Università di Roma "La Sapienza"

Corso di laurea in Informatica v.o.

Sulla ricerca di piccoli alberi di copertura minimi nel piano euclideo

Laureando

Ezio Sperduto

Matr. 11115441

Relatore

Prof. **Angelo Monti**

Anno accademico 2003/2004

Indice

Premessa

"La mente non ha bisogno, come un vaso, di essere riempita, ma piuttosto, come legna, di una scintilla che l'accenda vi infonda l'impulso della ricerca e un amore ardente per la verità."

Plutarco, De recta ratione audiendi

Il termine *algoritmo*, che significa procedimento di calcolo, è una versione moderna del termine latino medievale *algorismus*, che deriva dal nome del matematico usbeco Mohammed ibn-Musa *al-Khowarizmi*, vissuto nel IX secolo d.C. e famoso per aver scritto un noto trattato di algebra.

Col termine algoritmo si intende la descrizione precisa di una sequenza di azioni che devono essere eseguite per giungere alla risoluzione di un problema computazionale. La nozione di algoritmo, pur essendo vicina alla nozione di dimostrazione matematica, era già presente nelle origini della civiltà umana ben prima che fosse definita la nozione di dimostrazione.

In questo mio lavoro, che è il completamento di una interessantissima esperienza universitaria e di vita, viene trattato un problema combinatoriale e alcuni algoritmi approssimanti che lo risolvono. Mi sono avvalso, per questo studio, delle risorse che la rete mette a disposizione, ma soprattut-

to delle fonti attinte presso il Dipartimento di Informatica dell'Università di Roma "La Sapienza". Ho lavorato sotto la guida del professor Angelo Monti, al quale sono infinitamente grato per l'attenzione ed il tempo che mi ha dedicato. Insieme con lui ho esplorato molte sfaccettature di un argomento particolarmente stimolante fino a giungere ad un esauriente conoscenza di una delle frontiere del sapere informatico. In questo documento espongo un'attenta analisi prestazionale di algoritmi di recente pubblicazione. Vorrei dunque ringraziare tutte le persone che mi hanno aiutato, sia nel corso di quest'ultima fatica che durante tutto il mio studio accademico. Una particolare menzione va fatta al mio docente di analisi Enrico Rogora, per il quale nutro un enorme stima. Tra le persone a cui sono grato ci sono ovviamente anche i miei amici che mi hanno sostenuto e aiutato, tra i quali vorrei ricordare particolarmente quelli con i quali fin dai tempi del liceo ho condiviso la passione per l'informatica e quelli che mi hanno affiancato in questo corso di laurea. Proprio con questi ultimi mi sono trovato più volte a discutere delle più svariate questioni informatiche, sia inerenti alla preparazione di un esame che non. Sono molto entusiasta di quello che ho studiato nel periodo universitario e per questo sono riconoscente a tutti quelli che mi hanno istruito e che continuano a ricercare nell'ambito scientifico.

Il mio debito verso la mia famiglia è più grande che verso chiunque altro. Questo lavoro è dedicato a loro.

Capitolo 1

Introduzione

Immaginiamo che una compagnia telefonica debba servire delle città, e la scelta di queste dipenda dal capitale che è disposta ad investire nell'impresa. Avendo a disposizione una quantità limitata di infrastrutture, la compagnia è disposta ad individuare le città che, risultando idonee al cablaggio, siano tra loro più vicine possibile, in modo da minimizzare la spesa per i cavi di collegamento.

Volendo trovare un modello matematico per un tale problema, possiamo schematizzare le città come i vertici di un grafo. Gli archi del grafo rappresentano le possibili connessioni tra una città e l'altra. Una funzione peso che associ ad ogni arco un valore reale rappresenta il costo dei collegamenti fra le città. Un valore intero positivo ci servirà per indicare il numero di città che dobbiamo collegare. Il problema allora diventa: avendo un grafo nondiretto G, una funzione peso c e un intero k, trovare l'albero di copertura su G che colleghi k vertici di costo minimo rispetto a c.

Questo problema è conosciuto in letteratura come il problema k-MST,

una versione generalizzata del problema della ricerca dell'albero di copertura di costo minimo. Il problema della ricerca dell'MST (Minimum Spanning Tree) è un problema che ha avuto un ampio studio nell'ambito della teoria dei grafi. Già negli anni '50 lo studio della ricerca dell'albero di copertura di costo minimo ha avuto una soluzione con la scoperta, da parte di Kruskal [1] prima e di Prim [2] poi, di algoritmi con complessità polinomiale molto veloci. L'introduzione del vincolo della limitazione dei vertici da collegare porta ad una generalizzazione del problema (il problema MST è un istanza del k-MST ottenibile prendendo il parametro k pari alla cardinalità dei vertici del grafo). Alla luce degli ultimi studi fatti, la versione del problema limitata a k vertici è risultata più difficile nei confronti di quella relativa a tutti i vertici del grafo.

Dato il problema k-MST spesso torna utile, per essere più aderente alle applicazioni reali come nell'esempio citato prima, restringersi al caso in cui il grafo viene generato all'interno di una metrica o all'interno dello spazio euclideo. In questo ultimo scenario abbiamo uno spazio sul quale sono disposti dei punti, i vertici del grafo. Il grafo è completo e l'arco che insiste tra due vertici rappresenta il segmento che li connette. Il peso di un arco è dato dalla distanza dei suoi estremi. In questa situazione si possono sfruttare le proprietà della geometria euclidea per agevolare la ricerca dell'albero di lunghezza minima.

1.1 Risultati allo stato attuale

Il problema k-MST è stato trattato da un numero considerevole di ricercatori, e questo spiega la grande quantità di documenti scritti in proposito su riviste e libri. Il problema fu introdotto nel 1993 nell'articolo [10] dal bielorusso Zelikovsky e dal moldavo Lozovanu e parallelamente, ma indipendentemente, da Fischetti et al. in [11] e da Ravi et al. in [3]. In questi documenti viene esposta la dimostrazione di NP-completezza del problema (trattata nel Capitolo 2), cosa che ha suscitato un interesse maggiore nei confronti dell'argomento. Infatti gia in [3] Ravi et al. espongono un loro primo algoritmo approssimante per il calcolo del k-MST, sia nel caso generale che nel piano euclideo. L'algoritmo, trattato per esteso nel Capitolo 5, raggiunge un fattore di approssimazione di $O(\sqrt{k})$ per il caso generale, e $O(k^{1/4})$ nel caso del piano. Per quanto riguarda il caso generale un miglioramento si ebbe nel 1995 con il lavoro di Awerbuch et al. [12] che trovarono un algoritmo approssimante basato sulla tecnica greedy con un fattore di approssimazione $O(\log^2 k)$. Allo stesso anno risale anche il risultato di Rajagopalan e Vazirani [13] che garantisce un fattore di approssimazione pari a $O(\log k)$. Un netto passo avanti si registrò sempre nel 1995 quando nel lavoro di Blum, Ravi e Vempala [14] si dimostrò che, tramite il metodo di clusterzzazione dovuto a Goemans e Williamson [17], è possibile garantire, in tempo polinomiale, un fattore di approssimazione costante, e cioè 17. Il più recente algoritmo per il problema k-MST è quello proposto da Naveen Garg [18] che calcola un albero di copertura su k vertici di costo non superiore al triplo di quello ottimo. In seguito, in un articolo scritto da Arya e Ramesh [19] è stato ottenuto un miglioramento dal punto di vista del fattore di approssimazione.

Apportando modifiche all'algoritmo di Garg, i due ricercatori sono riusciti ad ottenere un fattore di 2.5 per il problema k-MST nel caso generale, a tutt'oggi il migliore fattore conosciuto. Inoltre Cheung e Kumar [22] hanno studiato a fondo il problema nel 1994, spiegando che è molto comune nel dominio delle *reti di comunicazione* e nel calcolo distribuito fault-tolerant con il nome di "quorum-cast problem".

Analizzando il caso del piano, la maggior parte degli studiosi ha sfruttato le proprietà derivanti dalla metrica euclidea (L_2) o L_1. Il successivo sviluppo dopo l'algoritmo di Ravi si deve a Garg e Hochbaum [20] che, migliorando l'idea utilizzata in [3], portarono il rapporto di approssimazione ad un valore di $O(\log k)$, parallelamente a Mata e Mitchell [21]. L'algoritmo di Garg e Hochbaum verrà trattato per esteso nel Capitolo 6. Eppstein in [23] ha diminuito ancora il fattore di approssimazione arrivando ad un valore $O(\log k / \log\log n)$ e ha esposto una tecnica generale per ottimizzare il tempo di calcolo (come funzione di n) degli algoritmi esistenti. In aggiunta ha dimostrato che la soluzione esatta del problema k-MST può essere trovata in un tempo $2^{k \log k} n + O(n \log n)$, che si riduce a $O(n \log n)$ per un valore di k fissato. Anche in questa versione euclidea, ovviamente, si è passati poi ad algoritmi più performanti, che approssimano la soluzione con un fattore costante. Questo è stato grazie alla tecnica chiamata *suddivisione a ghigliottina* ideata da Mithcell in [16] e utilizzata da Blum, Chalasani e Vempala [15] che ha permesso un calcolo estremamente vicino all'ottimo per il problema. La tecnica geometrica della ghigliottina ha inoltre mostrato un alta adattabilità ai problemi combinatoriali nel piano. Probabilmente è per questo che nel 1998 lo stesso Mitchell pubblicò un documento [24] nel quale si riportava

la prova di uno schema di approssimazione polinomiale (vedere Capitolo 2) per versioni euclidee di molti famosi problemi combinatoriali come il k-MST ed il problema del commesso viaggiatore. La tecnica di approssimazione fa perno sulle divisioni a ghigliottina, con le quali si riesce ad ottenere una soluzione tanto vicina all'ottimo quanto si vuole. Lo stesso risultato fu pubblicato pochi mesi prima da Sanjeev Arora [25], il quale mediante una serie di strumenti geometrici (portali, reticoli, traslazioni, etc.) riusciva ad ottenere risultati altrettanto soddisfacenti. Le due metodologie, nonostante appaiano diverse, mostrano molti punti di contatto, cosa che ha suscitato dispute relativamente alla paternità della scoperta. Tuttavia è oramai assodato che sia Mitchell che Arora hanno ottenuto indipendentemente il risultato massimo in relazione al problema del k-MST nel piano euclideo.

1.2 Motivazioni e metodologie adottate per lo studio

Dopo aver analizzato la vasta letteratura presente nel campo scientifico, in questo lavoro mi propongo di trattare in dettaglio alcune tecniche utilizzate per risolvere il problema k-MST ristretto al piano euclideo. Gli algoritmi trattati in quest'opera sono degli algoritmi approssimanti, ovvero che non garantiscono di produrre una soluzione esatta, ma ci si avvicinano tanto più quanto sono buoni. La scelta di trattare algoritmi approssimanti è dovuta all'arduità del problema, trattata nel Capitolo 2, che sembra non consentire il calcolo in tempo efficiente di soluzioni esatte, salvo l'ipotesi di un futuro risultato nella relazione tra le classi di complessità P e NP. Gli algoritmi

trattati, come del resto la maggior parte di quelli studiati, hanno un tempo di esecuzione polinomiale, e hanno fattori di approssimazione differenti. La selezione degli algoritmi presi per l'analisi è stata un ingrediente importante in questo lavoro. Un'indagine approfondita dei lavori esistenti rivela che tanto più un algoritmo ha un rapporto di approssimazione basso, tanto più avrà un tempo di calcolo elevato. In tal proposito questo studio focalizza l'attenzione su un gruppo di euristiche che, pur approssimando in maniera limitata la soluzione, presentano un costo computazionale ragionevole. Questa scelta ci permette di analizzare, con una normale potenza di calcolo, istanze di notevoli dimensioni del problema. In aggiunta sono proposte in questa sede due nuove tecniche per la ricerca del k-MST euclideo che hanno permesso uno spunto di creatività e un raffronto con gli algoritmi esistenti. Le valutazioni delle euristiche trattate sono state effettuate mediante un'implementazione corretta su elaboratore, e i risultati sono stati riportati nella apposita sezione, Capitolo 7.

1.3 Struttura del lavoro

Questo documento è concepito in modo unitario, fornendo validi legami tra le parti costituenti. Nel Capitolo 2 possiamo trovare una buona parte di nozioni fondamentali di informatica teorica poste per rendere apprezzabile e comprensibile ogni parte di questo lavoro. Un lettore particolarmente smaliziato potrebbe, tuttavia, sorvolare sui concetti di base, evitando letture di cose già note per l'esperienza. In realtà il Capitolo 2 contiene notevoli riferimenti al problema trattato e lega gli argomenti esposti al caso partico-

lare, questo per renderne più gradevole la lettura. Inoltre contiene, in merito alla esposizione delle classi di complessità computazionale, la prova che il problema che stiamo analizzando, il k-MST euclideo, appartiene alla classe dei problemi probabilmente intrattabili. Nelle parti successive di questo documento è possibile trovare, suddivise per capitoli, le varie trattazioni in dettaglio delle euristiche prese in esame e implementate. Nel Capitolo 7 si possono invece apprezzare i risultati sperimentali ottenuti dalle esecuzioni dei programmi. Si possono osservare le valutazioni derivanti dalle comparazioni dei vari risultati sia in modo numerico che in modo grafico. Inoltre si trovano le riflessioni in merito al lavoro svolto, le possibili espansioni di questo e le tematiche di interesse che nel corso dello studio sono saltate fuori e che consentono un approfondimento laterale. L'ordine della discussione è stato progettato per essere il più fluido possibile in modo da rendere chiaro il tutto.

Capitolo 2

Nozioni preliminari

In questo capitolo, dopo un breve richiamo ai concetti fondamentali della teoria dei grafi utilizzati nei capitoli seguenti nella trattazione delle varie euristiche, si farà un breve accenno alle classi di complessità computazionale e di approssimazione. Ciò permetterà di inquadrare meglio la difficoltà del problema oggetto di studio in questa tesi. In particolare, nella sezione 2.2.7 si riporterà la prova di NP-completezza del k-MST e nella sezione 2.2.8 i risultati noti per questo problema nell'ambito delle classi di approssimazione.

2.1 Teoria dei grafi

Un grafo semplice G è definito come $G = (V, E)$, dove V è un insieme astratto di elementi chiamati *vertici* del grafo ed E è un sottinsieme della famiglia delle coppie di vertici in V, cioè $E \subseteq \binom{V}{2}$. Gli elementi dell'insieme E prendono il nome di *archi*. Due vertici u e v si dicono adiacenti se la coppia u, v appartiene a V. Se $e = \{u, v\}$, i vertici u, v si chiamano gli estremi

dell'arco e. Un grafo G si dice completo se $E = \binom{V}{2}$. Un cammino tra i vertici u, v è una sequenza alternata di archi e vertici che inizia con u e termina con v, cioè $u, e_1, x_1, e_2, x_2, ..., e_h, v$, dove gli archi e_i sono tutti diversi e per ogni arco e_i si ha che $e_i = \{x_{i-1}, x_i\}$. Due vertici u, v sono connessi se tra loro esiste almeno un cammino. Un grafo si dice connesso se ogni coppia di vertici u, v in V, è connessa. La relazione di connessione tra i vertici del grafo è una relazione di equivalenza. Le classi di equivalenza di tale relazione sono chiamate le *componenti connesse* del grafo. Un cammino tra due vertici u, v con $u = v$ è chiamato ciclo. Un grafo privo di cicli si chiama *foresta*, un grafo privo di cicli connesso si chiama *albero*. Un albero viene definito *di copertura* se connette tutti i vertici appartenenti a V.

Un grafo pesato è un grafo con associata una funzione di peso $c : E \to \mathbb{R}^+$. Il costo di un grafo è dato dal valore $t = \sum_{e \in E} c(e)$. Un albero di copertura è minimo se il suo costo $c(T)$ è minore o uguale del costo $c(T')$ di ogni altro altro albero di copertura.

Un grafo nel piano euclideo o, più in generale, in uno spazio è un grafo dove i vertici sono punti dello spazio. Di solito quando si parla di grafi immersi in uno spazio si presuppone che si tratti di grafi pesati dove la funzione costo viene indicata dalla funzione distanza della metrica che caratterizza lo spazio.

2.1.1 Alberi di Steiner

Una puntualizzazione va fatta a riguardo di un particolare tipo di grafi, gli alberi di Steiner. Un albero di Steiner è un albero che ricopre un sottoinsieme dei vertici chiamati *terminali* sfruttando, eventualmente, i vertici non

terminali. Dato un insieme di punti S, un albero di Steiner nel piano euclideo ricoprente S è un albero che ricopre tutti i vertici in S passando, eventualmente, per alcuni punti non appartenenti a S. Tornerà utile il concetto di albero di Steiner minimo in più punti di questo lavoro. Dato un insieme S di punti nel piano euclideo definiamo il *rapporto di Steiner* il rapporto tra il costo del minimo albero di copertura su S e il costo del minimo albero di Steiner su S. Una proprietà importante sul rapporto di Steiner nella metrica euclidea è che questo valore è sempre minore o uguale a $2/\sqrt{3}$. Questo risultato, congetturato da Gilbert e Pollack nel 1968, è stato dimostrato formalmente nel 1992 da Du e Hwang, [28].

2.2 Teoria della complessità

Il problema del k-MST è stato analizzato a lungo da parte degli esperti del settore a causa della sua notevole difficoltà. Pensiamo infatti a risolvere il problema nella maniera più brutale possibile. Avendo un algoritmo veloce per trovare un albero di copertura di costo minimo su un grafo, basterebbe identificare l'insieme di cardinalità k dei vertici di una soluzione ottima. Se analizzassimo tutti i possibili sottinsiemi di k elementi sugli n vertici ci troveremmo a esaminare $\binom{n}{k}$ sottoinsiemi, vale a dire $O(n^k)$ sottoinsiemi. Per piccoli valori di k un simile approccio può essere ancora accettabile ma al crescere di k il costo computazionale di un simile algoritmo di forza bruta diventa insostenibile ed in ogni caso per k variabile l'algoritmo non è polinomiale. Gli algoritmi polinomiali sono gli unici per i quali eventuali avanzamenti tecnologici comportano la possibilità di trattare, negli stessi tempi,

istanze di dimensioni significativamente maggiori. La crescita di funzioni non polinomiali (ad esempio esponenziali) è così veloce che i valori diventano rapidamente troppo grandi. Assumendo che sia utilizzata una macchina che esegue un passo di computazione in 10 nanosecondi (10^{-8} secondi), otteniamo per algoritmi con diverse complessità temporali i tempi di esecuzione in Tabella 2.2 (si noti che l'età dell'universo viene stimata tra 10^{10} e $2 \cdot 10^{10}$ anni).

Compl.	$n = 2$	$n = 5$	$n = 10$	$n = 100$	$n = 1000$
n	20 ns	50 ns	100 ns	1 μs	10 μs
n^2	40 ns	250 ns	100 ns	1 μs	10 μs
n^3	80 ns	1.25 μs	10 μs	10 ms	10 sec
n^6	640 ns	150 μs	10 ms	$\approx$ 3 ore	$\approx$ 300 anni
2^n	40 ns	320 ns	≈ 10 μs	$\approx 10^{14}$ anni	$\approx 10^{284}$ anni
2^{n^2}	160 ns	0.3 sec	$\approx 10^{14}$ anni	$\approx 10^{3000}$ anni	$\approx 10^{300000}$ anni

Tabella 2.1: I tempi di calcolo di algoritmi con diverse complessità computazionale. Per n si intende la grandezza dell'*input*.

Per questo motivo si è cercato di trovare una soluzione più veloce e più intelligente per il problema. In un certo senso si è cercato di analizzare quanto il problema fosse difficile e per spiegare tale difficoltà facciamo ricorso ad alcuni cenni della teoria della complessità.

Uno degli obiettivi fondamentali della complessità computazionale è di classificare i problemi computazionali decidibili, dividendoli in *classi*, definite sulla base della quantità di risorse a disposizione. Le risorse tipicamente considerate sono lo spazio (quantità di memoria) e il tempo (durata del pro-

cesso di calcolo). Ci si chiede in particolare quali siano le proprietà che rendono certi problemi computazionali facili (risolvibili efficientemente) ed altri difficili (intrattabili). Un'istanza importante di questo problema generale è la famosa questione del confronto tra le classi **P** (la classe dei problemi trattabili) ed **NP** (una classe che contiene molti problemi tra cui la versione decisionale del k-MST che probabilmente sono intrattabili), uno dei problemi più rilevanti non solo dell'informatica teorica, ma anche della stessa matematica contemporanea. Basti pensare che su un tale problema si fondano i principi del più resistente sistema di cifratura esistente al mondo, utilizzato in molteplici ambiti, dall'economico al militare. Inoltre lo stesso problema è in cima all'elenco dei quesiti posti all'inizio del millennio dal Clay Mathematics Institute di Cambridge (Massachusetts - USA), il quale per celebrare la nuova era della matematica ripropone una sfida analoga a quella che lanciò Hilbert un secolo prima, premiando con un milione di dollari chi riesca a risolvere uno dei problemi ancora irrisolti.

2.2.1 Problemi decisionali

Problemi per i quali la soluzione equivale ad una scelta tra due valori alternativi, rappresentati da "si" e "no", sono l'oggetto principale di studio in teoria della complessità. Questi problemi prendono il nome di *problemi decisionali.*

Definizione 2.1. *Un* problema decisionale Π *consiste in un insieme* D_Π *di istanze ed in un sottoinsieme* S_Π *di istanze* positive. *Un istanza appartiene a* S_Π *se e soltatanto se la soluzione del problema, relativamente a quella istanza, è data dal valore "si".*

La centralità dei problemi decisionali nasce dal fatto che hanno una cotroparte formale, i *linguaggi* o gli *insiemi*, che si presta bene ad un'indagine matematica. Risulta quindi naturale far corrispondere ad un problema decisionale il problema dell'appartenenza di una parola ad un dato linguaggio, codificando ogni istanza del problema tramite una parola tale che l'istanza ha risposta "si" se e solo se la parola appartiene al linguaggio corrispondente.

Definizione 2.2. *Dato un insieme finito di simboli Σ, detto alfabeto, indichiamo con Σ^* l'insieme di tutte le stringhe finite di simboli di Σ. Se L è un sottoinsieme di Σ^*, si dice che L è un* linguaggio *sull'alfabeto Σ.*

Allo scopo di confrontare problemi emergenti da aree diverse e descritti tramite linguaggi diversi è opportuno "tradurre" le parole costruite su un alfabeto arbitrario in parole costruite su un alfabeto prefissato. Un problema decisionale può quindi essere visto come il problema di riconoscere il linguaggio definito dall'insieme delle descrizioni delle istanze positive oppure, in modo equivalente, come il problema di decidere l'appartenenza di una stringa all'insieme i cui elementi descrivono le istanze positive.

2.2.2 Notazione asintotica

Faremo ampio uso in seguito di particolari classi di funzioni indicate con $O(f(n))$ e $\Omega(f(n))$. Queste notazioni definiscono la complessità asintotica, assunta da un problema quando le dimensioni del suo input tendono a valori molto grandi. Siano f e g due funzioni definite sui naturali e a valori reali,

$$f, g : \mathbb{N} \to \mathbb{R}.$$

Si dice che

- f è di ordine *non superiore* a g, e si indica con $f(n) = O(g(n))$, se esiste un intero n_0 tale che, per ogni $n \geq n_0$, vale $|f(n)| < \alpha|g(n)|$, dove α è una costante positiva indipendente da n;

- f è di ordine *non inferiore* a g, e si indica con $f(n) = \Omega(g(n))$, se g è di ordine non superiore ad f, ovvero se $g(n) = O(f(n))$;

- f è di ordine *inferiore* a g, e si indica con $f(n) = o(g(n))$, se

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0;$$

- f è dello stesso ordine di g, e si indica con $f(n) = \Theta(g(n))$, se f è sia di ordine non superiore sia di ordine non inferiore a g.

2.2.3 Misure di complessità

Siamo ora in grado di formalizzare la nozione di *complessità di un algoritmo* che ci servirà per valutare le euristiche trattate in questo lavoro. La teoria della complessità si è sviluppata considerando inizialmente due fondamentali grandezze: il tempo e lo spazio. Le misure corrispondenti chiamate *TEMPO* e *SPAZIO* fanno riferimento al modello di calcolo delle macchine di Turing[1]. La misura *TEMPO* rappresenta il numero di istruzioni eseguite da una macchina di Turing deterministica durante una computazione. La

[1]La macchina di Turing è un modello di calcolo introdotto dal logico matematico Alan Turing per definire alcune misure di complessità dinamica. Si tratta di un'astrazione degli attuali calcolatori in cui la memoria è schematizzata da un nastro diviso in celle controllata da una testina. Poiché questo argomento esula dal nostro scopo rimandiamo per approfondimenti a [26].

misura *SPAZIO* indica invece il numero di celle di nastro utilizzate da una macchina di Turing deterministica durante una computazione.

2.2.4 Classi di complessità

Un concetto chiave nella teoria della complessità è quello di *classe di complessità*, ossia di insieme di tutti i problemi risolvibili su un dato modello di calcolo con una ben precisa limitazione nell'uso di una o più risorse. Più precisamente, per definire una classe di complessità, dobbiamo specificare un certo numero di parametri:

- il modello di calcolo di riferimento;
- una modalità con cui si eseguono le computazioni (il calcolo può ad esempio procedere in modo deterministico, non deterministico oppure probabilistico);
- una risorsa *costosa* disponibile entro un certo limite;
- una funzione $f : \mathbb{N} \to \mathbb{N}$ che esprime tale limite.

Definizione 2.3. *Sia f una funzione. La classe di complessità TEMPO(f) è l'insieme dei linguaggi riconosiuti da una MdT*[2] *che consuma, per ogni input x, al più $f(|x|)$ unità di tempo.*

La discussione che abbiamo svolto in precedenza a proposito del criterio di efficienza polinomiale motiva l'importanza di caratterizzare i problemi

[2]Macchina di Turing.

risolvibili in tempo polinomiale, ossia la classe dei problemi trattabili. La nozione di trattabilità vista come sinonimo di efficienza polinomiale è catturata dalla classe di complessità **P.** La classe **P** può essere definita come

$$\bigcup_{k=0}^{\infty} TEMPO(n^k).$$

La classe **P** è perciò definta in termini di una famiglia di funzioni costo (ossia dalle funzioni n^k, per $k = 0, 1, ...$) piuttosto che da una singola funzione.

2.2.5 Non determinismo e classe NP

Il seguente problema è centrale nella complessità computazionale, ma la cui importanza va ben al di là dei confini di questa disciplina:

> É più "difficile" calcolare la soluzione di un problema oppure verificare la correttezza di un'ipotetica soluzione?

Il confronto tra la difficoltà del calcolo e la difficoltà della verifica non è estraneo all'esperienza quotidiana. Quanti studenti e ricercatori hanno cercato invano di risolvere un esercizio o di dimostrare un teorema, per poi stupirsi della semplicità della soluzione (trovata da altri) dopo averla letta da qualche parte. Queste osservazioni ci inducono a pensare che la verifica sia più "facile" del calcolo. Ma il dimostrare questo coinciderebbe con il risolvere il problema aperto di fondamentale importanza del confronto tra le classi **P** e **NP**.

Una varietà di problemi computazionali presentano la seguente caratteristica: pur essendo presumibilmente intrattabili, ammettono una procedura di verifica efficiente. Questa proprietà è espressa dalla classe di complessità

NP che può essere infatti definita informalmente come la classe dei problemi decisionali per i quali ipotetiche soluzioni possono essere verificate in tempo polinomiale. Per dare una definizione formale della classe **NP** è necessario introdurre il concetto di *non determinismo.*

Definizione 2.4. *Una computazione si dice* non deterministica *quando consiste in una successione di passi discreti, ciascuno dei quali è una transizione da uno stato ad un insieme di stati (piuttosto che a un singolo stato).*

La MdT non deterministica ci consente di definire formalmente classi di complessità non determinstche e in particolare la classe **NP .**

Definizione 2.5. *La classe di complessità NTEMPO(f) è l'insieme dei linguaggi decisi da una MdTN*[3] *che consuma, su ogni input x che deve essere accettato, al più $f(|x|)$ unità di tempo.*

La classe **NP** può essere quindi definita come

$$\bigcup_{k=0}^{\infty} NTEMPO(n^k).$$

Il nostro problema k-MST come molti altri tra i quali TSP[4], FATTORIZZAZIONE[5] e il capostipite SAT[6] appartengono alla classe **NP** e sono candidati a non appartenere alla classe **P**, a sostegno della congettura che la classe **NP** contenga strettamente la classe **P**, ossia che esistano problemi risolvibili

[3]Macchina di Turing non deterministica.

[4]Il problema del commesso viaggiatore.

[5]Il problema della ricerca della fattorizzazione di un intero. Da non confondere con il problema del test di primalità che è di gran lunga più semplice; pochi anni fa è stata dimostrata la sua appartenenza alla classe **P**, [27].

[6]La soddisfacibiltà delle formule logiche.

in tempo polinomiale non deterministico che non sono risolvibili in tempo polinomiale deterministico, Figura 2.1.

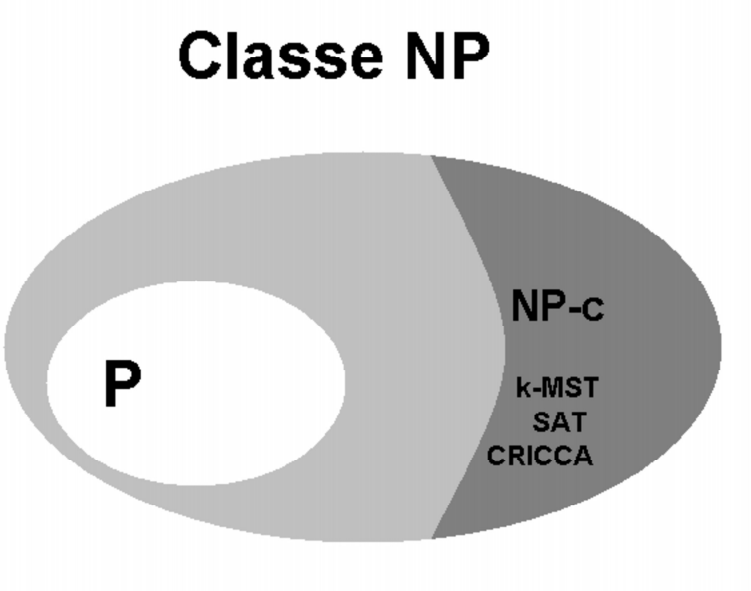

Figura 2.1: Le classi di complessità computazionale.

Utilizzando uno strumento, la riduzione di Karp in tempo polinomiale, è possibile orientare lo studio delle relazioni tra le classi **P** e **NP** alla ricerca dei problemi più "difficili" all'interno della classe **NP**, e dunque problemi che sembrano proporsi come ragionevoli candidati a non appartenere alla classe **P**. In particolare, nella classe **NP** è possibile individuare una classe di problemi, i problemi NP-completi, la cui complessità individuale è rappresentativa della complessità dell'intera classe e la cui appartenenza a **P** implicherebbe l'uguaglianza $\mathbf{P} = \mathbf{NP}$.

2.2.6 I problemi di ottimizzazione

I problemi decisionali hanno un ruolo molto importante nella teoria della complessità in quanto costituiscono gli elementi di base di altri problemi più interessanti e complessi, chiamati *problemi di ottimizzazione*, provenienti da campi diversi quali l'algebra, la teoria dei numeri, la teoria dei grafi, la linguistica computazionale, la teoria degli automi e la teoria dei giochi.

In termini intuitivi i problemi di ottimizzazione danno per scontato che l'istanza ammette una soluzione ed hanno per obbiettivo quello di individuare la soluzione "ottima" ossia la soluzione avente costo ottimo rispetto ad una data misura prefissata. Quando ci occupiamo di trovare il minimo albero di copertura noi vogliamo avere un soluzione *ammissibile*, ovvero un albero di copertura, che minimizzi la funzione costo definita sul dominio degli archi. Come è ovvio la complessità di un problema di ottimizzazione non può essere inferiore a quella del problema decisionale corrispondente: in effetti saper calcolare l'ottimo ci consente di verificare se esiste una soluzione avente una misura prefissata. Esistono molti problemi di ottimizzazione a cui tutti i problemi **NP** si riconducono in tempo polinomiale, ma per i quali non sono note dimostrazioni di appartenenza alla classe **NP**. Problemi con queste caratteristiche sono detti NP-ardui.

2.2.7 La prova di NP-completezza

Il problema k-MST è divenuto ancor più interessante quando Ravi ed altri in [3] ne hanno dimostrato l'appartenenza alla classe dei problemi NP-completi.

Teorema 2.1. *La versione decisionale del problema k-MST è NP-completa.*

Dimostrazione. Per provare l'NP-completezza del problema dimostreremo che la sua versione non decisionale è NP-ardua. Per far ciò proveremo che il problema della ricerca dell'albero minimo di Steiner è riducibile polinomialmente al problema k-MST. La dimostrazione che il problema sugli alberi di Steiner è NP-arduo è stata presa da [5]. In un istanza del problema dell'albero di Steiner abbiamo un grafo semplice G, un sottoinsieme dei vertici del grafo R (chiamati *terminali*) e un intero positivo M, e la domanda che ci poniamo è se esiste un albero ricoprente R e contenente al più M archi. Trasformiamo, ora, l'input in un istanza G', k, M del problema k-MST nella seguente maniera: dato un valore $X = |V(G)| - |R|$ e connettiamo ogni terminale di G ad un diverso cammino costituito da X nuovi vertici e da X nuovi archi a cui diamo peso 0. Assegnamo a ogni arco in G il peso 1 e per ogni coppia di vertici non adiacenti inseriamo un nuovo arco di peso ∞ (ricordiamo che il grafo deve essere completo).

Nella Figura 2.2 possiamo apprezzare la trasformazione del grafo G. Poi prendiamo k pari a $|R| \cdot (X+1)$. E ora ci chiediamo se esiste in G' un albero di peso al più M ed in grado di ricoprire k vertici. Se esiste un albero di Steiner su G ricoprente l'insieme R e contenente al più M archi, prendendo tutti i cammini che hanno un costo 0 e l'albero di Steiner otterremmo un k-MST di peso al più M in G'. Viceversa, per la nostra scelta di k e X, ogni k-MST in G' deve contenere almeno un nodo per il cammino corrispondente ad ogni terminale in R, perché altrimenti se esistesse un cammino completamente esterno all'albero, sarebbe in grado di collegare al più $k-1$ nodi. Quindi da ogni k-MST è facile ottenere, dopo una potatura, un albero di Steiner per R in G. Questo completa la riduzione. □

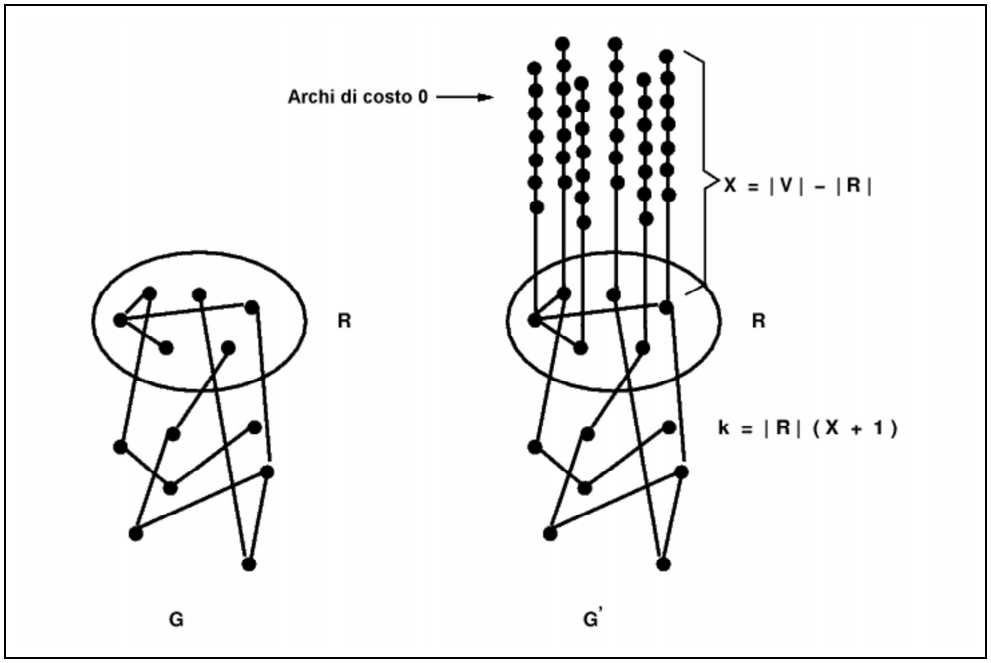

Figura 2.2: La riduzione dal problema NP-arduo di Steiner al k-MST.

L'estensione della dimostrazione al caso dei grafi planari e i punti nel piano segue in modo analogo dalla NP-completezza del problema della ricerca dell'albero di Steiner nei corrispondenti casi speciali. Dato un *embedding* planare di G possiamo creare un *embedding* di G' poiché abbiamo inserito in più solamente i cammini.

La prova di NP-completezza vale anche quando tutti i pesi degli archi sono presi nell'insieme $1, 2, 3$. La riduzione per questo caso è simile alla precedente. Senza perdere di generalità assumiamo che nell'istanza del problema dell'albero di Steiner, G sia connesso e $M \leq |V| - 1$. Posto $X = |V(G)| - |R|$ come prima, connettiamo ogni terminale di G ad un differente insieme di X vertici con archi di peso 1. Poi assegnamo ad ogni arco di G il peso 2 e tutti

gli altri archi il peso 3. Poniamo $k = |R| \cdot X + M + 1$ e il limite al costo del k-MST a $|R| \cdot X + 2M$. Se esiste un albero di Steiner in G ricoprente l'isieme R e contenente al più M archi, è facile costruire un k-MST di peso al più $|R| \cdot X + 2M$ in G'. Questo lo si può ottenere connettendo tutti i nuovi vertici aggiunti all'albero di Steiner usando gli archi di peso uno ed aggiungendo altri vertici (ricordiamo che il grafo è connesso e $M \leq |V| - 1$) di peso 2 fino ad avere $|R| \cdot X + M + 1$ vertici. Se esiste un k-MST di costo al più $|R| \cdot X + 2M$ in G' osserviamo che il k-MST non può contenere archi di peso 3 perché questo ha esattamente $k - 1 = |R| \cdot X + M$ archi e se contenesse un arco di peso 3 allora dovrebbe contenere almeno $|R| \cdot X + 1$ archi di peso 1, in contrasto con il fatto che ci sono solo $|R| \cdot X$ archi di peso 1 in G'. Inoltre il k-MST deve ricoprire R, e siccome ci sono al più M archi di peso 2, deve esistere su G un albero di Steiner che ricopre R contenente al più M archi.

Nel caso in cui il grafo sia completo e abbia archi con solo due pesi distinti, la soluzione del problema k-MST può essere trovata facilmente in tempo polinomiale. L'idea di base è la seguente: siano w_1 e w_2 i due pesi degli archi, con $w_1 < w_2$. Costruisci un sottografo G' di G contenente tutti gli archi di peso w_1. Scegli il minimo numero, chiamato r, di componenti connesse di G_1 per avere un totale di k nodi (eliminandone qualcuno se necessario). Costruisci un albero di copertura per ogni componente scelta e connetti gli alberi insieme in un unico albero mediante l'aggiunta di esattamente $r - 1$ archi di peso w_2. Si verifica, banalmente, che la soluzione trovata è ottima.

2.2.8 Approssimabilità

Per un gran numero di problemi NP-ardui, come il k-MST il k-EuMST, la ricerca della soluzione ottima può essere in pratica sostituita dalla ricerca di soluzioni *quasi*-ottimali (approssimazioni dell'ottimo), nella speranza che queste richiedano un costo computazionale sostanzialmente inferiore.

L'attività di ricerca sull'approssimazione dei problemi di ottimizzazione NP-ardui si è sviluppata proprio per l'esigenza molto concreta di capire se sia possibile determinare approssimazioni accurate in tempi ragionevoli. Sfortunatamente, per varie classi di problemi di ottimizzazione, anche calcolare una soluzione approssimata può risultare estremamente costoso. Più precisamente, per i problemi di ottimizzazione NP-ardui, emergono notevoli differenze di comportamento nei confronti dell'approssimabilità: lo spettro è ampio, si va da problemi facili da approssimare con cura, ad altri per cui anche la ricerca di approssimazioni poco accurate è un problema NP-arduo. Per questo motivo il tentativo di capire in quali casi e per mezzo di quali metodi sia possibile progettare algoritmi di approssimazione polinomiali per i problemi di ottimizzazione NP-ardui è molto rilevante da un punto di vista sia teorico che pratico.

I problemi di ottimizzazione per i quali esiste un analogo problema decisionale NP-completo, costituiscono la classe dei problemi **NPO**. Data un istanza di un problema in **NPO**, si chiama *algoritmo approssimante* un algoritmo tale che il rapporto tra il costo della soluzione prodotta da questo e il costo della soluzione ottima è limitato da una funzione f definita sulla grandezza dell'input. La funzione f prende il nome di *fattore di approssimazione* dell'algoritmo.

Il fattore di approssimazione è una misura molto utile della bontà di approssimazione di un algoritmo. Si pensi a problemi che hanno algoritmi approssimanti con fattore di approssimazione costante. Problemi di questo genere posseggono la proprietà di essere facilmente approssimabili. Dato un problema π, un algoritmo che approssima la soluzione di π con un fattore di approssimazione costante ρ viene chiamato ρ-approssimante. In base al tasso di approssimazione raggiunto dagli algoritmi per i problemi in **NPO**, in letteratura si distinguono varie classi di approssimabilità. L'insieme dei problemi NP-ardui per i quali esiste un algoritmo ρ-approssimante è catturato dalla classe **APX**. Quando fece la sua prima comparsa, nel 1993, il problema k-MST non faceva parte di questa classe. I primi algoritmi per questo problema raggiungevano un fattore di approssimazione $O(\sqrt{k})$. Blum et al. in [14], dimostrando l'esistenza di un algoritmo approssimante con un fattore costante, testimoniarono l'appartenenza del problema k-MST alla classe **APX**.

Non per tutti i problemi è facile trovare approssimazioni costanti, anzi esiste un solido gruppo di quesiti che sono altamente ostici da approssimare. Un esempio è il problema TSP, il commesso viaggiatore. Viene spontaneo chiedersi quanto un fattore di approssimazione possa essere piccolo. Ebbene esistono problemi NP-ardui che godono della proprietà di poter essere approssimati con un fattore di approssimazione pari a $1 + \epsilon$, con il valore di ϵ piccolo a piacere. Ovviamente per i problemi NP-ardui non si può avere un fattore di approssimazione di 1, salvo il caso in cui **P**=**NP**. Va osservato però che per approssimare in tale modo un problema si fa ricorso ad algoritmi che hanno una complessità polinomiale inversamente proporzionale al

valore di ϵ. Questo ad indicare che il maggiore tasso di approssimazione si paga con un costo in termini di tempo. Questa metodologia di approssimazione si chiama *schema di approssimazione in tempo polinomiale*. La classe di approssimabilità che comprende tutti i problemi in **NPO** che hanno uno schema di approssimazione in tempo polinomiale si chiama **PTAS**. Fra le varie classi citate valgono le seguenti inclusioni:

$$\mathbf{PTAS} \subseteq \mathbf{APX} \subseteq \mathbf{NPO}.$$

Uno dei maggiori problemi aperti della teoria dell'approssimazione è quello di stabilire se le inclusioni tra le classi **NPO**, **APX** e **PTAS** siano proprie. Una chiara evidenza della difficoltà di tale questione è data dal fatto che le inclusioni sono proprie se e solo se **P**$\neq$**NP**. Un'intuizione del perchè questo accada è la seguente: per alcuni problemi **NPO**, tra cui TSP, è possibile dimostrare direttamente che da un algoritmo ρ-approssimante, o da uno schema di approssimazione polinomiale, è possibile derivare un algoritmo di costo polinomiale che ne risolve la versione decisionale. Dato che la versione decisionale dei problemi **NPO** è un problema NP-completo, questo implicherebbe **P**=**NP**.

Per la versione euclidea del nostro problema, il k-EuMST è stato prodotto uno schema di approssimazione polinomiale, sia da Arora[25] che da Mitchell[24]. Dal punto di vista dell'approssimazione, il problema k-MST, nella sua formulazione generale, è risultato più difficile del caso euclideo. Viggo Kann ha mostrato che per il caso generale non è possibile trovare uno schema di approssimazione polinomiale, collocando il k-MST nella classe dei problemi APX-completi. La Figura 2.3 riporta un quadro di insieme in merito a quanto detto sugli algoritmi approssimanti.

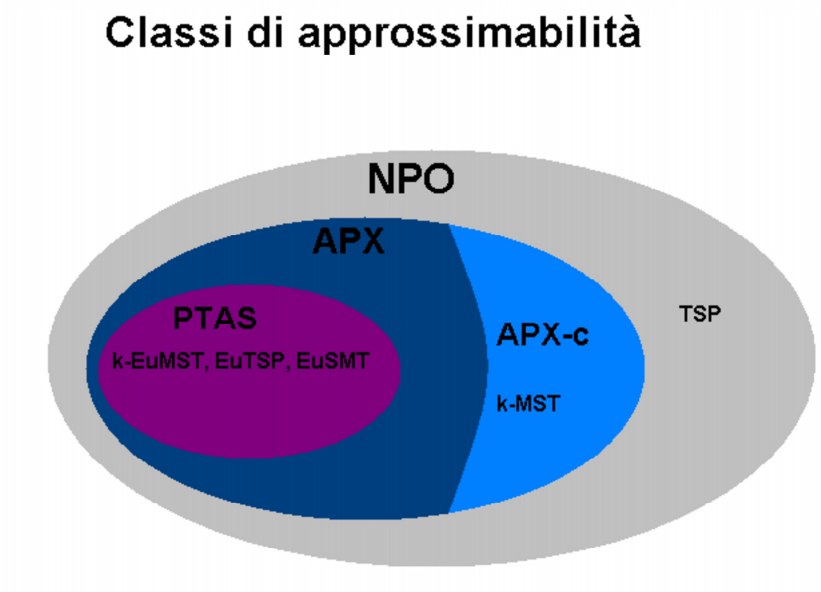

Figura 2.3: Le classi di approssimabilità ed alcuni problemi di ottimizzazione.

Capitolo 3

Algoritmo k-Kruskal

L'idea alla base di questa euristica è quella di prendere l'algoritmo di Kruskal e modificarlo in modo da avere come output un albero che ricopra solo k degli n punti del piano. Lo studio di un euristica molto semplice segue dalla nostra scelta di selezionare, tra i vari algoritmi approssimanti esistenti che risolvono il problema k-MST, quelli che abbiano una complessità di tempo molto bassa e che siano adatti quindi a trattare istanze molto grandi del problema, pena il rapporto di approssimazione. Questa euristica presenta un rapporto di approssimazione inferiore a k ed ha una complessità computazionale $O(n \log n)$. Una complessità così bassa è ottenuta grazie ad un'implementazione accurata che vedremo nelle sezioni seguenti.

3.1 L'algoritmo

Di seguito viene esposto l'algoritmo in pseudocodice:

1. $S = \emptyset$

2. Ordina in maniera non decrescente gli archi del grafo in base alla lunghezza
3. Parti dal primo arco dell'ordinamento e
4. Ripeti finché non ottieni almeno una componente connessa di k punti
   (a) Se l'arco e non crea un ciclo con gli archi contenuti in S allora $S = S \cup \{e\}$
   (b) Passa all'arco successivo nell'ordinamento
5. Sia C una componente connessa che ricopre almeno k punti
6. Ripeti finché la componente C ricopre più di k punti
   (a) Sia e un arco in C che collega una foglia (un punto che ha un solo arco incidente). $C = C - \{e\}$
7. Restituisci la componente connessa C come risultato

3.2 Il rapporto di approssimazione

In questa sezione analizziamo il rendimento dell'algoritmo.

Lemma 3.1. *Per ogni soluzione ottima esiste un arco e tale che costo*$(e) \geq$ *costo*(a)*, per ogni arco a appartenente alla soluzione prodotta dall'algoritmo* k-Kruskal.

Dimostrazione. Al penultimo passo del primo ciclo dell'algortimo k-Kruskal tutte le componenti connesse create hanno dimensione minore di k quindi

deve esistere, per ogni soluzione ottima, almeno un arco $e = (u, v)$ con u e v in componenti distinte. Quest'arco, ovviamente, non crea ciclo con gli archi già presi e l'unica ragione per cui non è presente nella soluzione costruita dall'algoritmo, SOL, è che non è stato ancora considerato. Poiché gli archi sono considerati in ordine nondecrescente, il suo costo deve essere superiore al costo di tutti gli archi finora considerati e quindi anche di quelli presi dall'algoritmo. Nell'ultimo passo k-Kruskal sceglierà l'arco e o un arco di peso non superiore ad e. La componente connessa di dimensione almeno k che si formerà conterrà quindi un albero i cui archi hanno tutti costo non superiore ad e e quindi l'albero di k vertici che si otterrà da questa componente non potrà avere archi il cui costo supera il costo di e. □

Teorema 3.1. *L'algoritmo k-Kruskal approssima la soluzione del k-MST con un fattore k-1.*

Dimostrazione. Sia SOL la soluzione prodotta dall'algoritmo k-Kruskal e T^* l'albero indotto da SOL. Per il lemma 3.1 si ha che

$$\forall a \in SOL, costo(a) \leq \max_{e \in T^*} costo(e) \leq costo(T^*)$$

L'albero trovato dall'algoritmo copre esattamente k nodi ed è costituito da $k-1$ archi quindi

$$costo(SOL) \leq (k-1)costo(T^*)$$

che porta a

$$\frac{costo(SOL)}{costo(T^*)} \leq k-1$$

□

Non è difficile provare, inoltre, che il fattore di approssimazione k non può essere migliorato. Ad esempio, supponiamo di avere $2k-1$ punti disposti su una retta, i primi $k-2$ siano a distanza ϵ, il $k-1$-esimo a disistanza $a+\epsilon$ dal successivo e ciascuno degli altri a distanza a dal successivo, con $\epsilon < a$. É facile convincersi che k-Kruskal produrrà come soluzione la catena composta dagli ultimi k nodi che ha costo $(k-1)a$ mentre se prendessimo i primi $k-1$ archi avremmo un costo pari a $(k-2)\epsilon + a + \epsilon$. Prendendo $\epsilon = 1/(k-1)$ ed $a = k$, il fattore di approssimazione è dato dal rapporto $(k^2-k)/(1+k)$ che è maggiore di $k-2$. Esplicito i passaggi per maggior chiarezza.

$$\frac{k^2-k}{1+k} = \frac{(k-2)(k^2-k)}{(k-2)(1+k)} = (k-2)\frac{k^2-k}{k^2-k-2}$$

Poiché

$$\frac{k^2-k}{k^2-k-2} > 1$$

si ha

$$(k-2)\frac{k^2-k}{k^2-k-2} > k-2$$

e quindi

$$\frac{k^2-k}{1+k} > k-2.$$

3.3 L'implementazione

Il secondo passo dell'algoritmo consiste nell'ordinamento degli archi da analizzare in ordine non decrescente rispetto alla loro lunghezza. Per fare questo si è utilizzato l'algoritmo di merge-sort che consente di ordinare n valori in un tempo $O(n \log n)$. Nel nostro caso, il grafo è definito da un insieme di punti nel piano euclideo e gli archi sono tutte le possibili coppie di nodi. Si

ha quindi $m = \binom{n}{2} = O(n^2)$ e la complessità dell'algoritmo per il solo punto 2 è $O(n^2 \log n)$.

Per ottimizzare il calcolo abbiamo fatto uso di uno strumento preso in prestito dalla geometria computazionale, la triangolazione di Delaunay, vedere [9]. Cerchiamo di capire come può essere utile questo oggetto nell'implementazione del nostro algoritmo. Iniziamo col dire che dato un insieme di punti nel piano euclideo, una triangolazione è un insieme di segmenti che congiungono i punti e che dividono il piano (salvo la regione esterna al poligono costruito) in aree di forma triangolare.

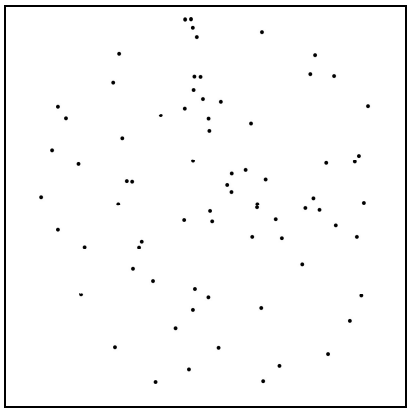

Figura 3.1: Un insieme di 75 punti

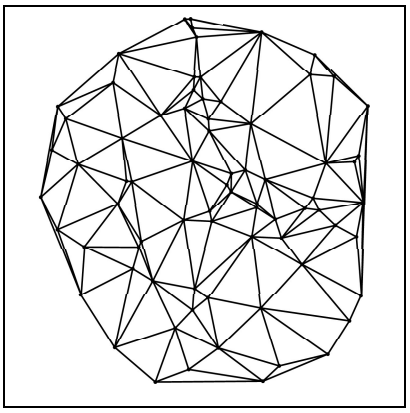

Figura 3.2: La triangolazione di Delaunay

Una triangolazione è una divisione di una superficie in un insieme di triangoli, con la proprietà che ogni lato di un triangolo è interamente condiviso da due triangoli adiacenti. Le triangolazioni sono usate in grafica digitale, si pensi al *surface fitting*, sono utili nell'approssimazione numerica del calcolo integrale e vengono sfruttate da applicativi come CAD e GIS. Nella Figura 3.1 vediamo un insieme di punti disposti a caso, uniformemente, nel piano

e, nella Figura 3.2 la loro triangolazione di Delaunay. Una triangolazione di Delaunay ha un'importante proprietà: se abbiamo dei punti nel piano euclideo e li consideriamo vertici di un grafo completo immerso nella metrica euclidea, l'albero di copertura minimo relativo a questo grafo è un sottoinsieme della triangolazione di Delaunay dei vertici. Questa proprietà ci aiuta non poco, visto che il nostro ambito di ricerca degli archi per l'algoritmo di Kruskal si riduce ai soli archi della triangolazione e non a tutti quelli del grafo. Inoltre, il vantaggio primario lo otteniamo dal fatto che la triangolazione possiede $O(n)$ archi, dove n sono i vertici del grafo.

Grazie a questa analisi abbiamo in tempo $O(n \log n)$, con n vertici, l'ordinamento per lunghezza non decrescente degli archi della triangolazione. Quello che resta da fare è selezionarli, seguendo l'ordine, uno per uno, evitando di creare cicli, e fermandoci non appena si arriva ad avere una componente connessa di dimensione k. Per fare questo abbiamo trovato utile l'impiego di una struttura *ad hoc* che consentisse di tener bassi i tempi di calcolo. Quella che abbiamo utilizzato è una struttura dati per insiemi disgiunti chiamata *Union-Find*, molto nota, trattata ad esempio in [6].

Mantenendo in una tale struttura gli insiemi di vertici partizionati per componenti connesse, si realizza facilmente il test di appartenenza di un vertice ad una componente connessa, cosa che permette di verificare in modo efficiente se un arco che andiamo a inserire crea o no un ciclo nel grafo già costruito. L'implementazione qui usata è la più veloce che esista, quella che utilizza le euristiche di *unione per rango* e *compressione dei cammini*. Questa implementazione ha un tempo di esecuzione nel caso pessimo di $O(mf(m,n))$, dove m sono gli archi, n sono i vertici e $f(m,n)$ è l'inversa della funzione di

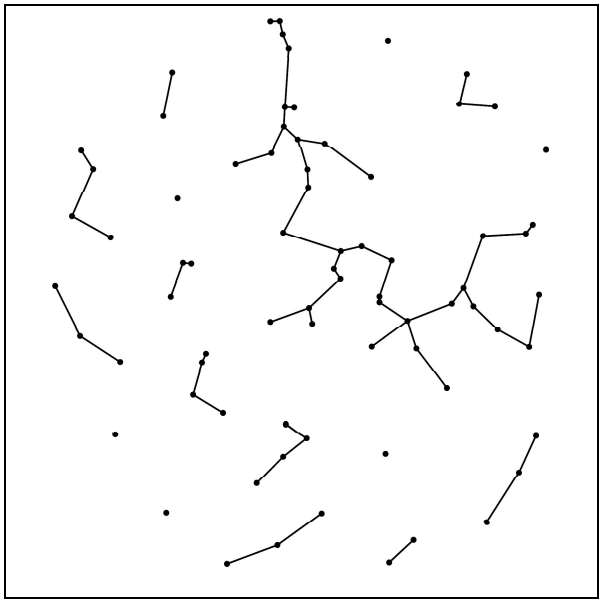

Figura 3.3: La foresta costruita dall'euristica k-Kruskal, con $n = 75$ e $k = 35$

Ackermann (che cresce molto lentamente). Per la dimostrazione del limite si rimanda a [6]. Nella Figura 3.3 si riporta la foresta prodotta dal k-Kruskal con $k = 35$ al termine del primo ciclo. Nella Figura 3.4 notiamo come, dalla foresta risultante dall'applicazione dell'algoritmo di Kruskal, la nostra euristica k-Kruskal seleziona l'albero che connette almeno k vertici. Gli altri alberi a questo punto, verranno scartati e non considerati più nei rimanenti passi del calcolo.

Una volta trovato l'albero di dimensione almeno k si passerà alla fase di potatura. Questa è necessaria perché è possibile che l'albero in questione abbia una dimensione superiore a k. Infatti, è facile notarlo supponendo che se l'algoritmo al passo i non aveva ancora prodotto componenti connesse di dimensione uguale o superiore a k, poteva comunque aver generato delle

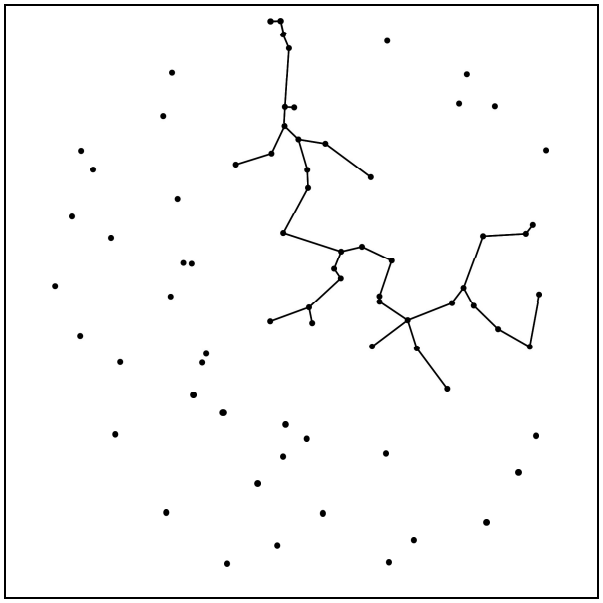

Figura 3.4: La componente connessa scelta dall'euristica k-Kruskal, $n = 75$ e $k = 35$

componente connesse di dimensione $k-1$. Al passo $i+1$ l'algoritmo potrebbe inserire un arco che connette due di queste componenti generandone una terza di dimensione $2(k-1) = 2k-2$ che, per k superiore a 2, indica una componente di dimensione superiore a k. La potatura viene effettuata visitando l'albero e limitandosi a prenderne i primi k vertici visitati con un costo computazionale $O(k)$. La Figura 3.5 riporta il risultato dell'algoritmo dopo la fase di potatura.

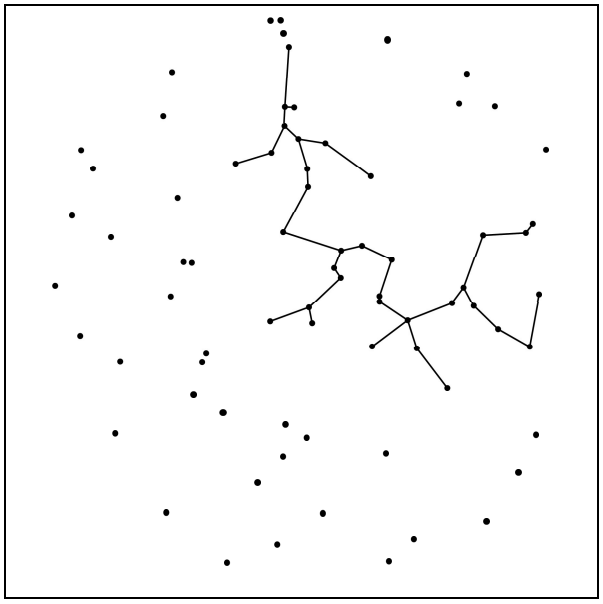

Figura 3.5: La soluzione prodotta dall'algoritmo k-Kruskal, $n = 75$ e $k = 35$

Capitolo 4

Algoritmo Minimo Quadrato

In questa sezione ci accingiamo ad analizzare un'ulteriore euristica sperimentale per il calcolo del minimo albero di copertura su k vertici nel piano. Come suggerisce il nome dell'algoritmo, la tecnica adottata consiste nell'incapsulare un insieme di k punti in una struttura convessa, la più piccola possibile. L'intuizione suggerisce che se i punti si trovino tutti addensati in una limitata regione del piano ci sarà un albero di copertura che li collega di costo *abbastanza* piccolo. Scendendo nei dettagli, la nostra euristica si preoccupa di cercare un quadrato con un lato, il più piccolo possibile, che contenga al suo interno almeno k punti.

Questi punti vengono poi collegati grazie ad un'applicazione dell'algoritmo k-Kruskal. Questa soluzione sfrutta in parte l'euristica precedentemente trattata in questo lavoro e in parte l'idea che si trova in [3], e ne utilizza anche una proprietà che garantisce un limite superiore alla lunghezza degli alberi con k punti che si trovano in un quadrato di lato δ. Questa euristica garantisce un fattore di approssimazione pari a $\sqrt{k}$, un valore inferiore

rispetto a $k^{1/4}$ ottenuto in [3]. La semplicità dell'euristica garantisce, però, un costo computazionale notevolmente più basso, pari a $O(n \log^3 n)$, quindi una tecnica molto veloce per essere testata su istanze con grande quantità di punti.

4.1 L'algoritmo

Di seguito viene esposto l'algoritmo in pseudocodice:

1. Per ogni punto p del grafo calcola il lato l_p del più piccolo quadrato centrato in p che contiene almeno k punti.
2. Sia p^* il punto cui corrisponde il quadrato di lato minimo, ovvero il valore l_p più basso.
3. Sia S l'insieme dei punti contenuti nel quadrato minimo centrato in p^*.
4. Applica k-Kruskal all'insieme S di punti.
5. Restituisci l'albero prodotto da k-Kruskal.

4.2 Il rapporto di approssimazione

Cerchiamo di capire quanto sia vicina la soluzione prodotta dall'algoritmo rispetto alla soluzione ottima del problema proposto. Per fare questo utilizzeremo un lemma che limita superiormente la lunghezza dell'albero calcolato.

Lemma 4.1. *La lunghezza di un albero di copertura minimo per qualsiasi insieme di k punti in un quadrato di lato σ è $O(\sigma\sqrt{k})$.*

Dimostrazione. Consideriamo una griglia nel quadrato dove ogni cella abbia un lato di $\sigma/\sqrt{k}$. Colleghiamo, con un segmento dritto, ogni punto in k al vertice più vicino nella griglia (incrocio di linee verticale e orizzontale). Consideriamo l'albero costituito da una linea verticale, tutte le linee orizontali nella griglia collegate alla line verticale, e tutti i segmenti che collegano i punti alle linee orizzontali. Le linee orizzontali e verticali dell'albero costruito hanno una lunghezza totale pari alla lunghezza delle linee σ per il numero $\sqrt{k}+1$, e quindi $O(\sigma\sqrt{k})$. I segmenti di aggancio dei punti hanno una lunghezza complessiva di $k \cdot O(\sigma/\sqrt{k}) = O(\sigma\sqrt{k})$, perché abbiamo k punti da connettere e ogni connessione può essere al più lunga come metà della diagonale di una cella di lato $\sigma/\sqrt{k}$. La struttura costruita è conosciuta come albero di Steiner. Poiché il costo del minimo albero di copertura è dello stesso ordine di grandezza dell'albero minimo di Steiner[1], il costo dell'albero di Steiner costruito maggiora il costo del minimo albero di copertura calcolato sui k vertici. Quindi

$$O(costo(MST)) = O(costo(SMT)) \leq O(costo(\tau) = O(\sigma\sqrt{k}),$$

dove MST è l'albero minimo di copertura sui k vertici, SMT è l'albero minimo di Steiner sui k vertici e τ è la struttura costruita sempre sui k vartici.

□

Una volta valutato alla peggio il risultato dell'algoritmo ci preoccupiamo

[1]Nel Capitolo 2 è stato analizzato questo risultato.

di capire quanto possa essere lungo, al minimo, un albero ci copertura minimo su k vertici.

Lemma 4.2. *La lunghezza di un albero di copertura minimo per qualsiasi insieme di k punti, dove la massima distanza tra due punti tra i k è δ, è $\Omega(\delta)$.*

Dimostrazione. Se nell'albero i due punti che sono a distanza δ sono adiacenti, banalmente la lunghezza raggiunge δ. Altrimenti, per collegare i due punti dobbiamo seguire un cammino che è una spezzata non rettilinea, la cui distanza supera δ. □

Teorema 4.1. *L'algoritmo Minimo Quadrato approssima la soluzione del k-MST con un fattore $O(\sqrt{k})$.*

Dimostrazione. Sia SOL^* la soluzione ottima e S l'insieme dei k punti di questa soluzione. Siano a e b i due punti che in S sono alla massima distanza, δ. Se prendiamo il quadrato che ha per centro il punto a e per lato 2δ, questo include al suo interno il cerchio che ha per centro il punto a e per raggio δ che a sua volta include tutti i punti di S. Infatti un punto di S non potrebbe essere fuori dal cerchio altrimenti avrebbe una distanza, rispetto ad a, maggiore di δ, contravvenendo l'ipotesi che δ è la massima distanza tra due punti di S. Ora, per il lemma 4.2, SOL^* è $\Omega(\delta)$. Chiamiamo l_{min} il lato del quadrato minimo che ha trovato la nostra euristica. Poiché l'algoritmo ha analizzato anche il punto a, come centro di un quadrato, necessariamente si avrà che $l_{min} \leq 2\delta$. Per il lemma 4.1 si ha che l'albero, SOL, prodotto dall'algoritmo ha un costo $O(l_{min}\sqrt{k})$, ma quindi anche $O(\delta\sqrt{k})$. Il rapporto di approssimazione raggiunto dall'euristica è quindi

$$\frac{costo(SOL)}{costo(SOL^*)} = \frac{O(l_{min}\sqrt{k})}{\Omega(\delta)} = \frac{O(\delta\sqrt{k})}{\Omega(\delta)} = O(\sqrt{k}).$$

□

4.3 L'implementazione

Analizziamo in dettaglio l'algoritmo. Il primo passo rappresenta il nucleo dell'euristica, infatti da solo costituisce il grosso del calcolo e raggiunge la complessità di $O(n \log^3 n)$. Le parti successive hanno un peso ininfluente in relazione al costo complessivo della procedura, inoltre sono solo quantità additive.

Si noti che per la realizzazione del punto 2 è sufficiente una scansione singola dei punti che permette la ricerca del minimo valore l_p associato. Il costo di questa operazione è pari a $O(n)$.

Con un ulteriore scansione dei punti è possibile ottenere l'insieme S, calcolato nel passo 3. Infatti basta selezionare, durante la scansione, i punti che appartengolo al quadrato $[X_p - l_p/2, X_p + l_p/2] \times [Y_p - l_p/2, Y_p + l_p/2]$ sempre in tempo $O(n)$.

Per lo studio fatto sull'euristica k-Kruskal otteniamo che la complessità del quarto punto è $O(n' \log n')$, dove $n' = |S| \leq n$.

Analizziamo a questo punto il primo passo della procedura. In questa fase ci preoccupiamo di trovare, per ogni punto p, il quadrato di lato minimo, con al centro p, che contiene almeno k punti. Per rimanere entro una complessità di $O(n \log^3 n)$, bisogna riuscire a trovare il quadrato minimo corrispondente a p in al più $O(\log^3 n)$ istruzioni. La cosa si può fare utilizzando una struttura

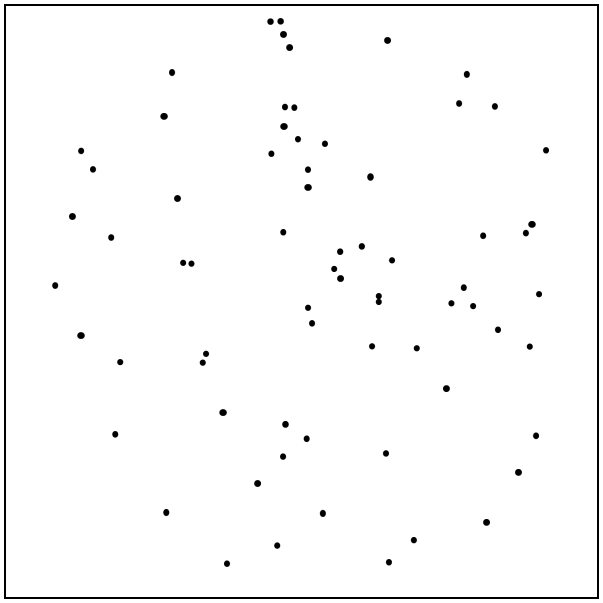

Figura 4.1: Un insieme di 70 punti disposti con distribuzione uniforme

adatta a mantenere i punti e molto rapida nel risolvere domande di tipo intervallo chiamata *range tree*.

Una breve guida a questa struttura la si può trovare nel testo [7] oppure, per un analisi più approfondita, in [8]. La struttura è composta, in linea di massima, da alberi di ricerca binari bilanciati. Come si vede in [7], l'occupazione di memoria di un range tree è $O(n \log n)$. Questo è dovuto alla presenza di $O(n \log n)$ strutture per memorizzare i nodi degli alberi bilanciati. Poiché l'inizializzazione della struttura viene realizzata mediante ricorsione e l'algoritmo effettua una chiamata ricorsiva per ogni struttura nodo costruita, il costo computazionale della procedura di costruzione del range tree è $O(n \log n)$. Una volta calcolato quanto costa la procedura preliminare di costruzione della struttura ci rimane da capire come trovare un quadrato di

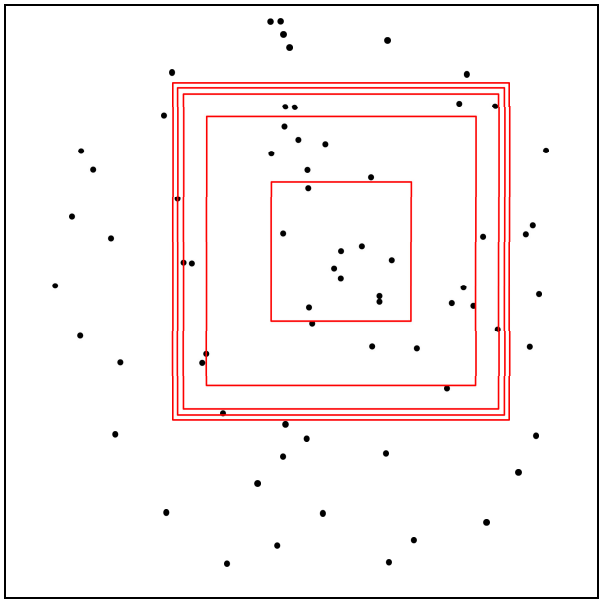

Figura 4.2: Ricerca del quadrato minimo che contenga 35 punti

lato minimo rispetto a un punto. Sapendo, però, che su un range tree si possono effettuare ricerche ad intervallo del tipo "Quali punti si trovano nell'area $[X_1, X_2] \times [Y_1, Y_2]$?" in maniera molto efficiente, il calcolo si semplifica parecchio. Su un range tree una tale interrogazione comporta un tempo di esecuzione di $O(\log^2 n + k)$. Il fattore k è dovuto al processamento in uscita dei k punti trovati dalla domanda, ma nel nostro caso è trascurabile poiché l'unico dato che interessa la risposta è il numero dei nodi contenuti nell'area prescelta. A questo punto, per la ricerca del minimo quadrato, basterà effettuare una serie di interrogazioni sulla struttura con le quali ridimensioneremo il nostro quadrato fino ad ottenere il minimo. Quante e quali interrogazione ci toccherà fare per arrivare alla soluzione?

Osserviamo per prima cosa che, il minimo quadrato contenente k punti

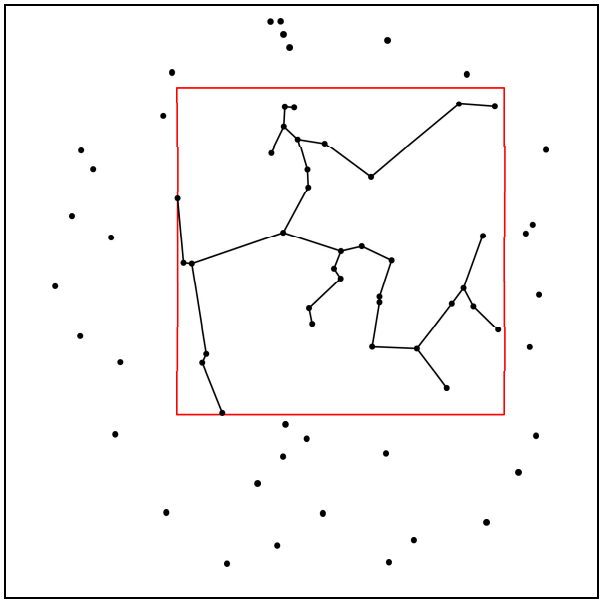

Figura 4.3: Calcolo dell'albero minimo nel quadrato di lato minimo. Nell'esempio ci sono 70 punti e occorre collegarne 35.

dovrà necessariamente avere un punto su uno dei suoi lati. L'affermazione tiene poiché l'assenza di punti sui bordi contrasterebbe con la minimalità del quadrato, si potrebbe infatti prendere un quadrato con un lato leggermente minore rispetto al precedente in maniera che contenga comunque i k vertici. Questa osservazione ci spinge a ricercare quale sia il punto che si trova sul bordo del quadrato di lato minimo. Il nostro problema si è trasformato dalla ricerca del quadrato alla ricerca del punto che si trova sul lato di questo. Per effettuare la ricerca di questo punto ci è convenuto analizzare separatamente i casi in cui il punto che si trovi sul bordo del quadrato, risieda sul lato a destra o a sinistra, e in alto o in basso. Una volta fatto questo ci basterà restituire il quadrato di lato minimo tra i quattro trovati. Senza perdere in

generalità, limitiamoci a trattare il caso in cui cerchiamo il quadrato minimo che abbia un punto sul lato destro. Su questa ipotesi si cercherà un punto alla destra del punto p, tale che il quadrato centrato in p e con il lato passante per il punto trovato contenga almeno k punti. Per la ricerca del punto sul bordo e, quindi, del quadrato utilizziamo una ricerca binaria sui punti alla destra di p ordinati per ascissa. Per chiarire le idee a riguardo riporto di seguito i passi della ricerca.

1. Per ogni punto p:
   (a) Sia $q = p$ e s un punto con ascissa massima
   (b) Ripeti finché q e s non sono adiacenti nell'ordinamento
      i. Sia t il punto a metà, nell'ordinamento, tra q e s
      ii. Sia $r = X_t - X_p$
      iii. Se il quadrato $[X_p - r, X_p + r] \times [Y_p - r, Y_p + r]$ non contiene k punti poni $q = t$
      iv. Altrimenti poni $s = t$

Alla fine della ricerca avremo un quadrato con centro in p e lato $2r$ che ha un punto sul lato destro. Lo stesso si farà per gli altri tre bordi. Una tale ricerca è banalmente realizzata in un tempo $O(\log n')$, dove n' indica il numero dei punti che si trovano alla destra del punto p. Alla peggio la ricerca effettuerà $O(\log n)$ interrogazioni sul range tree. Poiché l'interrogazione, come suddetto, ha un costo di $O(\log^2 n)$, il costo speso per la ricerca del quadrato minimo relativo ad un punto è $O(\log^3 n)$. Il tempo utilizzato dal passo 1 dell'euristica risulta complessivamente $O(n \log^3 n)$.

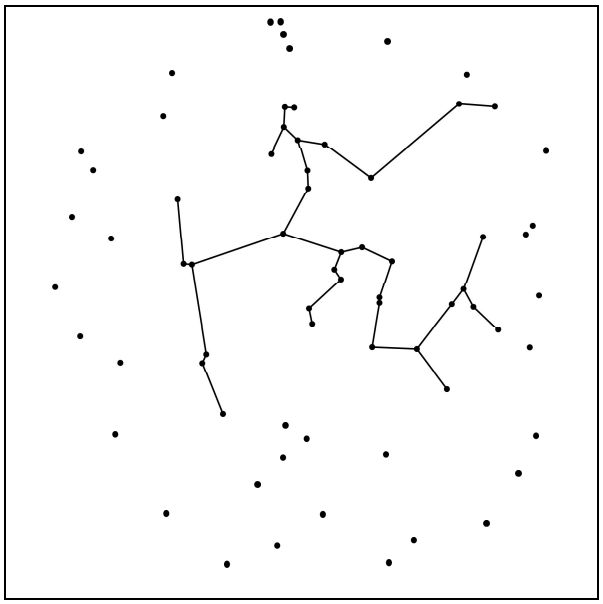

Figura 4.4: L'albero minimo calcolato. Nella figura osserviamo 70 punti dove ne sono stati collegati 35.

Capitolo 5

Algoritmo Ravi

L'algoritmo che ci apprestiamo ad esaminare è stato il primo algoritmo proposto in letteratura per il problema del k-MST nel piano. In questo contesto si è preferito, per comodità, chiamare l'algoritmo con il nome di uno dei ricercatori che l'ha inventato. L'euristica, proposta in [3], garantisce un rapporto di approssimazione della soluzione pari a $k^{\frac{1}{4}}$. Dato $S = \{s_1, s_2, \ldots, s_n\}$ un insieme di punti, per ogni coppia di punti s_i e s_j, in questo capitolo $d(i, j)$ denoterà la distanza nel piano euclideo tra s_i e s_j.

5.1 L'algoritmo

1. Per ogni coppia distinta di punti s_i,s_j in S esegui:

   (a) Costruisci il cerchio C con diametro $\delta = \sqrt{3}d(i, j)$ centrato nel punto medio del segmento $\langle s_i,s_j \rangle$.

   (b) Sia S_C il sottoinsieme di S contenuto in C. Se S_C contiene meno

di k punti, salta alla prossima iterazione del ciclo (prova con la prossima coppia di punti). Altrimenti, prosegui.

(c) Sia Q il quadrato di lato δ che circoscrive C. Dividi Q in k celle, ognuna con lato $\frac{\delta}{\sqrt{k}}$.

(d) Ordina le celle in base al numero di punti di S_C che contengono e scegli il minimo numero di celle tali che la somma dei numeri dei punti che contengono sia almeno k. Se necessario, elimina arbitrariamente alcuni punti dall'ultima cella scelta in modo che il numero dei punti contenuti in tutte le celle sia esattamente k.

(e) Costruisci il minimo albero di copertura per i k punti scelti. Il peso dell'albero costruito è la soluzione associata alla coppia di punti $\langle s_i, s_j \rangle$.

2. Restituisci in output il minor valore trovato nelle varie soluzioni.

5.2 Il rapporto di approssimazione

La dimostrazione che l'algoritmo garantisce il rapporto di approssimazione citato è strutturata in tre fasi: una prima fase nella quale verrà mostrato come, data una soluzione ottima, questa non possa essere arbitrariamente piccola. Nella seconda fase si proporrà una limitazione superiore al costo dell'albero prodotto dall'euristica. Nella terza, infine, mettendo insieme i pezzi, si arriverà al risultato.

Limite inferiore del peso di un k-MST ottimo

Lemma 5.1. *Sia S un insieme di punti nel piano, con diametro Δ(il diametro è la massima distanza tra due punti dell'insieme). Siano a e b due punti in S tali che $d(a,b) = \Delta$. Allora il cerchio con diametro $\sqrt{3}\Delta$ centrato nel punto medio del segmento $\langle a,b\rangle$ contiene S.*

Dimostrazione. Supponiamo che esista un punto $p \in S$ non contenuto nel cerchio di diametro $\sqrt{3}\Delta$ centrato nel punto medio del segmento $\langle a,b\rangle$. Se p giace sull'asse del segmento $\langle a,b\rangle$, allora $d(a,p) = d(b,p)$. Definiamo M il punto medio del segmento $\langle a,b\rangle$. I punti a,p,M formano un triangolo rettangolo dove $d(a,M) = \frac{\Delta}{2}$ perché $d(a,b) = \Delta$, e $d(M,p) > \frac{\sqrt{3}}{2}\Delta$ perché p non appartiene a C. Ma questo deduce che $d(a,p) > \sqrt{(\frac{\sqrt{3}}{2}\Delta)^2 + (\frac{\Delta}{2})^2} = \sqrt{\frac{3}{4}\Delta^2 + \frac{\Delta^2}{4}} = \sqrt{\frac{\Delta^2}{4}(3+1)} = \Delta$, la quale cosa contraddice il fatto che il diametro di S è Δ in quanto i punti a e p in S distano tra loro più di Δ. Se p non giace sull'asse del segmento $\langle a,b\rangle$, allora p è più vicino ad a o b rispetto all'altro. Senza perdere in generalità diciamo che p è più vicino ad a, quindi $\widehat{bMp} > 90°$. Ma allora $d(p,b) > \sqrt{d(M,p)^2 + d(b,M)^2} \geq \sqrt{(\frac{\sqrt{3}}{2}\Delta)^2 + (\frac{\Delta}{2})^2} = \Delta$. Cosa che contrasta con il fatto che il diametro di S è Δ. □

Lemma 5.2. *Data una griglia qudrata nel piano con il lato di ogni cella pari a σ. La lunghezza di un MST per un qualsiasi insieme di t punti, dove ogni punto è preso da una cella distinta, è $\Omega(t\sigma)$.*

Dimostrazione. Prendiamo un punto dall'insieme e eliminiamo tutti i punti contenuti nelle otto celle adiacenti al punto scelto. Ripetendo questa procedura finché abbiamo due punti in celle adiacenti, otterremo un sottoinsieme di $t/9 = \Omega(t)$ punti tali che la distanza tra una qualsiasi coppia di questi sia

almeni σ. Per costruire un qualunque albero tra punti che sono σ-distanti a coppie paghiamo almeno una lunghezza σ per ogni arco. Poichè dobbiamo collegare $\Omega(t)$ punti spendiamo $\sigma \cdot (\Omega(t) - 1)$ che equivale a $\Omega(t\sigma)$. □

Sia P^* un insieme di punti in una soluzione ottima di un'istanza del problema. Sia Δ il diametro di P^*, e sia OPT la lunghezza di un MST per P^*. Consideriamo un'iterazione nella quale il cerchio costruito dall'euristica sia definito da due punti a e b in P^* tali che $d(a, b) = \Delta$. Per il Lemma 5.1 il cerchio costruito con i punti a e b contiene interamente P^*. Sia g il numero di celle quadrate usate dall'euristica per la selezione dei k punti in questa iterazione. É facile notare che $OPT \geq \Delta$, perchè Δ è il diametro di P^*. Poiché l'euristica utilizza un minimo numero (g) per la selezione dei k punti, i punti in P^* devono appartenere a g o più celle. Una cosa da notare è che il lato di una cella quadrata è $\sqrt{3}\Delta/\sqrt{k}$. Questo porta al seguente corollario del Lemma 5.2.

Corollario 5.1. $OPT = \Omega(g\Delta/\sqrt{k})$

Limite superiore del peso della soluzione prodotta dall'euristica

In questa sezione si proverà un limite superiore al peso dell'albero di copertura che genera l'euristica precedentemente descritta.

Lemma 5.3. *La lunghezza di un albero di copertura minimo per qualsiasi insieme di q punti in un quadrato di lato σ è $O(\sigma\sqrt{q})$.*

Dimostrazione. Deriva direttamente dal lemma 4.1 □

Lemma 5.4. *La lunghezza dell'albero di copertura costruito dall'euristica è $O(\sqrt{g}\Delta)$.*

Dimostrazione. Sia Q_i l'insieme di punti nella i-esima cella scelta dall'euristica, $1 \leq i \leq g$. Così $\Sigma_{i=1}^{g}|Q_i| = k$. Consideriamo la seguente procedura, articolata in due passi, che costruisce un albero di copertura per i punti in $\bigcup_{i=1}^{g} Q_i$.

Passo I: Costruisci il minimo albero di copertura per i punti in Q_i,$1 \leq i \leq g$. Osserviamo che i punti in Q_i sono contenuti all'interno di un quadrato di lato $\sqrt{3}\Delta/\sqrt{k}$. Applicando il Lemma 5.3, la lunghezza di un MST per Q_i è $O(\frac{\Delta}{\sqrt{k}}\sqrt{|Q_i|})$. Così la lunghezza totale di tutti gli alberi minimi di copertura costruiti in questo passo è $O(\frac{\Delta}{\sqrt{k}}\Sigma_{i=1}^{g}\sqrt{|Q_i|}) = O(\sqrt{g}\Delta)$ per la diseguaglianza di Cauchy-Schwartz (diseguaglianza di Hölder).

Passo II: Connetti i g alberi di copertura costruiti al Passo I in un unico albero di copertura nella seguente maniera. Scegli un punto a caso da ogni insieme Q_i, $(1 \leq i \leq g)$, e costruisci un MST su i g punti scelti. Osserviamo che questi g punti sono all'interno di un quadrato di lato $\sqrt{3}\Delta$. Così per il Lemma 5.3, la lunghezza dell'MST costruito in questo passo è $O(\sqrt{g}\Delta)$.

Quanto detto dimostra che l'albero di copertura costruito dalla procedura è $O(\sqrt{g}\Delta)$. □

L'analisi finale

Vedremo ora quanto sia buona la soluzione prodotta dall'euristica rispetto all'ottimo del problema.

Teorema 5.1. *L'algoritmo polinomiale Ravi, dati n punti nel piano euclideo, e un intero positivo $k \leq n$, costruisce un albero ricoprente almeno k di questi punti tale che la lunghezza totale dell'albero è al più $O(k^{\frac{1}{4}})$ volte quella di un*

qualsiasi albero di minima lunghezza ricoprente k qualsiasi punti tra quelli iniziali.

Dimostrazione. Come mostrato precedentemente, $OPT = \Omega(\Delta)$, e per il Corollario 5.1, $OPT = \Omega(g\Delta/\sqrt{k})$. Così $OPT = \Omega(\max\{\Delta, g\Delta/\sqrt{k}\})$. Inoltre, per il Lemma 5.4, la lunghezza dell'albero di copertura prodotto dall'euristica è $O(\sqrt{g}\Delta)$. Questo implica che il rapporto di approssimazione dell'algoritmo è $O(\min\{\sqrt{g}, \sqrt{k/g}\})$. Ora, se $\sqrt{g} < \sqrt{k/g}$ allora $\sqrt{g} \cdot \sqrt{g} < \frac{\sqrt{k}}{\sqrt{g}} \cdot \sqrt{g}$ e $g < \sqrt{k}$ quindi $\sqrt{g} < k^{\frac{1}{4}}$, che porta ad avere $O(k^{\frac{1}{4}})$. Se $\sqrt{g} > \sqrt{k/g}$ abbiamo che $g > \sqrt{k}$, $\sqrt{g} > k^{\frac{1}{4}}$, $\frac{1}{\sqrt{g}} < \frac{1}{k^{\frac{1}{4}}}$, $\frac{\sqrt{k}}{\sqrt{g}} < \frac{\sqrt{k}}{k^{\frac{1}{4}}}$, $\frac{\sqrt{k}}{\sqrt{g}} < k^{\frac{1}{4}}$, che porta ad avere $O(k^{\frac{1}{4}})$. Possiamo concludere dicendo che il rapporto di approssimazione dell'algoritmo è $O(k^{\frac{1}{4}})$. □

5.3 L'implementazione

Come mostrato nello pseudocodice, l'algoritmo richiede un analisi, eventualmente per ogni coppia di punti del piano, della zona circostante, sulla quale genera una griglia e calcola l'albero. Per la scansione di tutte le coppie di punti nell'implementazione si è fatto uso di una procedura, indipendente dal contesto, che genera tutte le combinazioni $\binom{n}{h}$, dove nel nostro caso $h = 2$. Questa scansione, per le proprietà dei coefficienti binomiali, richiede $O(n^2)$ iterazioni. Per ogni coppia di punti viene costruito, al punto (a), un cerchio e si calcola, al punto (b), se il cerchio contiene almeno k punti, Figura 5.2. Mentre la costruzione del cerchio viene effettuata in un tempo costante, il calcolo del passo (b) consiste in uno scorrimento degli n punti, e quindi un costo $O(n)$.

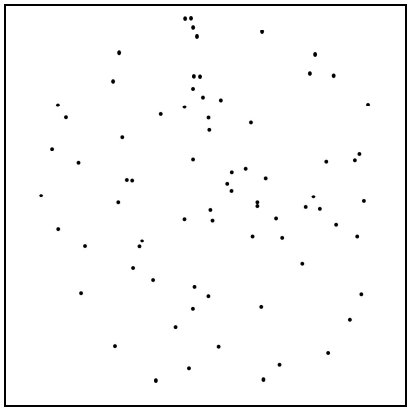

Figura 5.1: Un insieme di 75 punti

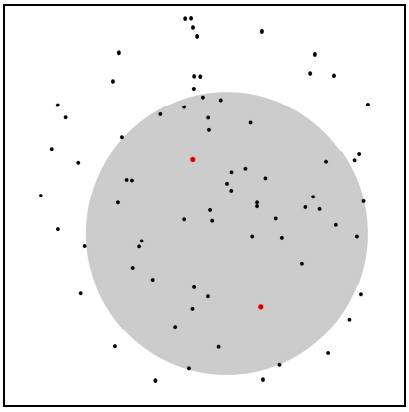

Figura 5.2: In evidenza il cerchio calcolato per l'analisi di una coppia di punti

Nel passo (c) si pone il problema di creare un struttura che rappresenti una griglia di $\sqrt{k} \times \sqrt{k}$ celle e che associ ad ogni cella la lista dei punti che essa contiene. Questa struttura ha bisogno di $O(k)$ iterazioni per essere inizializzata e di una scansione sul vettore dei punti, $O(n)$, per essere riempita, Figura 5.3.

Arrivati all'istruzione (d) dobbiamo ordinare le celle in base al numero di punti che contengono. Questo lo si fa in tempo $O(k \log k)$. Poi bisogna prendere i punti dalle celle più popolate fino a sceglierne k, e questo lo si può fare accedendo alla struttura delle celle ordinata per $O(k)$ volte. L'ultimo passo del ciclo, il passo (e), genera l'albero vero e proprio, a partire dai k punti scelti. Per l'esecuzione del passo (e) si è pensato di riutilizzare il lavoro fatto prendendo come procedura di costruzione dell'albero, l'euristica k-Kruskal esaminata nel Capitolo 3. Questo perché l'euristica k-Kruskal è stata creata per garantire ottime prestazioni dal punto di vista computazionale. Infatti,

in questa maniera il calcolo del punto (e) costa solamente $O(k \log k)$. Inoltre è importante notare che non viene, in questo utilizzo, sfruttata la procedura di potatura che caratterizza l'euristica k-Kruskal, in modo da non far interferire le valutazioni dei due algoritmi. Questo si realizza passando all'euristica k-Kruskal un parametro k pari al numero di punti in input. In questa maniera otteniamo il calcolo dell'albero di copertura sui punti selezionati dall'euristica Ravi mediante la tecnica di Kruskal pura [1], come si nota nella Figura 5.4.

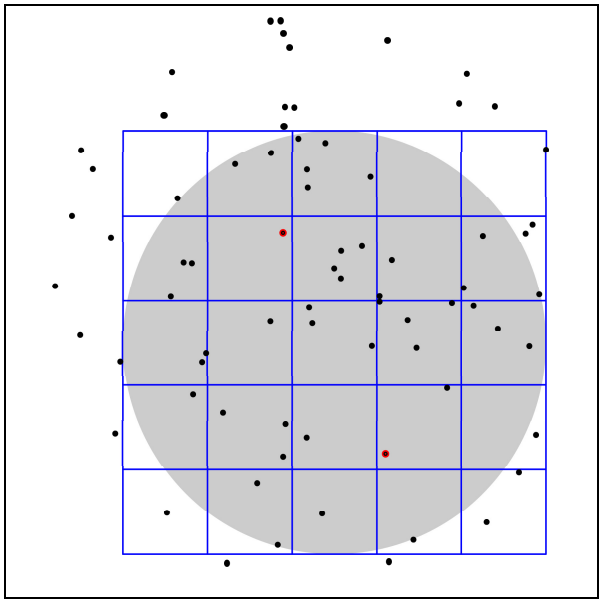

Figura 5.3: La griglia dell'algoritmo Ravi, $n = 75$ e $k = 35$

Va osservato infine che l'algoritmo Ravi restituisce, al punto 2 l'albero minimo tra tutti quelli trovati.

Dalla combinazione del punto 1 e del passo (e), il più costoso del ciclo, il tempo di calcolo richiesto dall'intera euristica risulta $O(n^2 \cdot k \log k)$.

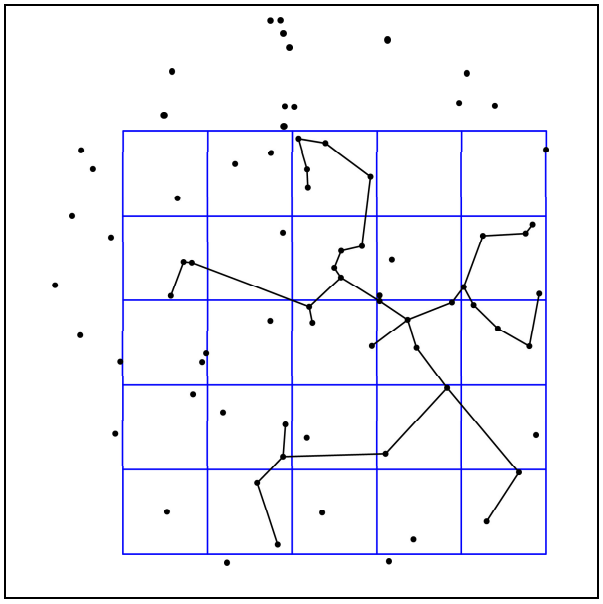

Figura 5.4: Uno degli alberi prodotti dall'algoritmo Ravi, $n = 75$ e $k = 35$

Capitolo 6

L'algoritmo Garg

Spesso, per ottenere dei buoni risultati, invece di costruire un algoritmo da zero conviene modificare uno già esistente per aumentarne il rendimento. É quello che hanno fatto Garg e Hochbaum in [20], seguendo la linea portante che era stata utilizzata dall'algoritmo Ravi e portandola all'estremo. Invece di dividere il piano in una griglia e prendere il numero minore di celle che contengano k punti, hanno usato una famiglia di griglie rettangolari. L'insieme di punti selezionato è tale che per ogni griglia vengano prese solo "poche" celle. Hanno dimostrato che, per una particolare scelta delle griglie, l'albero di copertura costruito sui punti indotti da queste sia abbastanza piccolo. Esaminiamo l'algoritmo per capire come queste griglie consentano di individuare un insieme di punti che permetta di costruire un albero di copertura di costo $O(\log k)$ volte quello dell'ottimo. Se prendiamo un insieme di punti $S = \{s_1, s_2, \ldots, s_n\}$, per ogni coppia di punti s_i e s_j, $d(i, j)$ in questo capitolo sarà la distanza nel piano euclideo tra s_i e s_j.

6.1 L'algoritmo

1. Per ogni coppia distinta di punti s_i,s_j in S esegui:

   (a) Costruisci il cerchio C con diametro $\delta = \sqrt{3}d(i,j)$ centrato nel punto medio del segmento $\langle s_i,s_j \rangle$.

   (b) Sia S_C il sottoinsieme di S contenuto in C. Se S_C contiene meno di k punti, salta alla prossima iterazione del ciclo (prova con la prossima coppia di punti). Altrimenti, prosegui.

   (c) Sia Q il quadrato di lato δ che circoscrive C.

   (d) Calcola l'insieme S^* di k punti contenuti nella regione Q che hanno **minimo potenziale**

   (e) Costruisci il minimo albero di copertura per i k punti scelti. Il peso dell'albero costruito è la soluzione associata alla coppia di punti $\langle s_i,s_j \rangle$.

2. Restituisci in output il minor valore trovato nelle varie soluzioni.

Nel punto (e) dell'algoritmo si fa riferimento al calcolo dell'insieme di punti di minimo potenziale. L'illustrazione di questo oggetto che permette all'algoritmo di raggiungere le prestazioni dichiarate si trova nel paragrafo 6.2. Per la descrizione della procedura che calcola in tempo polinomiale l'insieme con minimo potenziale si rimanda al paragrafo 6.3.

6.2 Il rapporto di approssimazione

Il primo passo da compiere per arrivare alla prova che il rapporto di approssimazione dell'algoritmo Garg è $O(\log k)$ consiste nella spiegazione della funzione *potenziale*. Noi cerchiamo un insieme di punti che siano "vicini", nel senso che il peso del minimo albero di copertura costruito su di essi sia piccolo. Per questo definiamo una funzione *potenziale* sulla famiglia dei sottinsieme dei vertici, $P : 2^V \to \mathbb{R}^+$. La nostra scelta della funzione potenziale è tale che per un insieme di punti, questi siano "vicini" se e solo se il potenziale dell'insieme è piccolo.

Data una regione quadrata nel piano Q, sia G_i una griglia rettangolare su Q, dove ogni cella è un quadrato. Definiamo la *dimensione* della griglia G_i la lunghezza del lato della sua cella.

Definizione 6.1. *Chiamiamo G_i-potenziale di un insieme $S \subseteq V$, denotato da $G_i(S)$, il valore $x_i t_i$, dove x_i è la dimensione della griglia G_i e t_i è il numero di celle di G_i che contiene i punti di S.*

La definizione opera su una sola griglia e non garantisce che per un insieme di punti S che ha un basso valore $G_i(S)$ si può costruire un albero di copertura di costo basso. Infatti immaginiamo che i punti di S siano raggruppati in poche celle della griglia G_i, ma che queste celle siano molto lontane tra di loro. Si avrebbe che il G_i-potenziale di S sarebbe basso, ma l'albero di copertura minimo costruito su S avrebbe un costo alto a causa degli archi utilizzati per connettere coppie di punti appartenenti a celle diverse. Per questo motivo è intelligente considerare una famiglia di griglie G_i che abbiano dimensioni diverse, con le quali calcolare i diversi valori dei potenziali.

Osserviamo che, se δ è il lato del quadrato Q, qualunque insieme di punti, rispetto alla griglia di dimensione δ, ha G_i-potenziale uguale a δ. Inoltre se la griglia G_i ha dimensione minore di δ/k, qualunque insieme di punti avrà G_i-potenziale minore di δ. Questo rende inutile, per i nostri scopi, considerare griglie di dimensione minore di δ/k. Sia G_0 una griglia rettangolare di dimensione $x_0 = \delta/k$ in Q. Definiamo G_i come la griglia dove ogni cella è un quadrato composto da quattro celle della griglia G_{i-1}. Se x_i denota la dimensione di G_i, allora $x_i = 2x_{i-1} = 2^i x_0$. Così $x_{\log k} = 2^{\log k} x_0 = kx_0 = \delta$, $G_{\log k}$ è il quadrato Q[1]. In questa maniera otteniamo $\log k$ differenti griglie: G_0,G_1,...,$G_{\log k-1}$. La Figura 6.1 mostra le griglie G_0, G_1, G_2 disegnate in modo puntinato, tratteggiato e pieno rispettivamente.

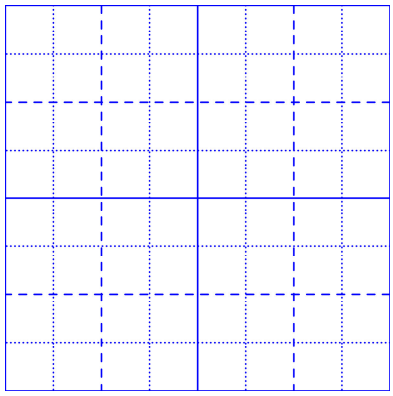

Figura 6.1: Le griglie G_0 (puntinato), G_1 (tratteggiato), G_2 (pieno).

Definizione 6.2. *Il* potenziale *di un insieme di punti S è la somma dei*

[1]In questo contesto si assume che k sia una potenza di 2. Per un valore generale di k sostituire nella trattazione il termine $\log k$ con $\lceil \log k \rceil$.

G_i-potenziali di S, e cioè

$$P(S) = \sum_{i=0}^{\log k - 1} G_i(S).$$

Nel seguente risultato vediamo che un insieme di punti "vicini" ha un potenziale basso.

Lemma 6.1. *Se V^* è l'insieme dei punti di una soluzione ottima, allora*

$$P(V^*) = \sum_{i=0}^{\log k - 1} G_i(V^*) \leq 8 \log k \cdot OPT,$$

dove OPT è il costo del minimo albero di copertura costruito su V^*.

Dimostrazione. Consideriamo una griglia G_i di dimensione x_i e t_i celle che contengano tutti i punti di V^*. Prima proveremo che il G_i-potenziale di V^*, $G_i(V^*)$, è al più $8 \cdot OPT$.

Distinguiamo due casi in base al valore di t_i:

Caso 1: $t_i \leq 4$. Poiché la griglia con le celle più grosse ha dimensione $\delta/2$, allora $x_i \leq \delta/2$. Inoltre se p, q sono i punti presi dall'algoritmo per la costruzione delle griglie, $\delta = \sqrt{3} d(p,q) \leq \sqrt{3} \cdot OPT$[2] e quindi

$$G_i(V^*) = x_i t_i \leq 2\delta \leq 2\sqrt{3} \cdot OPT < 8 \cdot OPT.$$

Caso 2: $t_i > 4$. Coloriamo le celle della griglia nella maniera seguente: assegnamo a ogni cella un colore dipendente dalle parità della riga e della colonna alle quali la cella appartiene. Siccome ci sono quattro possibilità, utilizzando solamente quattro colori possiamo fare in modo che non ci siano

[2] L'ultimo passaggio è dovuto al fatto che un albero di copertura su un insieme S di punti deve essere almeno lungo quanto la massima distanza che c'è tra due punti di S.

due celle adiacenti dall'alto, dal basso o diagonalmente che abbiano lo stesso colore. Poiché i punti di V^* giaciono in t_i celle, ci sono almeno $\lceil t_i/4 \rceil$ celle che contengono punti di V^* e appartengono allo stesso colore. Per collegare due celle aventi lo stesso colore è richiesto un arco di lunghezza almeno x_i. Quindi l'albero di copertura su V^* deve avere lunghezza almeno $x_i(\lceil t_i/4 \rceil - 1)$ che, essendo $t_i > 4$, è almeno $x_i t_i/8$. Proseguendo

$$G_i(V^*) = x_i t_i \leq 8 \cdot OPT.$$

Poiché $G_i(V^*) \leq 8 \cdot OPT$ per ogni G_i, con $0 \leq i \leq \log k - 1$, il lemma è dimostrato. □

Per mostrare quanto il valore del potenziale incida sul costo dell'albero di copertura di un insieme di punti, proveremo la direzione inversa del lemma 6.1, e cioè che un insieme di punti che ha un piccolo potenziale presenta un costo basso del minimo albero che li ricopre.

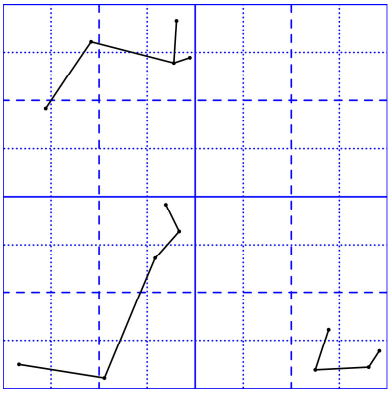

Figura 6.2: Costruendo l'albero.

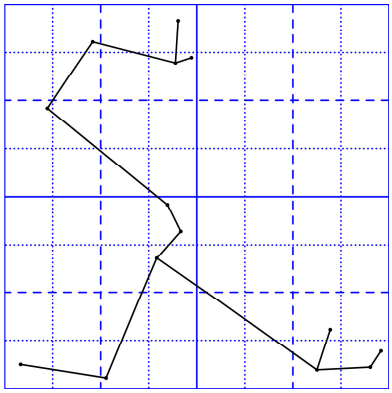

Figura 6.3: La connessione dei sottoalberi in celle diverse della griglia G_2.

Lemma 6.2. *Il minimo albero di copertura costruito su un qualsiasi insieme, S, di k punti ha un costo massimo di $\sqrt{2}(\sum_{i=0}^{\log k-1} G_i(S))$.*

Dimostrazione. Mostreremo questo costruendo un albero di copertura su S di peso non superiore di quanto affermato. L'albero è costruito in maniera gerarchica; prima colleghiamo tutti i punti di S contenuti in una cella della griglia G_0 e poi aggiungiamo archi tra le celle di G_0 per collegare i punti di S in una cella di G_1 e così via (Figura 6.2 e Figura 6.2).

Iniziamo con la griglia G_0 e per ogni cella di questa griglia prendiamo un albero sui punti di S contenuti nella cella. Consideriamo una cella contenente m punti di S. Poiché nessun arco preso ha una lunghezza superiore al diametro della cella ($\sqrt{2}x_0$), il costo totale degli archi presi da questa cella è al più $\sqrt{2}x_0(m-1)$. Quindi il costo totale degli archi presi da tutte le celle della griglia G_0 è al più $\sqrt{2}x_0(k-t_0)$, dove t_0 è il numero delle celle di G_0 contenenti punti di S. Tuttavia $x_0 t_0 = G_0(S)$ che implica che il peso degli archi presi in questo passo è $\sqrt{2}(x_0 k - G_0(S)) = \sqrt{2}(\delta - G_0(S))$.

L'algoritmo per connettere i punti di S procede per iterazioni. All'inizio dell'i-esima iterazione tutti i punti di S contenuti in una cella di G_{i-1} sono connessi. In questa iterazione noi aggiungiamo archi tra le celle di G_{i-1} per garantire che, per ogni cella di G_i, tutti i punti di S contenti in questa cella più grande siano connessi. Consideriamo una cella di G_i. Sia m ($0 \leq m \leq 4$) il numero delle celle che contengono punti di S appartenenti alla griglia G_{i-1} incluse nella cella di G_i in questione. Questo ci porta ad aggiungere $m-1$ archi di peso al massimo pari alla lunghezza del diametro della cella ($\sqrt{2}x_i$). Il peso degli archi presi per questa cella è dunque $\sqrt{2}x_i(m-1)$. Quindi il peso totale degli archi presi da tutte le celle della griglia G_i è $\sqrt{2}x_i(t_{i-1}-t_i)$, dove

t_{i-1} e t_i sono i numeri di celle in G_{i-1} e G_i, rispettivamente, che contengono punti di S. Poiché $x_i = 2x_{i-1}$, il peso totale degli archi presi in questo passo è al più

$$\sqrt{2}x_i(t_{i-1} - t_i) = \sqrt{2}(2x_{i-1}t_{i-1} - x_i t_i) = \sqrt{2}(2G_{i-1}(S) - G_i(S)).$$

Alla fine delle $\log k$ iterazioni avremo un albero ricoprente tutti i punti di S. Il costo totale dell'albero è al più la somma dei costi appena calcolati, relativi agli archi aggiunti per ciascuna iterazione e cioè

$$\sqrt{2}(\delta - G_0(S) + 2G_0(S) - G_1(S) + \cdots + 2G_{\log k - 1}(S) - G_{\log k}(S)),$$

che risulta uguale a $\sqrt{2}\sum_{i=0}^{\log k - 1} G_i(S)$. Questo completa la prova. □

Teorema 6.1. *L'algoritmo Garg produce una soluzione approssimata per il problema k-MST nel piano euclideo con un fattore di approssimazione* $O(\log k)$.

Dimostrazione. Avendo provato che un insieme di punti ha un piccolo potenziale se e solo se possiede un albero di costo basso che lo ricopre (lemmi 6.1 e 6.2) dimostriamo che l'abero minimo di copertura calcolato su S^* ha peso al più $O(\log k)$ volte OPT. Ricordiamo che S^* è l'insieme di k punti presi su Q avente il minimo potenziale. Siccome l'insieme di punti di un k-MST ottimo, V^*, è comunque contenuto in Q, si ha che $P(S^*) \leq P(V^*)$ e quindi

$$costo(SOL) \leq \sqrt{2}P(S^*) \leq \sqrt{2}P(V^*)$$

$$\leq 8\sqrt{2}\log k \cdot OPT = O(\log k) \cdot OPT,$$

dove SOL è la soluzione prodotta dall'euristica. □

Viene qui proposto un esempio per mostrare che il divario tra il limite inferiore al costo dell'ottimo ed il limite superiore al costo dell'albero prodotto dall'algoritmo è $O(\log k)$. Per limite inferiore al costo dell'ottimo intendiamo l'analogo del corollario 5.1. Nel nostro caso avremo per ogni griglia G_i che $OPT = \Omega(g_i \cdot \delta/k)$, dove g_i è il numero minimo di celle della griglia G_i che contengono punti.

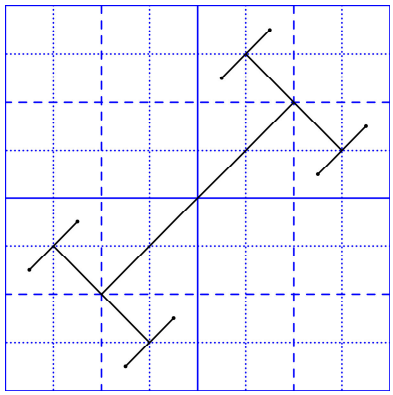

Figura 6.4: Un esempio su misura.

Nella Figura 6.4 si vede un insieme di $k = 8$ punti in un quadrato di lato δ. In questo caso per ogni griglia, durante l'esecuzione, viene aggiunta una lunghezza complessiva degli archi pari a δ. Alla fine delle $\log k$ iterazioni avremo un costo, per l'albero di copertura costruito su questo insieme di punti, di $\delta \log k$. I punti sono disposti in maniera che ogni cella di G_i che contiene punti, li contiene posizionati esattamente in due celle delle quattro della griglia G_{i-1} che la compongono. Questo implica che $t_i = k/2^i$ e così il potenziale di questo insieme di punti rispetto a ogni griglia è esattamente δ. Numeriamo le quattro celle della griglia G_{i-1} che compongono una cella della

griglia G_i come 1,2,3,4 in senso orario partendo dalla cella in alto a sinistra. La nostra distribuzione di punti è scelta in maniera che le due celle di G_{i-1} nelle quali sono contenuti i punti di G_i sono numerate con 2 e 4 se i è pari e 1 e 3 se i è dispari. Dato che $OPT = \Omega(g_i \cdot \delta/k)$, nell'esempio il valore massimo di g_i lo si ha sulla griglia G_0 dove $g_i = k$ e quindi $OPT = \Omega(\delta)$. Non è difficile convincersi che il minimo albero di Steiner[3] che ricopre questi punti ha un costo $\Omega(\delta \log k)$, quindi $\log k$ volte il valore del limite inferiore dell'ottimo.

6.3 L'implementazione

L'algortimo Garg presenta la stessa struttura di base che è presente nell'algoritmo Ravi. Il frammento che differenzia le due euristiche si trova nella parte centrale dell'algoritmo ed è costituito dalla ricerca dell'insieme di punti sul quale costruire l'albero di copertura. Mentre nell'algoritmo Ravi abbiamo utilizzato per far questo una griglia, l'algortimo Garg, come già detto, analizza una famiglia di griglie. Per questo motivo nei due algoritmi ci sono parti comuni che presentano la stessa complessità. Anche qui analizziamo tutte le coppie di punti; il punto 1 che fa questo costa $O(n^2)$. Come in Ravi la costruzione del cerchio al punto (a) viene fatta in tempo costante e il calcolo che ci occorre per sapere se ci siano o no k punti nel cerchio viene effettuato al punto (b) con un costo di $O(n)$. Il punto (c) rappresenta un passo strutturale e viene calcolato in tempo unitario. Il passo (d) invece rappresenta il calcolo dell'insieme di minimo potenziale e rappresenta il nucleo dell'euristi-

[3] Vedi Capitolo 2.

ca. Prima di esplicitare la procedura per il calcolo dell'insieme di punti di minimo potenziale è giusto spiegare come questo può essere fatto in modo semplice e corretto.

La ricerca dell'insieme di punti di minimo potenziale, secondo l'algoritmo proposto nell'euristica Garg, ha un costo computazionale polinomiale e fa uso della tecnica della *programmazione dinamica*. L'algoritmo associa ad ogni cella di ogni griglia G_i un vettore L di k elementi. Il p-esimo elemento nel vettore indica il valore del minimo potenziale di un insieme di p punti contenuti nella cella in relazione alle griglie $G_0, G_1, ..., G_{i-1}, G_i$. Formalmente,

$$L(p) = \min_{S:|S|=p} \sum_{j=0}^{i} G_j(S),$$

dove l'insieme di punti S è contenuto nella cella di G_i presa in considerazione.

I vettori delle celle della griglia G_0 possono essere costruiti senza fare calcoli. Se una cella di G_0 contiene m punti, allora gli elementi $L(1), L(2), ..., L(m)$ sono inizializzati a x_0, mentre ai rimanenti elementi viene assegnato un valore notevolmente alto che noi possiamo intendere come ∞. Avendo a disposizione i vettori per le celle della griglia G_{i-1}, possiamo calcolare una cella nella griglia G_i nella seguente maniera. Consideriamo le quattro celle di G_{i-1} che compongono questa cella di G_i. Per ottenere il p-esimo elemento dobbiamo trovare la maniera migliore di dividere p ($p = p_1 + p_2 + p_3 + p_4$) tale che prendendo p_1, p_2, p_3, p_4 punti rispettivamente da queste quattro celle sia minimo il totale dei potenziali rispetto alle celle $G_0, G_1, ..., G_{i-1}$. Questa operazione può essere fatta in tempo polinomiale considerando tutte le 4-partizioni del numero p e, per ogni partizione, usando i vettori delle celle di G_{i-1} per valutare il totale dei potenziali calcolati in precedenza sulle celle delle griglie

$G_0, G_1, ..., G_{i-1}$. Il p-esimo valore del vettore associato ad una cella di G_i è la somma del minimo potenziale ricercato sulle 4-partizioni più il valore del potenziale della cella di G_i e cioè x_i. Così, per $p \geq 1$, si ha

$$L(p) = \min_{p_1,p_2,p_3,p_4} \{L_1(p_1) + L_2(p_2) + L_3(p_3) + L_4(p_4)\} + x_i,$$

dove $p_1 + p_2 + p_3 + p_4 = p$ e L_1, L_2, L_3, L_4 sono i vettori delle quattro celle di G_{i-1} contenute in questa cella.

La lista L, può essere calcolata efficientemente in tempo $O(k^2)$ così:

$$\begin{aligned} L(p) &= L_1(p), \\ L(p) &= \min_{0\leq i\leq k} L(i) + L_2(p-i), \\ L(p) &= \min_{0\leq i\leq k} L(i) + L_3(p-i), \\ L(p) &= \min_{0\leq i\leq k} L(i) + L_4(p-i), \end{aligned}$$

e, per $p \geq 1$,

$$L(p) = L(p) + x_i.$$

In definitiva possiamo dare la procedura per il calcolo dell'insieme di punti di minimo potenziale contenuti in una regione quadrata Q[4]:

1. Per ogni cella di G_0 fai
   (a) Per p che va da 0 a k fai
      i. Se la cella contiene almeno p punti allora $L(p) = x_0$
      ii. Altrimenti $L(p) = \infty$

[4]Le griglie $G_0, G_1, ...$ verranno costruite sul quadrato Q.

2. Per i che va da 1 a $\log k$ fai

   (a) Per tutte le celle di G_i fai

      i. Calcola il vettore L.

Lemma 6.3. *Il k-esimo elemento nel vettore associato alla cella Q della griglia $G_{\log k}$[5] identifica l'insieme di k punti contenuto in Q con minimo potenziale.*

Dimostrazione. Sia S^* l'insieme dei punti associati al k-esimo elemento del vettore di Q. Dalla nostra definizione dei vettori L segue che S^* ha il minimo delle somme dei G_i-potenziali in relazione alle griglie $G_0, G_1, ..., G_{\log k}$ tra tutti i possibili insiemi di k punti contenuti in Q, e cioè:

$$\sum_{i=0}^{\log k} G_i(S^*) \leq \sum_{i=0}^{\log k} G_i(S),$$

dove $|S^*| = |S| = k$ e $S \subseteq V$. Inoltre, poiché $G_{\log k}(\phi) = \delta$ per ogni insieme $\phi \subseteq V$, S^* minimizza anche $P(\cdot) = \sum_{i=0}^{\log k - 1} G_i(\cdot)$ e quindi è un insieme di minimo potenziale. □

Va sottolineato che per tener traccia dell'insieme di punti associato ad un valore di un vettore L sono stati usati, per ogni cella di ogni griglia, quattro vettori ausiliari. Questi servono, durante il calcolo della lista L, per memorizzare i valori p_1, p_2, p_3, p_4 della 4-partizione che minimizza il valore del potenziale parziale di una cella.

Il costo, in termini di tempo, della procedura che trova l'insieme di punti di minimo potenziale è $O(k^4)$. Questo perché il numero totale delle celle di

[5] In questo contesto si assume che k sia una potenza di 2. Per un valore generale di k sostituire nella trattazione il termine $\log k$ con $\lceil \log k \rceil$.

tutte le griglie è $1^2 + 2^2 + 4^2 + \cdots + k^2 = O(k^2)$, e il calcolo di ogni vettore L per ogni cella ha una complessità di $O(k^2)$. A questo punto la complessità dell'algoritmo Garg si deduce dalla composizione del punto 1, quello dove si analizzano tutte le coppie degli n punti ($O(n^2)$), e la complessità della procedura di ricerca del minimo potenziale. Infatti tra le varie istruzioni presenti nel ciclo principale dell'algoritmo Garg il calcolo dell'insieme di minimo potenziale rappresenta il passo più costoso dal punto di vista computazionale, mentre gli altri passi hanno un costo additivo irrilevante. Infine il costo dell'euristica Garg risulta $O(n^2 \cdot k^4)$, un valore abbastanza alto in relazione alle euristiche studiate in questo lavoro.

Capitolo 7

Analisi sperimentale

Nei capitoli precedenti abbiamo studiato dal punto di vista teorico quattro euristiche per il problema k-EuMST, valutando quale sia il fattore di approssimazione garantito da queste. In questo capitolo vengono riportati i risultati di una sperimentazione che ha lo scopo di mostrare le prestazioni delle quattro euristiche su un campione di casi concreti.

7.1 L'ambiente

Le simulazioni sul calcolatore sono state eseguite implementando le varie euristiche nel linguaggio di programmazione ANSI C. Questa scelta è stata sostenuta dal fatto che il linguaggio ideato da Brian Kernighan, rispetto a molti altri linguaggi ampiamente diffusi, presenta una notevole flessibilità e permette di scrivere codice molto efficiente. Il compilatore utilizzato è stato il GCC nella versione 2.96. Il sistema operativo sul quale sono state eseguite le simulazioni è Linux, con la distribuzione Red Hat 7.3. Il kernel del sistema

è il 2.4.18-3. La macchina sulla quale si sono svolti i test è un laptop della Toshiba con microprocessore Intel Pentium III da 1.0 Ghz, avente 256 Mbyte di memoria RAM. La potenza di calcolo della macchina a disposizione non è estremamente alta ma tuttavia, le prove che ci siamo posti di eseguire e le euristiche prese in esame sono state tali da permettere comunque uno studio significativo.

7.2 Le istanze utilizzate per i test

Per testare le nostre euristiche è stato necessario implementare una procedura che generasse istanze del problema k-EuMST. I punti dell'istanza sono stati generati secondo una distribuzione uniforme in un quadrato di lato unitario, ovvero prendendo, per ogni punto, il valore dell'ascissa e dell'ordinata in maniera equiprobabile nell'intervallo $[-0,5; 0,5]$. La procedura che genera i punti in maniera randomica fa uso del generatore di numeri (pseudo)casuali implementato come primitiva nel linguaggio C.

Al variare del numero n di punti dell'istanza il valore k dei punti che si vogliono collegare è stato fissato in tre modi diversi: prendendolo pari al logaritmo di n in un caso, pari $n/2$ in un altro ed infine assegnandogli un valore costante e indipendente da n.

Un altro fattore che ha guidato la scelta delle sperimentazioni da eseguire sono stati i tempi di calcolo. A causa dei lunghi tempi di calcolo è stato necessario, talvolta, limitare il numero delle istanze. Questo limite lo si è avvertito in particolare quando sono state effettuate sperimentazioni che richiedono l'utilizzo della procedura di forza bruta che calcola esaustivamente

la soluzione ottima. In questi casi è stato necessario limitarsi ad istanze con al più 20-30 punti. Ho stimato che una computazione con 50 punti con l'algoritmo di ricerca esaustiva avrebbe richiesto almeno un mese. Con le altre euristiche è stato possibile utilizzare istanze con un numero maggiore di punti, ma superati i 1000 anche le euristiche Ravi e Garg cominciano a richiedere tempi di calcolo onerosi. Per questo è stato necessario campionare l'intervallo nel quale prendiamo i valori di n per ridurre il numero delle istanze analizzate. Tra un'istanza e la successiva abbiamo introdotto un *gap* di tre valori. Questo ha permesso di ingrandire il nostro ambito di indagine.

Una volta effettuati i primi test è sorto un problema: i risultati degli esperimenti davano origine a grafici altamente oscillanti. Questo perché il costo della soluzione di un istanza del k-MST dipende direttamente, oltre che dai parametri n e k, dalla disposizione geometrica dei punti dell'istanza. La strategia adottata per ovviare a tale inconveniente è quella di prendere, per ogni valore di n e k, un campione di più istanze generate randomicamente e prendere la media dei costi delle soluzioni trovate. Questa tecnica ha aumentato la fluidità delle funzioni disegnate nei grafici. Ovviamente la grandezza del campione è altresì collegata ai tempi di calcolo. Da un corretto compromesso di tutti questi parametri sono stati ricavati i risultati sperimentali riportati in questo capitolo.

7.3 Le euristiche a confronto

In un primo grafico (vedi Figura 7.1) riportiamo i costi degli alberi prodotti dalle euristiche quando il numero n dei punti delle istanze utilizzate varia tra

20 e 100. Per questo test è stato utilizzato un valore di k costante pari a 20.

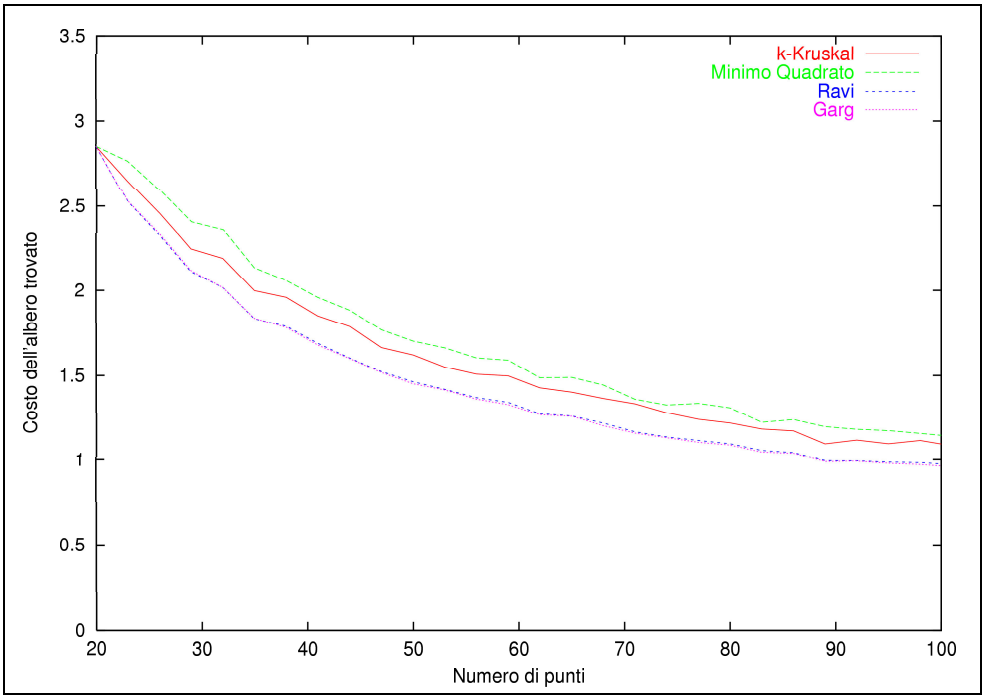

Figura 7.1: I risultati delle varie euristiche. Sperimentazioni eseguite con $k = 20$.

Quello che risalta da questo primo grafico è che le due euristiche più sofisticate, Ravi e Garg, si comportano molto bene e producono soluzioni di pari qualità. É interessante notare il vantaggio che l'euristica k-Kruskal presenta rispetto a Minimo Quadrato, a dispetto dei fattori di approssimazione analizzati teoricamente nei capitoli precedenti che avrebbero dovuto portare l'euristica Minimo Quadrato a prevalere sull'euristica k-Kruskal. Vedremo in seguito un'analisi più accurata di questo fenomeno. Bisogna dire però che i costi delle soluzioni delle euristiche k-Kruskal e Minimo Quadrato non sembrano scostarsi molto da quelle di Garg e Ravi al crescere del parametro

n. Dai risultati si può osservere che per un valore di k fissato, al crescere di n, il costo delle soluzioni prodotte è comunque decrescente per tutte le euristiche. Questo è dovuto al fatto che all'aumentare dei punti disposti nel piano è sempre più probabile collegarne uno stesso numero k con dispendio, in termini di lunghezza degli archi, minore.

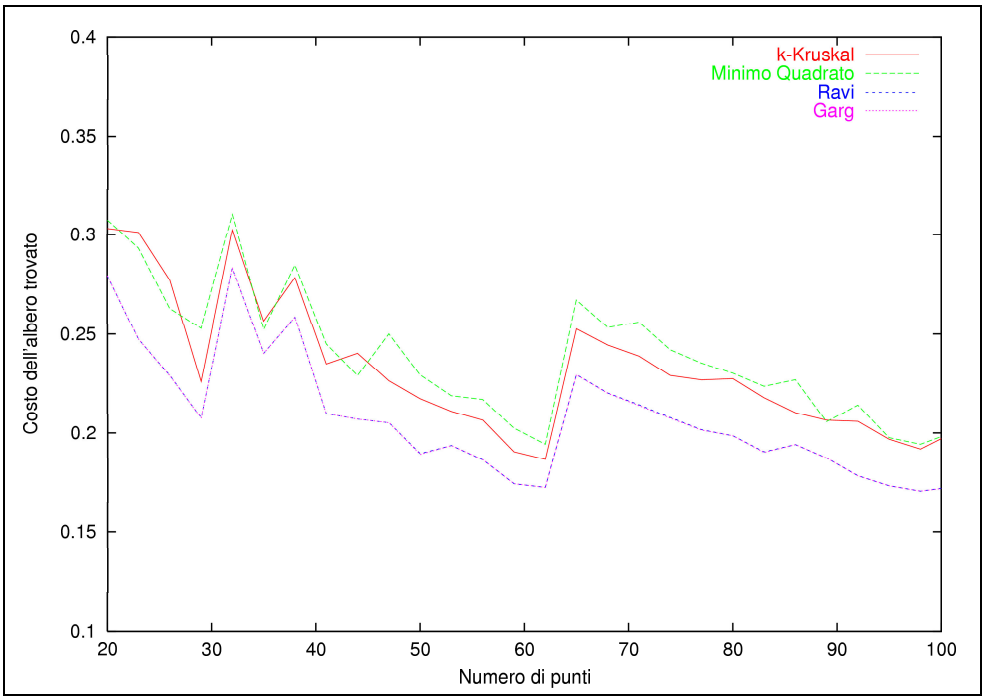

Figura 7.2: I risultati delle varie euristiche. Sperimentazioni eseguite con $k = \lfloor \log n \rfloor$.

Nel secondo grafico (Figura 7.2) il primo esperimento è stato ripetuto sostituendo al valore costante k il logaritmo del numero dei punti delle istanze (più precisamente k è la parte intera inferiore del logaritmo di n). Notiamo che il valore delle soluzioni sembra decrescere per tutte, ma si presenta con dei picchi in corrispondenza di alcuni valori di n, nonostante i valori rappre-

sentino la media di un gran numero di prove. Questo è dovuto al fatto che per il valore di n preso in un intervallo compreso tra 2^i e 2^{i+1} il valore di k risulta costante. Ad esempio per n compreso tra 32 e 64 si può apprezzare una curva riconducibile al grafico precedente. I picchi nel grafico si presentano, dunque, quando il valore di n diventa una potenza di due. Anche in questo grafico la qualità delle soluzioni di Garg e Ravi risultano nettamente superiori rispetto alla qualità delle soluzioni prodotte da k-Kruskal e Minimo Quadrato. Il costo delle soluzioni degli algoritmi Garg e Ravi sembra essere simile in quasi tutti i test, cosa che induce a pensare che il loro valore sia molto vicino a quello della soluzione ottima.

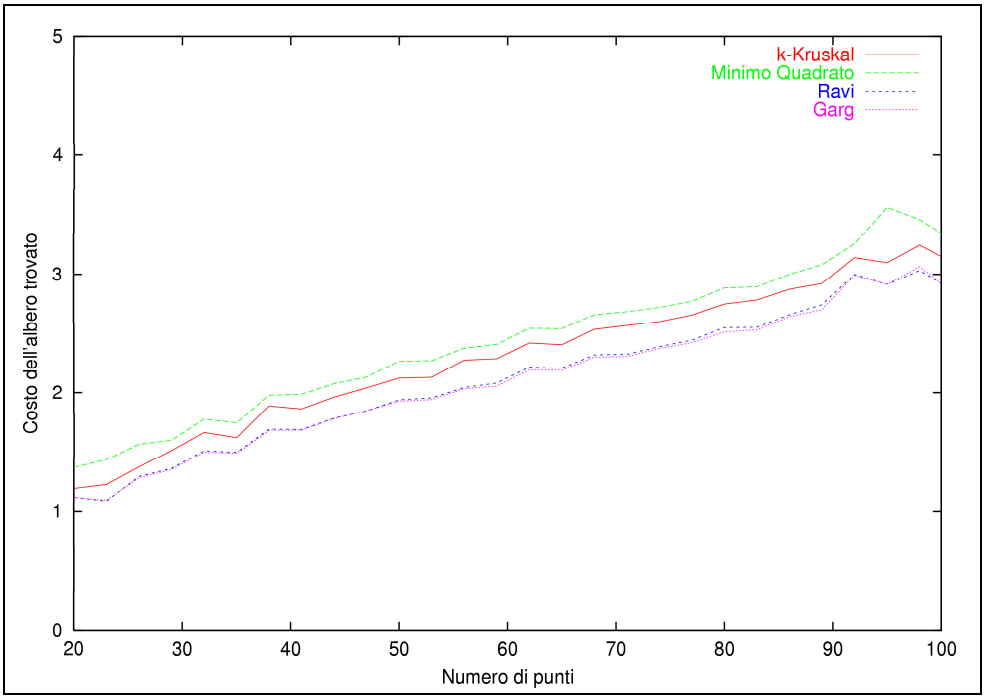

Figura 7.3: I risultati delle varie euristiche. Sperimentazioni eseguite con $k = \lfloor n/2 \rfloor$.

Nella Figura 7.3 l'esperimento è stato ripetuto ponendo k pari alla metà del numero di nodi dell'istanza. Si può vedere come la crescita di n e di k sia accompagnata dalla crescita delle soluzioni. Anche qui gli algoritmi Garg e Ravi producono delle soluzioni confrontabili e particolarmente buone, mentre le altre due euristiche soffrono.

Da quanto visto risulta strano che in tutti e tre gli scenari delle simulazioni abbiamo osservato che l'algoritmo k-Kruskal si comporta meglio dell'algoritmo Minimo Quadrato. Dalla nostra analisi teorica, però, abbiamo provato che l'euristica Minimo Quadrato porta come limite superiore del suo fattore di approssimazione $O(\sqrt{k})$ mentre per l'euristica k-Kruskal vale k. Inoltre è stato provato che la seconda, che può commettere un errore più grosso, raggiunge effettivamente quel tasso di errore. Allora sembra strano che la seconda, k-Kruskal, sperimentalmente dia soluzioni di migliore qualità. Questo può essere dovuto a due motivi, esaminiamo il primo. Poiché il limite superiore del fattore di approssimazione provato per le euristiche è, per la prima un valore esatto, $k - 1$, mentre per la seconda solamente un valore dello stesso ordine di grandezza di $\sqrt{k}$, è possibile che per Minimo Quadrato il limite superiore nasconda nella notazione O grande costanti moltiplicative e termini additivi minori piuttosto grandi. Questo implicherebbe che per un intervallo limitato di valori l'euristica Minimo Quadrato si comporti peggio dell'euristica k-Kruskal ed è possibile che la nostra sperimentazione sia stata effettuata proprio all'interno di questo intervallo.

Una seconda ipotesi è che, data la nostra particolare natura delle istanze utilizzate nella nostra sperimentazione (punti equamente distribuiti nel piano), gli algoritmi producano, al caso medio, delle soluzioni che risultino

migliori per k-Kruskal. Per capire di quale dei due motivi si tratti è stato effettuato un test *ad hoc* solo su queste due euristiche e su valori di n assai più grandi di 100. Questo è stato possibile perché le due euristiche in esame hanno tempi di esecuzione particolarmente bassi. Il risultato è riportato nel grafico 7.4.

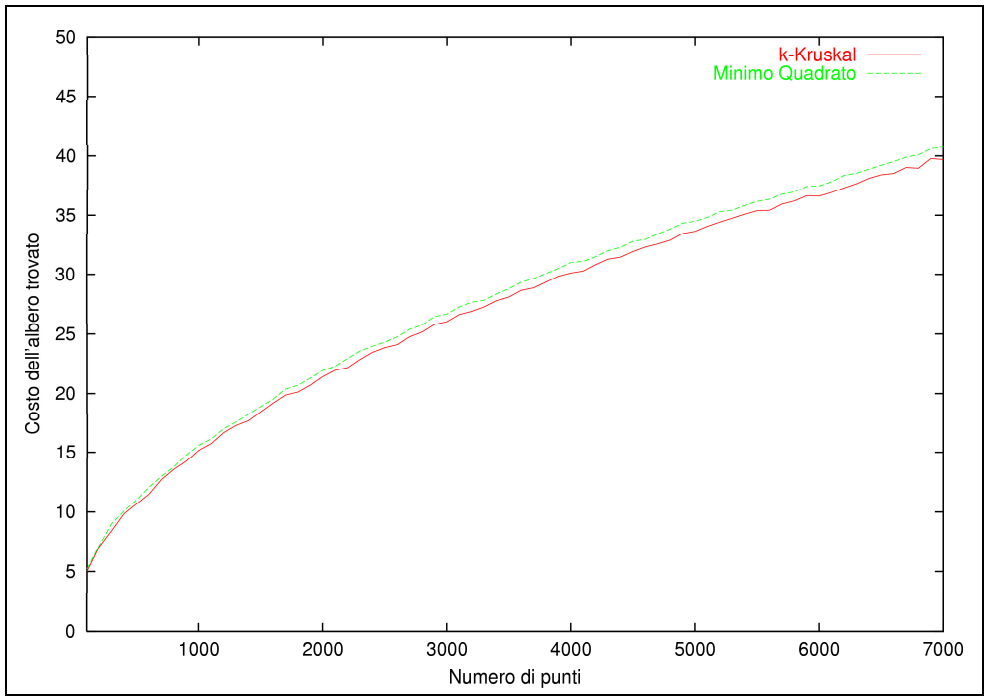

Figura 7.4: I risultati di k-Kruskal e Minimo Quadrato su istanze con molti punti. Sperimentazioni eseguite con $k = 3/4n$ con n che va da 20 a 7000.

Il grafico sembra avvalorare la seconda ipotesi. Per questo test si è scelto di provare istanze fino a 7000 punti. Il valore di k era impostato come $3/4 \cdot n$ e quindi arrivava fino a 5250. Se fosse vera la prima ipotesi, oltre un certo valore di k l'euristica Minimo Quadrato avrebbe prodotto soluzioni migliori di k-Kruskal ma fino 5000 questo fenomeno non si è verificato, anzi

lo scostamento delle soluzioni è andato crescendo. Più formalmente, se il rapporto di approssimazione di Minimo Quadrato è $O(\sqrt{k}) = \alpha\sqrt{k}$, il punto di intersezione delle due curve si sarebbe avuto all'incirca quando $\alpha\sqrt{k} = k$ e cioè $\alpha = \sqrt{k}$ ovvero quando $k = \alpha^2$. Da una stima grossolana il coefficiente moltiplicativo α che si troverebbe davanti al rapporto di approssimazione studiato per l'euristica Minimo Quadrato non dovrebbe essere superiore a 10 ma questo significherebbe che per valori di k superiori a 100 (vale a dire $n \geq 134$) la migliore qualità delle soluzioni prodotte dall'euristica Minimo Quadrato avrebbe dovuto cominciare a manifestarsi. Se per valori di k fino a 5250 non si è verificato è lecito aspettarsi che non si verificherà più e che il migliore rendimento di k-Kruskal sulla media delle istanze generate in queste simulazioni diventa la spiegazione più plausibile.

7.4 I tempi di calcolo

Nelle sperimentazioni bisogna fare i conti con il tempo che si ha a disposizione, soprattutto quando si devono analizzare algoritmi dalla complessità elevata. I tempi diventano esageratamente lunghi quando si devono, oltretutto, paragonare i risultati sperimentali con quelli esatti ottenuti da una procedura che ha una complessità esponenziale. Per questi motivi molte delle sperimentazioni sono state eseguite su istanze di dimensioni piuttosto limitate, rendendo talvolta diffioltosa l'interpretazione dei risultati.

Avendo a che fare con pochi risultati ci si potrebbe fare un idea sbagliata dell'andamento di una euristica. É dunque corretto prendere con le pinze i valori ottenuti su un insieme limitato di istanze, ma converrebbe considerarli

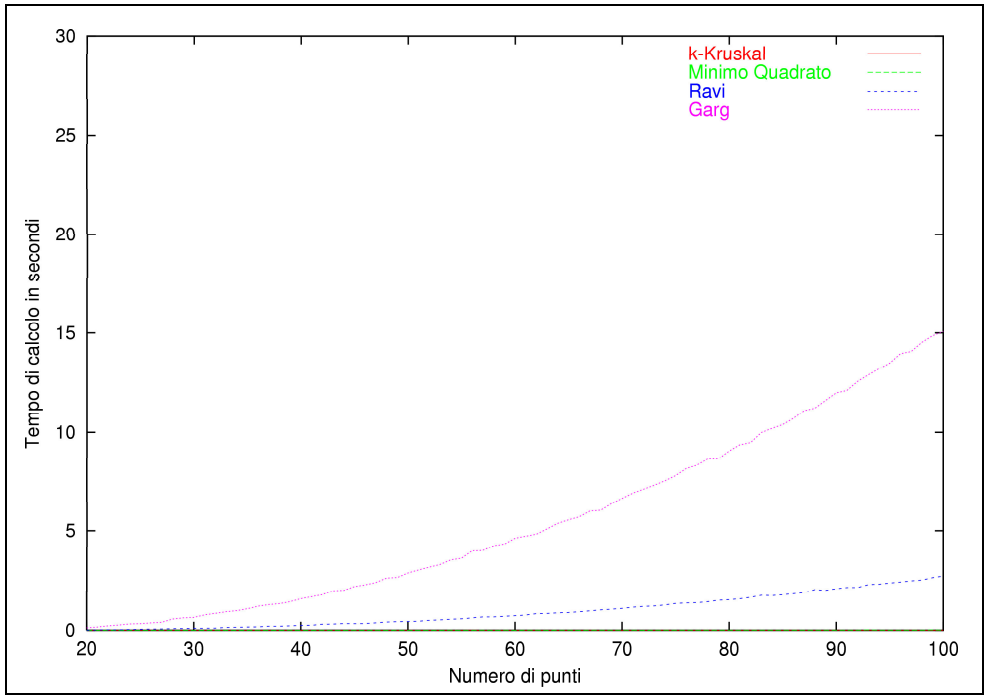

Figura 7.5: I tempi delle varie euristiche. Sperimentazioni eseguite con $k = 20$.

solamente per un riscontro globale. Nei prossimi grafici verranno mostrati i tempi di calcolo delle quattro euristiche prese in esame in questo documento.

La Figura 7.5 evidenzia come l'algoritmo Garg per istanze con 100 punti dove se ne vogliono collegare 20 richiede un tempo di circa 15 secondi. Con gli stessi parametri l'algoritmo Ravi ne impiega meno di 5. I tempi di esecuzione delle euristiche k-Kruskal e Minimo Quadrato sono pressoché nulli. Nel grafico in Figura 7.6 sono riportati i tempi di calcolo quando il parametro k è uguale al logaritmo di n. Anche in questo caso la differenza di complessità delle euristiche incide sui tempi di calcolo. Vediamo, anche se in maniera meno marcata, che le euristiche più sofisticate hanno tempi più lunghi. Per

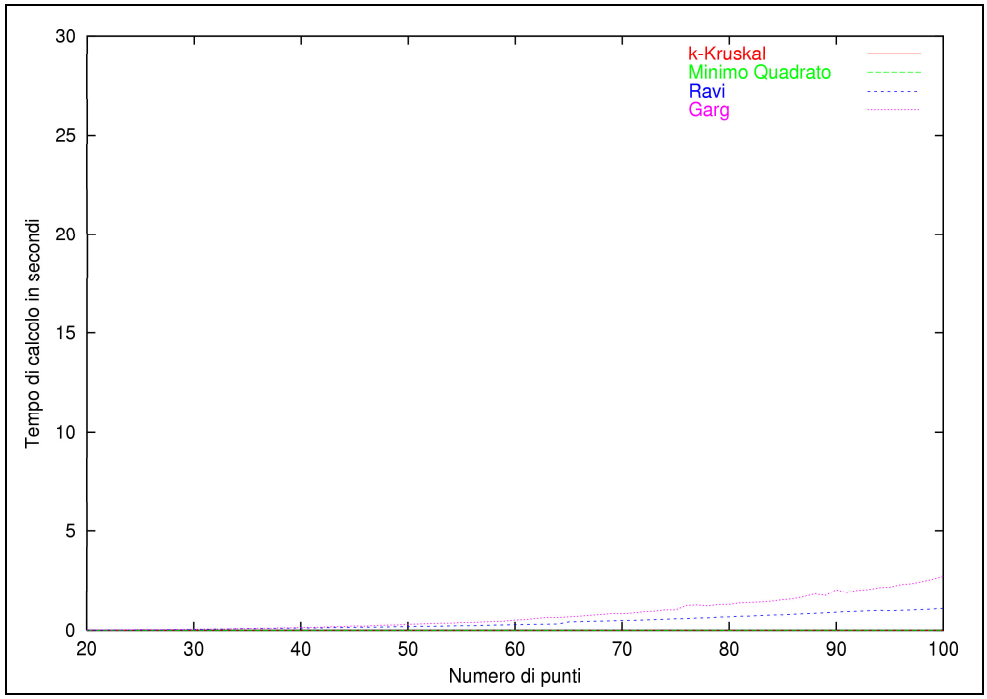

Figura 7.6: I tempi delle varie euristiche. Sperimentazioni eseguite con $k = \lfloor \log n \rfloor$.

valutare i tempi di esecuzione dei vari algoritmi si è fatto uso di un indagine a campione è poi si fatta la media dei valori. Questo perché il tempo di calcolo di un algoritmo, anche se indipendente dalla forma dell'istanza sul quale è lanciato, è strettamente legato a parametri interni della macchina sul quale gira e del sistema operativo che lo gestisce. Dato che la macchina a disposizione non è stata progettata per effettuare delle sperimentazioni pure ma per un uso casalingo, i risultati dei test sui tempi potrebbero essere influenzati dall'interferenza di altre applicazioni. A tal proposito si è cercato di adottare il maggior numero di precauzioni possibili cercando di evitare il caricamento di task diversi dall'algoritmo da testare.

Nel terzo grafico, Figura 7.7, si vede come l'algoritmo Garg presenta subito una forte impennata dovuta alla sua notevole complessità e al fatto che in questo test abbiamo preso $k = n/2$.

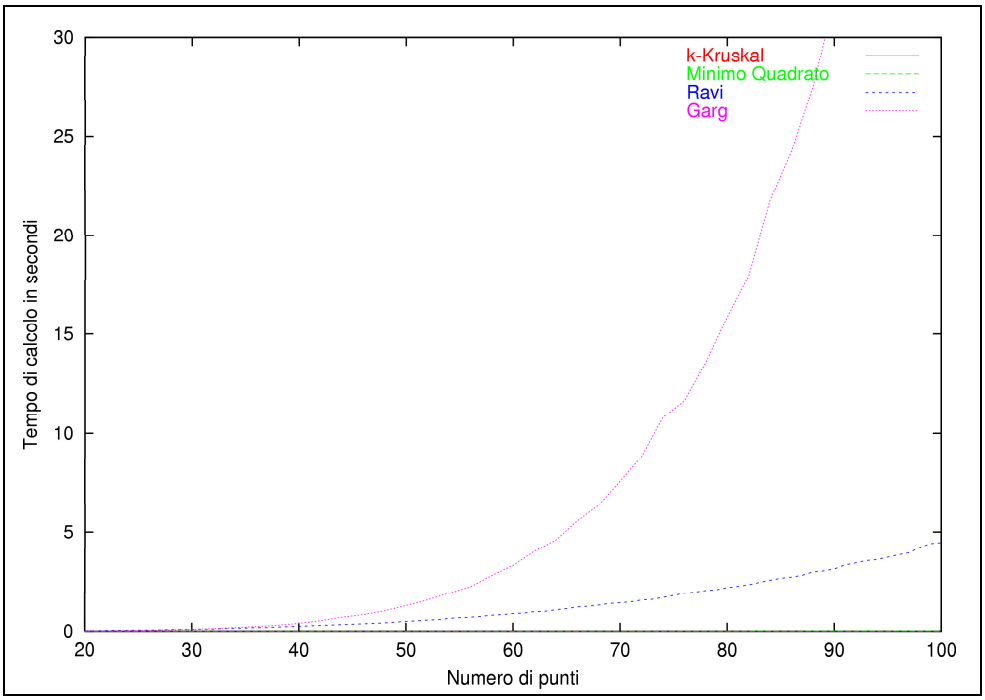

Figura 7.7: I tempi delle varie euristiche. Sperimentazioni eseguite con $k = n/2$.

Siccome ci sono notevoli variazioni tra le pendenze delle varie curve è stato difficile rappresentarle tutte in unico grafico. La scala scelta non permette di comprendere a fondo quanto cresca il tempo di calcolo dell'euristica Garg. Per questo motivo abbiamo ritenuto utile riportare nella Tabella 7.1 i valori di questa funzione. Per scelta di brevità nella tabella sono riportati solo pochi valori, uno ogni dieci.

Poiché dai precedenti grafici sui tempi di calcolo le euristiche più velo-

n	Tempo (sec.)
20	0,0175
30	0,1025
40	0,3925
50	1,28
60	3,3
70	7,59
80	15,85
90	31,8875
100	214,68

Tabella 7.1: I tempi di calcolo dell'algoritmo Garg.

ci, k-Kruskal e Minimo Quadrato, hanno mostrato dei tempi estremamente bassi, per apprezzarne la differenza è stata effettuato un esperimento che coinvolge l'esecuzione di queste due sole euristiche utilizzando istanze con un numero di nodi molto più elevato. Nella Figura 7.8 si trova un scala dei tempi molto ingrandita, che va da 0 a 4 secondi solamente. Le prove sono state effettuate con il valore k costante uguale a 20. Nonostante siano state testate con istanze grandi 2500 punti le due euristiche hanno mostrato tempi estremamente ridotti. Questo grazie alle loro complessità. Ricordando, k-Kruskal ha un costo computazionale di $O(n \log n)$ mentre Minimo Quadrato ha un costo di $O(n \log^3 n)$. Tutte e due con migliaia di nodi richiedono pochi secondi di calcolo. Arrivati a istanze con decine di migliaia di nodi il tempo di esecuzione dell'algoritmo Minimo Quadrato diventa insostenibile.

L'algoritmo k-Kruskal ha mostrato rapidi tempi di calcolo anche su istanze con milioni di nodi.

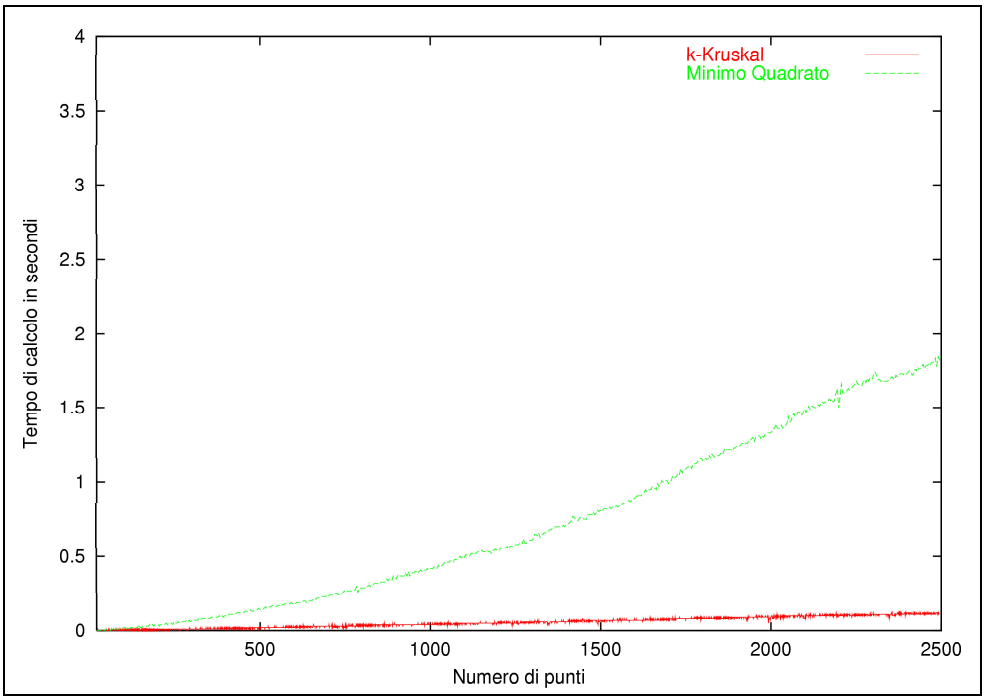

Figura 7.8: I tempi delle euristiche k-Kruskal e Minimo Quadrato a confronto. Sperimentazioni eseguite con $k = 20$. Osservare che il valore della scala dei tempi è tra 0 e 4 secondi.

7.5 I risultati sperimentali con un fissato numero di punti

Nelle precedenti simulazioni abbiamo dato un valore più importante al parametro n, il numero di punti da generare per le istanze random. Per il valore k

dei punti da collegare sono state adottate tre opzioni, in modo da collegarlo al parmetro n oppure lasciandolo fisso. Per completezza è giusto verificare un ulteriore caso, nel quale il numero di punti generati sia un parametro fisso e il valore dei punti da connettere sia variabile su un intervallo compreso tra 2 e n. In questa situazione potrebbero verificarsi nuovi fenomeni da analizzare.

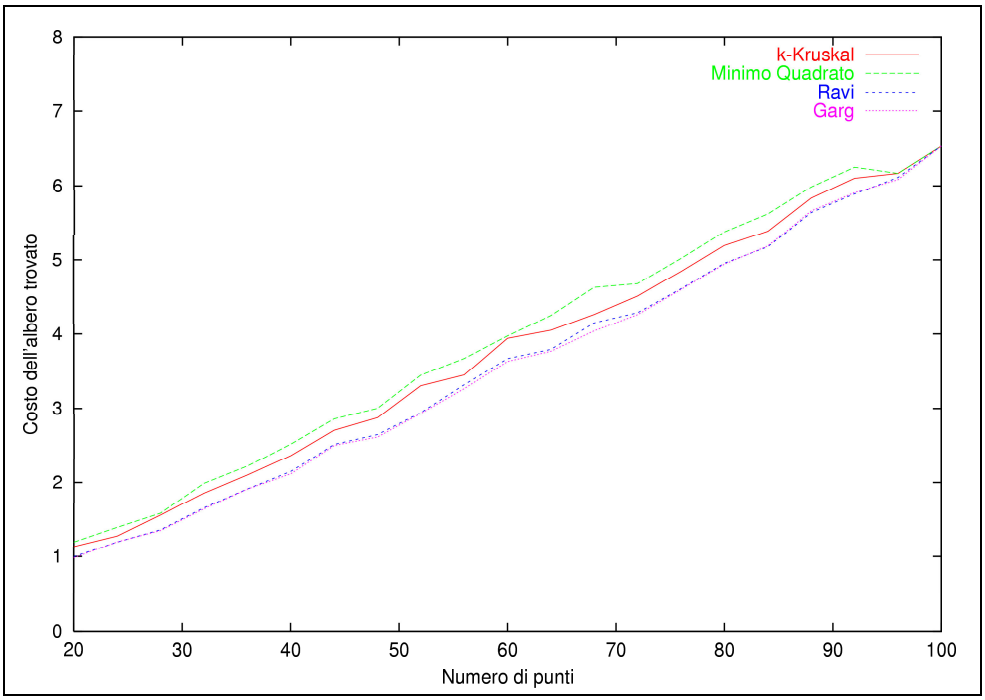

Figura 7.9: Le quattro euristiche a confronto. Sperimentazioni eseguite con $n = 100$ in funzione di k variabile tra 20 e 100.

Come per i precedenti test i lunghi tempi di esecuzione hanno impedito le prove su grandi istanze. In Figura 7.9 il massimo numero di punti da analizzare consentito dalle sperimentazioni è stato 100. Si può osservare come ci sia una differenza nei valori delle soluzioni dei vari algoritmi nella parte centrale del grafico. Ovviamente per valori di k molto piccoli o vicini

a n le euristiche danno tutte soluzioni simili e vicine all'ottimo. Questo fenomeno si verifica perché è banale prendere un albero di copertura su 2 punti, basta prendere l'arco di costo minimo e cioè la coppia di punti più vicini. Se invece $k = n$ allora le euristiche si uniformano al calcolo dell'albero di copertura di costo minimo su tutti i punti.

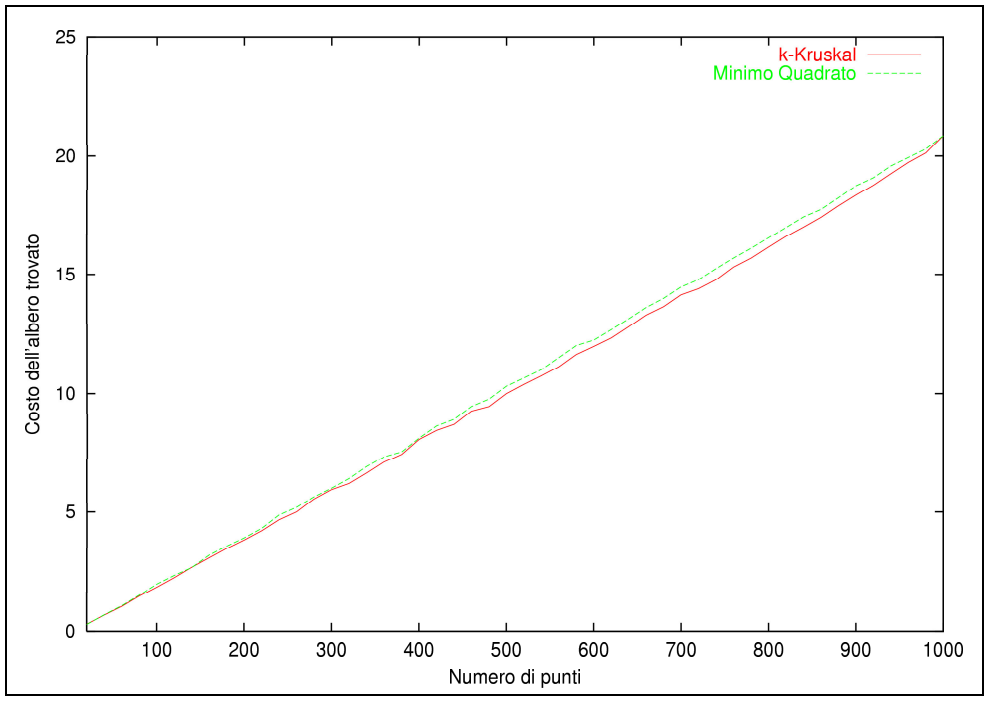

Figura 7.10: Le euristiche k-Kruskal e Minimo Quadrato a confronto. Sperimentazioni eseguite con $n = 1000$ in funzione di k variabile tra 20 e 1000.

Per avere un idea più precisa dell'andamento di questo grafico è stata fatta, per le euristiche più veloci, una prova ausiliare su istanze con 1000 punti. Nella Figura 7.10 vediamo un gafico assai più smussato per le euristiche k-Kruskal e Minimo Quadrato. Per questo test sono stati fatti, per ogni valore

di k, dei campioni su centinaia di istanze. Da questo grafico risulta apprezzabile il punto dove si verifica il maggior divario tra le soluzioni prodotte dalle due euristiche. In prossimità del valore 750, che corrisponde ad un valore $k = 3/4 \cdot n$, si nota la massima distanza tra le due funzioni rappresentate. Su entrambi i grafici si nota comunque un andamento crescente dei valori delle soluzioni. Questo, analiticamente, può essere spiegato dal fatto che per un dato numero di punti, aumentando la quantità di punti da collegare, linearmente cresce il costo delle connessioni per l'albero di copertura.

7.6 Le sperimentalzioni in confronto con le soluzioni esatte

In questa sezione si trovano le sperimentazioni che permettono di valutare effettivamente la bontà delle soluzioni prodotte dalle euristiche. Possiamo apprezzare, nei grafici, la differenza che si trova tra i costi delle soluzioni prodotte dalle euristiche e la soluzione ottima. In questi test per il calcolo della soluzione esatta delle istanze in esame abbiamo utilizzato un algoritmo di ricerca esaustiva con complessità computazionale esponenziale. A causa dei tempi elevati richiesti dall'algoritmo di forza bruta i campioni utilizzati per il calcolo della media sono piuttosto ridotti. Nonostante questo si possono comunque apprezzare buoni risultati. I valori riportati dalle curve indicano il costo additivo delle soluzioni prodotte dalle euristiche rispetto all'ottimo, quindi una misura dello sbaglio che l'euristica ha commesso nel calcolare l'albero di copertura.

Da tutti e tre i grafici, anche questa volta, si possono trarre le stesse

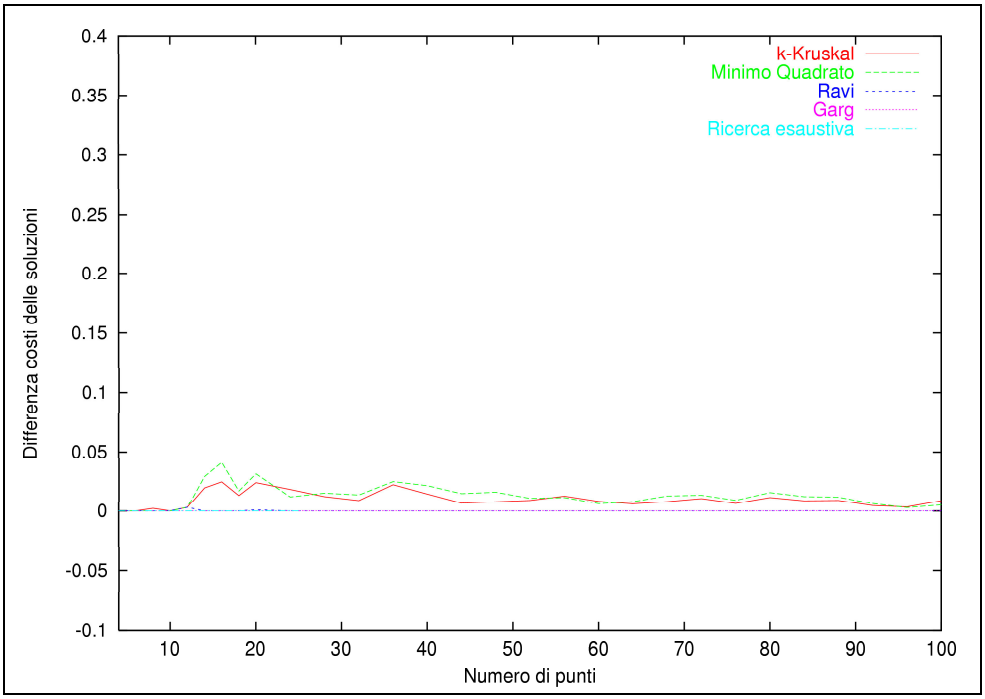

Figura 7.11: Il costo aggiuntivo richiesto dalle soluzioni prodotte dalle quattro euristiche rispetto al costo della soluzione ottima, n compreso tra 4 e 100, $k = 4$.

conclusioni che sono venute fuori dai precedenti test. Gli algoritmi Minimo Quadrato e k-Kruskal mostrano un rendimento inferiore rispetto alle due euristiche più sofisticate. Le euristiche Garg e Ravi, come da ipotesi, sono praticamente aderenti in tutti i casi alla soluzione esatta. Anche in queste simulazioni, inoltre, si osserva il vantaggio che l'algoritmo k-Kruskal ha nei confronti di Minimo Quadrato.

Nei tre grafici in Figura 7.11, Figura 7.12 e Figura 7.13 abbiamo usato uno stesso intervallo di valori sull'asse delle ordinate ma differenti intervalli sull'asse delle ascisse. La scelta di mantenere gli stessi valori sull'asse Y

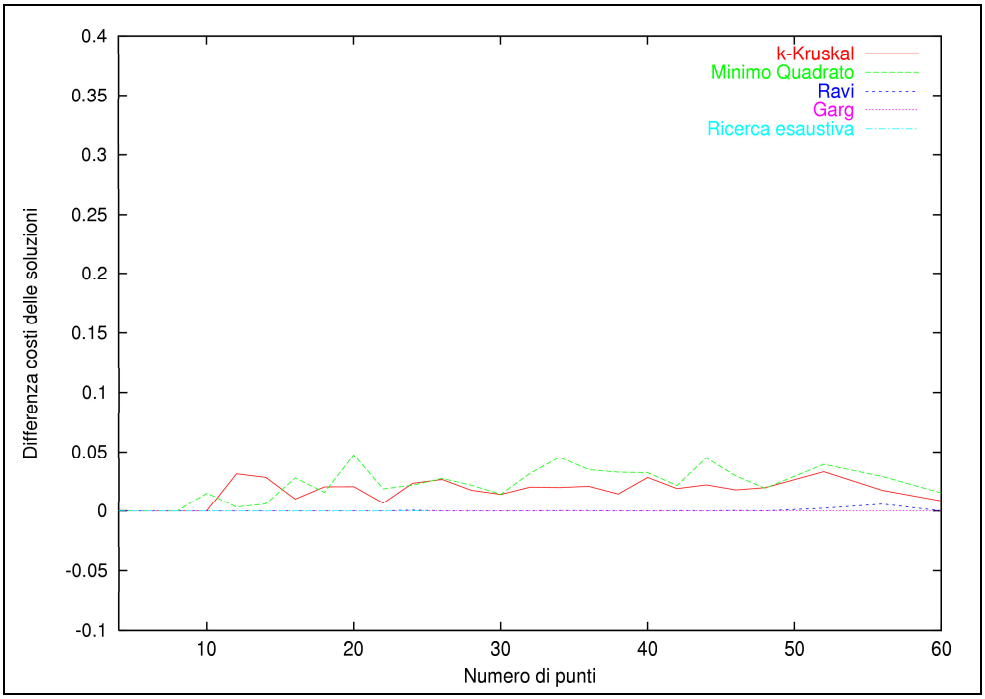

Figura 7.12: Il costo aggiuntivo richiesto dalle soluzioni prodotte dalle quattro euristiche rispetto al costo della soluzione ottima, n compreso tra 4 e 60, $k = \lfloor \log n \rfloor$.

permette un confronto tra i tre esperimenti e valutare in quale dei tre si sono ottenuti i risultati migliori. Per quanto riguarda l'intervallo di valori riportato sull'asse X, siccome il tempo di esecuzione aumenta al crescere di k, non è stato possibile eseguire tutti e tre gli esperimenti su gli stessi valori. Infatti per il primo esperimento, poiché k aveva un valore costante piccolo è stato possibile analizzare istanze con 100 punti. Nel secondo esperimento abbiamo preso $k = \lfloor \log n \rfloor$, e siamo arrivati ad analizzare istanze con massimo 60 punti. Nel terzo esperimento, dove $k = \lfloor n/2 \rfloor$, non è stato possibile andare oltre i 25 punti per istanza.

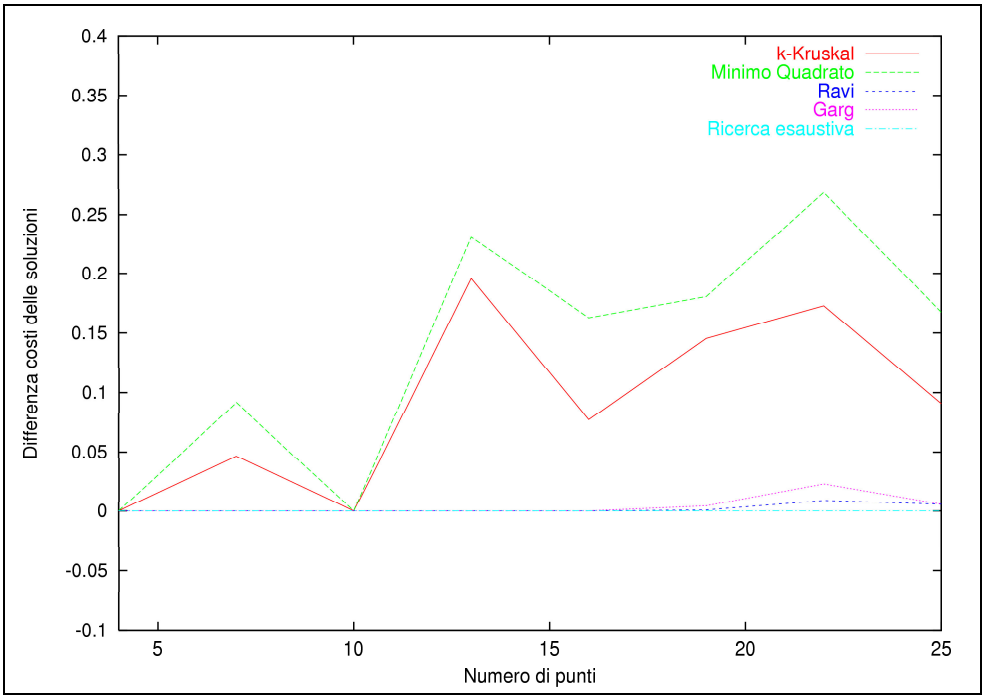

Figura 7.13: Il costo aggiuntivo richiesto dalle soluzioni prodotte dalle quattro euristiche rispetto al costo della soluzione ottima, n compreso tra 4 e 25, $k = n/2$.

Un'altra riflessione derivante da queste ultime simulazioni è che l'errore prodotto nel calcolo della soluzione delle euristiche è in accordo con la crescita di k. Mentre nel primo grafico si può apprezzare un errore abbastanza limitato, nel secondo e soprattutto nel terzo l'errore va via via crescendo. Anche per le due euristiche più performanti, nel terzo scenario (Figura 7.13), gia su soli 23 nodi l'errore è notevole.

7.7 Sviluppi futuri

Poiché il tempo a disposizione è limitato e la curiosità no, restano aperti molti interrogativi che possono essere punti di partenza per uno sviluppo del mio lavoro.

Un primo spunto nasce dall'esigenza di migliorare le euristiche. A parte l'euristica Garg, in tutte le altre si è fatto uso, per il calcolo dell'albero di copertura, di una procedura di potatura di archi nel caso ci siano più di k punti collegati. La procedura qui utilizzata per la potatura dell'albero è una banale visita in profondità dei nodi dell'albero che non tiene conto, eventualmente, delle lunghezze degli archi da tagliare, ma si limita a restituire il sottoalbero costituito dai primi k nodi visitati. Un significativo miglioramento potrebbe aversi con l'utilizzo di un funzione di potatura più "furba" come ad esempio quella che elimina nodi foglia dell'albero fino a che restino esattamente k nodi e, nel selezionare il nodo da eliminare, sceglie sempre quello collegato al padre da un arco di lunghezza massima. Una funzione di potatura "furba" potrebbe mostrare migliorie non banali come quella utilizzata in [19] che ha portato al valore minimo del fattore di approssimazione per il problema k-MST nel caso generale.

Sarebbe altrettanto interessante studiare il comportamento di queste euristiche su istanze che non siano caratterizzate dall'avere i punti del piano equidistribuiti. Addensamenti di punti potrebbero variare di molto i risultati delle sperimentazioni e cambiare la nostra opinione sulle euristiche. Un altro studio interessante sarebbe un analisi teorica del fattore di approssimazione che tenga conto dell'equidistribuzione dei punti.

Sotto molti aspetti il più importante approfondimento in questo ambito

penso, a ragione, che rimanga lo studio delle tecniche che hanno garantito nell'ultimo periodo i migliori risultati su problemi euclidei come il nostro k-EuMST. Mi riferisco alla tecnica della ghigliottina [24] e al metodo di clusterizzazione di Goemans e Williamson [17]. Il mio interesse nei confronti di queste metodologie è sostenuto da tre motivi: per prima cosa senza di queste sarebbe stato impossibile far scendere nella classe di approssimazione APX i problemi al quale si applicano. Come seconda cosa, si sono dimostrate delle tecniche polivalenti, applicandosi a molte casistiche di problemi, cosa che a mio parere dimostra un grande senso di creatività. Come terza e ultima motivazione, direi che queste tecniche presentano un elevato costo computazionale che sarebbe utile tentare di abbattere per renderle effettivamente competitive con le euristiche più semplici.

Bibliografia

[1] J. B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proc. American Mathematical Society, 7(1), pp. 48–50, 1956.

[2] R.C. Prim, *Shortest connection networks and some generalizations*, Bell System Tech Journal, 36(6), pp. 1389–1401, 1957.

[3] R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz and S. S. Ravi, *Spanning trees short or small*, SIAM Journal on Discrete Mathematics, 9(2), pp. 178–200, 1996.

[4] E. W. Dijkstra, *A note on two problems in connection with graphs*, Numerische Mathemetik, 1, pp. 269–271, 1959.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1997.

[6] T. H. Cormen, C. E. Leierson, R. L. Rivest, *Introduzione agli algoritmi*, volume 2, pp. 432–436, Jackson Libri, 1994.

[7] C. Demetrescu, I. Finocchi, G. Italiano, *Algoritmi e strutture dati*, pp. 380–383, McGraw-Hill, 2004.

[8] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1990.

[9] F. R. Preparata, M. I. Shamos, *Computational Geometry: An Introduction*. New York: Springer-Verlag, 1985.

[10] A. Zelikovsky, D. D. Lozovanu,*Minimal and bounded trees*, Tezele Cong. XVIII Acad. Romano-Americane, Kishinev, pp. 25–26, 1993.

[11] M. Fischetti, H. W. Hamacher, K. Jørnsten, F. Maffioli, *Weighted k-cardinality trees: complexity and polyhedral structure*, Networks, 24, pp. 11–21, 1994.

[12] B. Awerbuch, Y. Azar, A. Blum, S. Vempala, *Improved approximation guarantees for minimum-wheight k-trees and prize-collecting salesmen*, Proc. 27th Annual ACM Symposium on Theory of Computing, pp. 277–283, 1995.

[13] S. Rajagopalan, V. Vazirani, *Logarithmic approximation of minimum weight k trees*, Manuscript, 1995.

[14] A. Blum, R. Ravi, S. Vempala, *A costant-factor approximation algorithm for the k-MST problem*, Proc. 28th Annual ACM Symposium on Theory of Computing, pp. 442–448, 1996.

[15] A. Blum, P. Chalasani, S. Vempala, *A costant-factor approximation algorithm for the k-MST problem in the plane*, Proc. 27th Annual ACM Symposium on Theory of Computing, pp. 294–302, 1995.

[16] J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-MST problem*, Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 402–408, 1996.

[17] M. X. Goemans, D. P. Williamson, *A General Approximation Technique For Constrained Forest Problems*, SIAM Jurnal on Computing, 24, pp. 296–317, 1995.

[18] N. Garg, *A 3-approximation for the minimum tree spanning k vertices*, Proc. 37th Annual Symposium on Fundations of Computer Science, pp. 302–309, 1996.

[19] S. Arya, H. Ramesh, *A 2.5 factor approximation algorithm for the k-MST problem*, Inform. Proc. Letters, 65(3), pp. 117–118, 1998.

[20] N. Garg, D. S. Hochbaum, *An $O(\log k)$ approximation for the k minimum spanning tree problem in the plane*, Proc. 26th Annual ACM Symposium on Theory of Computing, pp. 432–438, 1994.

[21] C. Mata, J. S. B. Mitchell, *Approximation algorithms for geometric tour and network design problems*, Proc. 11th Annual ACM Symposium on Computational Geometry, pp. 360–369, 1995.

[22] S. Y. Cheung, A. Kumar, *Efficient quorumcast routing algorithms*, Proc. IEEE INFOCOM '94 Conference on Computer Communications, 2, pp. 840–847, 1994.

[23] D. Eppstein, *Faster geometric k-point MST approximation*, Tech. Report 95-13, Dept. Inform. Computer Science, Univerisity of California, Irvine, CA, 1995.

[24] J. S. B. Mitchell, *Guillotine Subdivisions Approximate Polygonal Subdivisions: A Simple Polynomial-Time Approximation Scheme for Geometric TSP, k-MST, and Related Problems*, SIAM Journal on Computing, 28(4), pp. 1298-1309, 1999.

[25] S. Arora, *Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and other Geometric Problems*, Journal of ACM, 45(5), pp. 753–782, 1998.

[26] M. Sipser, *Introduction to the Theory of Computation*, PWS Publishing, Boston, 1997.

[27] M. Agrawal, N. Kayal, N. Saxena, *PRIMES is in P*, Manuscript, 2002.

[28] D. Z. Du, F. K. Hwang, *A proof of the Gilbert-Pollak conjecture on the Steiner ratio*, Algorithmica, 7, pp. 121–135, 1992.