Company, Inc., 1993.

[82] C.H. Papadimitriou and M. Yannakakis, The Travelling Salesman Problem With Distances One and Two, Mathematics of Operations Research, Vol 18, Nro 1, February 1993.

[83] C. H. Papadimitriou, The complexity of the Lin-Kernighan Heuristic for the Travelling Salesman Problem, Siam Journal Of Computing, Vol. 21, No. 3, pp. 450-465, June 1992

[84] E. Rich, Artificial Inteligence McGraw-Hill International Editions, 1986.