UNIVERSITÀ DEGLI STUDI DELL'AQUILA
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Analisi sperimentale di algoritmi approssimati per il problema del commesso viaggiatore in grafi fortemente metrici

Relatore:
Prof. Guido Proietti

Co-Relatore:
Prof. Luca Forlizzi

Candidato:
Alessio Sclocco

Anno Accademico 2007/2008

Indice

1. Introduzione

Il problema del commesso viaggiatore, in seguito TSP (dall'inglese Traveling Salesman Problem), è uno dei più interessanti e studiati problemi di ottimizzazione combinatoria.

In questa tesi analizzeremo sperimentalmente, dopo averli introdotti dal punto di vista teorico, il comportamento di quattro algoritmi α-approssimati per il TSP su istanze fortemente metriche.

Precondizioni a questo lavoro di analisi sperimentale sono state l'implementazione in C++ degli algoritmi α-approssimati da analizzare, di un generatore di grafi fortemente metrici casuali e la prima stesura di una libreria che facilitasse il lavoro di test ed eventuali future implementazioni di nuovi algoritmi.

1.1 Contenuti

Il capitolo 2 presenta un'introduzione a quello che è il problema del commesso viaggiatore ed introduce i concetti di TSP metrico e disuguaglianza triangolare.

Il capitolo 3 introduce il concetto di algoritmo approssimato e mostra alcuni risultati riguardanti l'approssimazione nel caso di TSP metrico e β-metrico. Vengono inoltre descritti gli algoritmi implementati in questo lavoro di tesi.

Il capitolo 4 mostra come si è svolta l'attività implementativa introducendo inoltre le scelte, le librerie utilizzate e gli artefatti software prodotti.

Il capitolo 5 presenta una somma dei risultati sperimentali i cui dati completi sono disponibili nell'appendice 1.

Il capitolo 6 contiene le conclusioni relative all'analisi dei risultati sperimentali.

Il capitolo 7 è dedicato alla bibliografia.

L'appendice 2 contiene l'upper bound del fattore di approssimazione per gli algoritmi di cui al capitolo 3 calcolati per i valori di β presi in considerazione; l'appendice 3 contiene i risultati sperimentali su alcune istanze della libreria TSPLIB95 e l'appendice 4 su istanze casuali generate da spazi Lp.

2. Il problema del commesso viaggiatore

Dato un insieme di città ed i costi per spostarsi dall'una all'altra relativi ad ogni coppia di esse, il problema del commesso viaggiatore, in seguito TSP, consiste nel trovare la via meno costosa per visitare tutte le città esattamente una volta e tornare infine al punto di partenza.

Più formalmente, dato un grafo completo e indiretto $G=(V,E)$ con una funzione di costo $c: E \to \mathbb{R}^+$, il TSP consiste nel trovare il ciclo hamiltoniano di costo minimo, dove per ciclo hamiltoniano di costo minimo intendiamo il minimo ciclo semplice che contiene ciascun vertice di V.

Il TSP è certamente uno dei più famosi e più studiati problemi di ottimizzazione combinatoria, oltre ad essere uno dei problemi con implicazioni pratiche più rilevanti; sfortunatamente anche essendo un problema decidibile è dimostrato essere NP-completo [1] e perciò risolvibile in tempo polinomiale solo nel caso poco probabile che P=NP.

Tra i motivi che hanno reso il TSP così interessante vi è la possibilità di dividere i suoi possibili input in classi; questa suddivisione in classi ha portato ad un fiorire di studi focalizzati sull'una o sull'altra, studi che hanno prodotto risultati molto interessanti ed alla base anche di questo lavoro di tesi.

Per questa tesi, infatti, non prenderemo in considerazione il TSP generale, così come definito precedentemente, ma una restrizione di esso ad una delle sue classi di input che chiameremo TSP β-metrico; le istanze che prenderemo in esame infatti rispetteranno la β-disuguaglianza triangolare:

$$c(\{u,v\}) \leq \beta(c(\{u,x\})+c(\{x,v\})) \; \forall u,v,x \in V \quad .$$

In particolare prenderemo in esame una sottoclasse del TSP β-metrico; non esamineremo infatti tutti i possibili valori di β ma ci dedicheremo solo alle istanze tali che $\frac{1}{2} \leq \beta < 1$, istanze che definiamo fortemente metriche; in corrispondenza di queste istanze parleremo di TSP fortemente β-metrico.

Tralasceremo quindi, ai fini del nostro lavoro, le istanze per cui $\beta \geq 1$, in corrispondenza delle quali parleremo di TSP debolmente β-metrico; nel seguito useremo il termine TSP metrico come sinonimo di TSP β-metrico nel caso in cui $\beta = 1$.

Intuitivamente possiamo già affermare che il nostro interesse per le istanze fortemente β-metriche nasce dalla proprietà di queste per cui il percorso più economico tra due nodi risulta essere sempre quello diretto.

Vedremo successivamente come questa particolarità verrà sfruttata dagli algoritmi presi in esame.

3. Gli algoritmi

Come detto al capitolo 2 il TSP è un problema NP-completo ed è perciò poco probabile che si riesca a trovare un algoritmo tempo polinomiale per la sua risoluzione; è tuttavia, come abbiamo precedentemente scritto e come il lettore potrà facilmente immaginare anche dalla sola descrizione informale, uno dei problemi di ottimizzazione combinatoria più importanti e per questo più studiati; proprio per questo motivo sono stati sviluppati per esso algoritmi "*veloci*" in grado di restituire soluzioni non troppo lontane dall'ottimo, ovvero quelli che chiamiamo algoritmi approssimati.

Più formalmente diciamo che un algoritmo A per un problema di ottimizzazione è approssimato con fattore di approssimazione pari ad α se per ogni input di dimensione n, il costo C della soluzione prodotta da A si discosta al più di un fattore α dal costo C^* di una soluzione ottima, ovvero $max(\{\frac{C}{C^*}, \frac{C^*}{C}\}) \leq \alpha$ per ogni soluzione prodotta da A; in questo caso diciamo che A è α-approssimato [1].

Sfortunatamente non è possibile trovare buoni algoritmi approssimati per istanze generali del TSP se P≠NP e $\alpha \geq 1$ [1], è tuttavia possibile trovarne di buoni per il TSP metrico, per il quale sono conosciuti diversi algoritmi α-approssimati come ad esempio l'algoritmo di Christofides o il Double MST.

Interessante, ai fini del nostro lavoro, è il miglioramento di α che si ottiene nel caso un algoritmo α-approssimato per il TSP metrico prenda in input una istanza fortemente β-metrica, così come dimostrato dal seguente teorema:

Teorema: Sia A un algoritmo di approssimazione per il TSP metrico con fattore di approssimazione pari ad α e sia $\frac{1}{2} \leq \beta < 1$, allora A è un algoritmo di approssimazione per il TSP fortemente β-metrico con fattore di approssimazione pari a $\frac{\alpha \cdot \beta^2}{\beta^2 + (\alpha - 1)(1 - \beta)^2}$ [2].

Proprio l'esistenza di questo teorema rende interessante per il nostro lavoro di analisi sperimentale analizzare anche algoritmi, come i precedentemente citati Double MST e Christofides, inizialmente sviluppati come approssimazioni per il TSP metrico e dai quali ci aspettiamo un miglioramento in caso di istanze di TSP fortemente β-metriche come quelle che prenderemo in esame.

Gli algoritmi α-approssimati che verranno quindi analizzati in questo lavoro di tesi sono: Double MST, Refined Double MST, Christofides e Cycle Cover.

Nel seguito del capito li prenderemo in considerazione singolarmente evidenziando per ognuno di essi rapporto di approssimazione, struttura ed eventuali peculiarità.

3.1 Double MST (2MST)

Il Double MST è un algoritmo 2-approssimato per il TSP metrico che diventa un algoritmo $\frac{2\beta^2}{2\beta^2-2\beta+1}$ -approssimato nel caso di TSP fortemente β-metrico [3].

La struttura del Double MST è la seguente:

Input: un grafo indiretto completo $G=(V,E)$ con funzione di costo $c: E \to \mathbb{R}^+$ tale che soddisfi la β-disuguaglianza triangolare rafforzata.

1) Costruire un minimo albero ricoprente (MST) T di G ;
2) Eseguire una visita in profondità su T partendo da un vertice arbitrario v_0 ;
3) Sia $v_0, v_1, \dots, v_{n-1}$ la sequenza restituita dalla visita in profondità, allora restituire in output la sequenza $v_0, v_1, \dots, v_{n-1}, v_0$.

Un albero ricoprente, ST (Spanning Tree), T di un grafo G connesso ed indiretto è un sottografo T di G indiretto, aciclico e connesso che copre tutti i vertici di G .
Il minimo albero ricoprente, MST (Minimum Spanning Tree), è l'albero ricoprente T di G di peso minimo.

3.2 Refined Double MST (R2MST)

Il Refined Double MST è un algoritmo 2β-approssimato per il TSP β-metrico rafforzato [12] la cui struttura è molto simile all'algoritmo precedentemente visto, ovvero il Double MST, da cui differisce solo per alcune scelte effettuate al punto (2).

La struttura del Refined Double MST è la seguente:

Input: un grafo indiretto completo $G=(V,E)$ con funzione di costo $c: E \to \mathbb{R}^+$ tale che soddisfi la β-disuguaglianza triangolare rafforzata.

1) Costruire un minimo albero ricoprente (MST) T di G ;
2) Eseguire un'*accurata* visita in profondità su T partendo da un vertice arbitrario $v_.$;
3) Sia $v_0, v_1, \dots, v_{n-1}$ la sequenza restituita dalla visita in profondità, allora restituire in output la sequenza $v_0, v_1, \dots, v_{n-1}, v_0$.

Dove parliamo di *visita in profondità accurata* intendiamo una visita dove, ad ogni nodo visitato, il prossimo nodo da visitare non è un qualsiasi figlio del nodo corrente ma un nodo che appartiene all'insieme dei *closest children* del nodo in T .
Chiamiamo closest children di v in T quei nodi v' , appartenenti al sottoalbero T' indotto in T da v e dai suoi figli, tali che $c(\{v, v'\})$ non sia maggiore del costo tra v ed ogni altro nodo in T' .

3.3 Christofides (CH)

L'algoritmo di Christofides è una delle euristiche più conosciute per il TSP metrico dato che, pur non essendo la più veloce tra quelle esistenti, è quella che offre il miglior rapporto di approssimazione, ovvero $\frac{3}{2}$.

Anche per l'algoritmo di Christofides, come precedentemente anche per il Double MST, possiamo dire che nel caso di TSP fortemente β-metrico questo diventa un algoritmo $(1+\frac{2\beta-1}{3\beta^2-2\beta+1})$ -approssimato [2].

La struttura di Christofides è la seguente:

Input: un grafo indiretto completo $G=(V,E)$ con funzione di costo $c: E \to \mathbb{R}^+$ tale che soddisfi la β-disuguaglianza triangolare.

1) Costruire un minimo albero ricoprente (MST) T di G ;
2) Trovare un matching perfetto di costo minimo M sul sottografo completo indotto su G dai nodi aventi grado dispari in T ;
3) Costruire un circuito euleriano su $T \cup M$;
4) Costruire un ciclo hamiltoniano eliminando le ripetizioni dei nodi dal circuito euleriano;
5) Restituire in output il ciclo hamiltoniano costruito al punto (4).

Quando parliamo di matching di un grafo G intendiamo un insieme di archi del grafo senza vertici in comune; diciamo inoltre che un matching è perfetto se copre tutti i vertici di G .
Un matching perfetto di peso minimo di G è un matching perfetto il cui costo non sia maggiore del costo di ogni altro matching perfetto di G .

Un circuito euleriano su un grafo indiretto G è un cammino sul grafo che attraversa esattamente una volta tutti gli archi di G , partendo e terminando sullo stesso vertice.

3.4 Cycle Cover (CC)

L'algoritmo Cycle Cover è un algoritmo, come il Refined Double MST ed a differenza di Double MST e Christofides, espressamente ideato per il TSP fortemente β-metrico ed ha fattore di approssimazione pari a $(\frac{2}{3}+\frac{1}{3}\cdot\frac{\beta}{1-\beta})$ [2]. Questo fattore di approssimazione risulta migliore di quello degli altri algoritmi visti sinora nel caso in cui $\frac{1}{2}\leq\beta<\frac{2}{3}$.

La struttura del Cycle Cover è la seguente:

Input: un grafo indiretto completo $G=(V,E)$ con funzione di costo $c: E\rightarrow\mathbb{R}^+$ tale che soddisfi la β-disuguaglianza triangolare rafforzata.

1) Costruire un cycle cover di costo minimo C di G i cui cicli abbiano lunghezza non minore di 3;
2) Per $1\leq i\leq k$ trovare l'arco meno costoso $\{a_i,b_i\}$ in ogni ciclo $C_i\in C$;
3) Costruire un ciclo hamiltoniano da C sostituendo gli archi $\{\{a_i,b_i\}t.c.1\leq i\leq k\}$ con gli archi $\{\{b_i,a_{i+1}\}t.c.1\leq i\leq k-1\}\cup\{\{b_k,a_1\}\}$;
4) Restituire in output il ciclo hamiltoniano di cui al punto (3).

Quando parliamo di cycle cover C di un grafo indiretto G intendiamo un insieme di cicli che coprano tutti i vertici di G tali che ogni vertice di G appartenga esattamente ad uno dei cicli.
Un cycle cover di costo minimo di G è un cycle cover il cui costo non sia maggiore del costo di ogni altro cycle cover di G .

L'algoritmo per la costruzione del cycle cover di costo minimo ha la seguente struttura [4]:

Input: un grafo indiretto completo $G=(V,E)$ con funzione di costo $c: E\rightarrow\mathbb{R}^+$

1) Costruire un grafo ausiliario G' a partire da G tale che:
- Per ogni nodo $v\in V(G)$ vengono creati due nodi $v',v''\in V(G')$;
- Per ogni arco $e\in E(G)$ vengono creati due nodi $e',e''\in V(G')$ collegati da un arco di peso pari a 0;
- Per ogni arco $e=\{v,u\}\in E(G)$ vengono creati quattro archi $\{v',e'\},\{v'',e'\},\{u',e''\},\{u'',e''\}$ in $E(G')$ con peso pari all'arco $e\in E(G)$.

2) Calcolare un matching perfetto di peso minimo sul grafo G' ;
3) Se due archi di G' selezionati dal matching provengono entrambi dallo stesso arco $e\in E(G)$, allora aggiungere e agli archi tra cui evidenziare i cicli;

4) Cercare tra gli archi evidenziati al punto (3) tutti i possibili cicli disgiunti di lunghezza maggiore o uguale a 3;
5) Restituire in output questo insieme di cicli.

4. Implementazione

Precedentemente a questo lavoro di tesi già Annalisa D'Andrea [3] ed Antonio Di Francesco [4] nelle loro tesi di laurea avevano analizzato gli algoritmi di cui al capitolo 3 e ne avevano realizzato delle implementazioni in Java; purtroppo diverse limitazioni relative alle prestazioni hanno reso necessaria una nuova implementazione più efficiente dal punto di vista temporale e dell'occupazione di memoria.

Alla base delle prestazioni non ottimali del codice in questione vi era l'utilizzo di una libreria relativa ai grafi pensata non tanto per essere efficiente ma per permettere la visualizzazione grafica di questi; tale libreria utilizzava una rappresentazione mediante matrici di adiacenza che produceva un notevole spreco di memoria avendo noi a che fare con un input rappresentato da grafi indiretti.

Oltre ad avere a disposizione una libreria che minimizzasse l'occupazione di memoria era nell'interesse del lavoro di implementazione che la libreria semplificasse quanto più possibile lavorare con tutte le strutture dati e gli algoritmi di base, quali ad esempio le visite di alberi.

Avendo queste necessità e lavorando in C++ la scelta è ricaduta sulla “Libreria di Algoritmi e Strutture Dati” (ASD) [5].
La versione da cui si è partiti è la 2.1, la licenza sotto cui questa libreria viene distribuita è la “GNU Lesser General Public License”.

Alla libreria in questione sono state inizialmente eliminate tutte le funzionalità non necessarie al lavoro di tesi, funzionalità comunque reintegrabili nuovamente in futuro se ulteriori sviluppi dovessero renderle necessarie, e sono state apportate delle modifiche relative sia alle interfacce che alle implementazioni di classi ed algoritmi.
Sono inoltre state aggiunte alla libreria tutte le strutture dati, quali i grafi β-metrici e le istanze di TSP, necessarie allo svolgimento del lavoro, oltre agli algoritmi di cui al capitolo 3, all'algoritmo per il calcolo del matching perfetto di peso minimo, del cycle cover e del circuito euleriano.

Per quanto riguarda la necessità di dover risolvere problemi di matching perfetto di peso minimo si è scelto inizialmente di utilizzare un approccio orientato alla programmazione lineare intera (PLI).
A tal fine si è reso necessario avere a disposizione un solutore di PLI e la scelta è ricaduta su GLPK [6], essenzialmente a causa di un'esperienza pregressa di utilizzo e perché tra i vari solutori analizzati era l'unico disponibile con una licenza open source e senza dover pagare per l'utilizzo; la versione di GLPK utilizzata è stata la 4.25.

Dopo alcuni esperimenti si è però deciso di optare per un'altra soluzione in quanto il numero di variabili del problema risultava troppo grande per essere risolto con un'occupazione di risorse accettabile.

Si è quindi sostituita questa implementazione del matching perfetto di peso minimo con l'implementazione scritta in C++ da Vladimir Kolmogorov [9] e basata sull'algoritmo di Edmonds [11].
Il codice del Blossom V [10] è stato scaricato da Internet ed integrato all'interno della libreria.

Alla libreria ASD è stata inoltre aggiunta la possibilità di caricare e salvare istanze del TSP mediante il formato TSPLIB95 [7].
Questa modifica si è resa necessaria per poter salvare le istanze da testare in un formato riconosciuto dal solutore esatto [8] e rendere inoltre le istanze non volatili e quindi testabili nuovamente in seguito a modifiche degli algoritmi.

Ai fini dell'analisi sperimentale sono stati realizzati cinque eseguibili esemplificativi:

- generatore_grafi: permette di creare un grafo indiretto completo con pesi degli archi casuali forniti in input un valore per β ed il numero di nodi desiderati per il grafo; permette inoltre di salvare l'istanza generata di TSP β-metrico nel formato TSPLIB95.

- doubleMST: solutore di TSP metrico che implementa l'algoritmo α-approssimato Double MST.

- refinedDoubleMST: solutore di TSP metrico che implementa l'algoritmo α-approssimato Refined Double MST.

- christofides: solutore di TSP metrico che implementa l'algoritmo α-approssimato Christofides.

- cycleCover: solutore di TSP metrico che implementa l'algoritmo α-approssimato Cycle Cover.

Il codice è stato compilato e testato su Ubuntu Linux 8.04, il compilatore utilizzato è g++ 4.2.4.

5. Risultati sperimentali

In questo capitolo proporremo il fattore di approssimazione medio relativo all'esecuzione degli algoritmi da noi implementati su istanze del TSP fortemente β-metriche generate in modo casuale.
Il fattore di approssimazione relativo ad ognuno dei singoli esperimenti è reperibile nell'appendice 1.

Per quanto riguarda la generazione casuale delle istanze sono state generate, per ogni valore di β nell'insieme $\{0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95\}$, dieci istanze di 1000 nodi con peso degli archi tale che $100000 \leq w(e) \leq \beta * 200000$.
La scelta di 100000 come valore di base è del tutto arbitraria e funzionale solo all'avere un buon numero di valori nell'intervallo.

Per poter ottenere la soluzione ottima delle istanze generate, in modo da poter calcolare il fattore di approssimazione α, è stato usato il solutore esatto Concorde [8].

β	2MST	R2MST	CH	CC
0.55	1.02030	1.02024	1.01196	1.00011
0.60	1.04065	1.04000	1.02389	1.00038
0.65	1.06136	1.06073	1.03733	1.00060
0.70	1.08132	1.08109	1.04874	1.00073
0.75	1.10122	1.10038	1.06141	1.00085
0.80	1.12078	1.12104	1.07329	1.00120
0.85	1.14204	1.14190	1.08477	1.00138
0.90	1.16336	1.16485	1.09938	1.00116
0.95	1.18340	1.18453	1.11418	1.00129

Al fine della verifica degli algoritmi sono stati inoltre eseguiti degli esperimenti su alcune istanze della libreria TSPLIB95 [7] e su istanze metriche casuali generate da spazi L1, L2 ed LInf i cui relativi fattori di approssimazione sono riportati nelle appendici 3 e 4.

Tutti gli esperimenti sono stati condotti su un calcolatore Intel Celeron M 1500MHz con 1521944 KB di RAM e sistema operativo Ubuntu Linux 8.04.

6. Conclusioni

Dai risultati sperimentali possiamo vedere che, sia in media, sia nei singoli esperimenti, il fattore di approssimazione di tutti e quattro gli algoritmi si è tenuto di molto al di sotto del limite superiore di approssimazione, limite che è possibile vedere calcolato per i valori di β interessati nell'appendice 2.

Alla luce degli esperimenti possiamo porre in un ordinamento relativo i valori di approssimazione medi degli algoritmi:

- CC < CH < {2MST, R2MST}

Da questo ordinamento appare evidente che l'algoritmo che si è comportato meglio è il Cycle Cover, a sua volta seguito da Christofides.
Per quanto riguarda gli algoritmi Double MST e Refined Double MST possiamo certamente affermare che il loro rapporto di approssimazione medio risulta sempre superiore a quello dei due algoritmi migliori, ma non possiamo metterli in relazione tra loro in quanto troppo vicini nei rapporti sia in media che nei singoli esperimenti.

Analizzando i dati possiamo notare che l'andamento del fattore α risulta crescente per quanto riguarda gli algoritmi Christofides, Double MST e Refined Double MST con un incremento medio pari a:

1) CH = 0.01277
2) 2MST = 0.02038
3) R2MST = 0.02053

Per quanto riguarda l'algoritmo Cycle Cover l'andamento del fattore α è risultato crescente per $0.55 \leq \beta \leq 0.85$, con incremento medio pari a 0.00021, mentre si riscontra una discontinuità nella crescita per $0.90 \leq \beta \leq 0.95$.

Riportiamo nella tabella seguente lo scarto tra il fattore medio di approssimazione ed il limite superiore al variare di β:

β	2MST	R2MST	CH	CC
0.55	-0.17771	-0.07976	-0.11187	-0.07396
0.60	-0.34396	-0.16000	-0.20338	-0.16628
0.65	-0.48909	-0.23927	-0.27274	-0.28511
0.70	-0.60833	-0.31891	-0.32509	-0.44371
0.75	-0.69878	-0.39962	-0.35964	-0.66581
0.80	-0.76157	-0.47896	-0.38125	-0.99880
0.85	-0.79755	-0.55810	-0.39223	-1.55417
0.90	-0.81224	-0.63515	-0.39141	-2.66550
0.95	-0.81107	-0.71547	-0.38374	-5.99871

Notiamo che l'andamento dello scarto nei nostri esperimenti è crescente per tutti gli algoritmi tranne che per Christofides, per il quale l'andamento è crescente fino ad un picco posto in corrispondenza del valore 0.85 di β, valore dopo il quale l'andamento diviene decrescente.

L'occupazione di memoria delle implementazioni realizzate è risultata accettabile, non sono stati necessari infatti particolari accorgimenti per gestire istanze dell'ordine di grandezza di 1000 nodi.
I tempi di esecuzione non sono stati studiati con rigore, stime approssimative comunque sono le seguenti:

- 2MST ≃ 25s
- R2MST ≃ 25s
- CH ≃ 30s
- CC ≃ 1:20h

L'algoritmo di Cycle Cover ha comunque margini di miglioramento per quanto riguarda il tempo medio di esecuzione, margine che potrebbe essere sfruttato utilizzando un algoritmo migliore nella fase di individuazione dei cicli sul grafo originario.

7. Bibliografia

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Introduzione agli Algoritmi*, Jackson Libri (1994)

[2] Hans-Joachim Böckenhauer, Juraj Hromkovič, Ralf Klasing, Sebastian Seibert, Walter Unger, *Approximation algorithms for the TSP with sharpened triangle inequality*, Information Processing Letters 75 (2000), 133-138

[3] Annalisa D'Andrea, *Approssimazione del problema del commesso viaggiatore in grafi supermetrici in tempo lineare*, Tesi di laurea.

[4] Antonio Di Francesco, *Un algoritmo di approssimazione per il problema del commesso viaggiatore sui grafi fortemente metrici*, Tesi di laurea.

[5] Libreria di Algoritmi e Strutture Dati, http://gauguin.info.uniroma2.it/~italiano/Teaching/Algoritmi/asd/doc/index.html

[6] GNU Linear Programming Kit, http://www.gnu.org/software/glpk/

[7] TSPLIB95, http://www.informatik.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

[8] Concorde TSP Solver, http://www.tsp.gatech.edu/concorde.html

[9] Vladimir Kolmogorov, *Blossom V: a new implementation of a minimum cost perfect matching algorithm*, UCL Technical Report (2008)

[10] Blossom V, http://www.adastral.ucl.ac.uk/~vladkolm/software.html

[11] Jack Edmonds, *Path, trees and flowers*, Canadian Journal of Mathematics (1965)

[12] Davide Bilò, Luca Forlizzi, Guido Proietti, *Approximating the metric TSP in linear time*, 34th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG'08), 30 June-2 July, 2008, Durham University, UK.
Vol. 5344 of Lecture Notes in Computer Science, Springer-Verlag, 43-54.

8. Appendice 1

β 0.55

2MST	R2MST	CH	CC
1.01968	1.02079	1.01249	1.00016
1.02003	1.02024	1.01083	1.00004
1.02075	1.02057	1.01173	1.00022
1.02154	1.02052	1.01234	1.00030
1.02008	1.02050	1.01130	1.00011
1.02056	1.01924	1.01218	1.00009
1.01955	1.02023	1.01231	1.00010
1.02055	1.01938	1.01231	1.00007
1.02045	1.02118	1.01219	1.00005
1.01984	1.01981	1.01196	1

Fattore di approssimazione delle istanze con β = 0.55

β 0.60

2MST	R2MST	CH	CC
1.04187	1.04185	1.02591	1.00047
1.04118	1.03904	1.02377	1.00003
1.04364	1.04341	1.02451	1.00074
1.04290	1.04193	1.02260	1.00021
1.03875	1.03809	1.02543	1.00045
1.03800	1.03643	1.02199	1.00028
1.03847	1.03922	1.02299	1.00051
1.04187	1.04010	1.02455	1.00026
1.04133	1.03988	1.02418	1.00069
1.03853	1.04009	1.02302	1.00019

Fattore di approssimazione delle istanze con β = 0.60

β 0.65

2MST	R2MST	CH	CC
1.05698	1.05553	1.03686	1.00085
1.06257	1.05904	1.03511	1.00038
1.06290	1.06182	1.03867	1.00049
1.06113	1.06114	1.03862	1.00062
1.06082	1.06147	1.03663	1.00055
1.06413	1.06389	1.03879	1.00036
1.06203	1.06104	1.03721	1.00140
1.05721	1.05825	1.03691	1.00052
1.06447	1.06576	1.04014	1.00045
1.06143	1.05938	1.03443	1.00038

Fattore di approssimazione delle istanze con β = 0.65

β 0.70

2MST	R2MST	CH	CC
1.07916	1.07900	1.04631	1.00107
1.08491	1.08412	1.05140	1.00114
1.07405	1.07392	1.04854	1.00068
1.07929	1.07714	1.04651	1.00072
1.08102	1.08277	1.04715	1.00097
1.08831	1.08779	1.05222	1.00063
1.08184	1.08146	1.05149	1
1.08080	1.08008	1.04543	1.00061
1.08237	1.08025	1.04985	1.00074
1.08153	1.08445	1.04857	1.00083

Fattore di approssimazione delle istanze con β = 0.70

β 0.75

2MST	R2MST	CH	CC
1.10050	1.10324	1.06423	1.00078
1.10365	1.09804	1.05682	1.00089
1.10438	1.09948	1.05775	1.00038
1.10324	1.09796	1.06511	1.00200
1.10228	1.10844	1.06140	1.00075
1.10231	1.10273	1.06299	1.00023
1.10153	1.09732	1.06586	1.00060
1.10436	1.09897	1.05518	1.00106
1.09651	1.09828	1.06400	1.00140
1.09486	1.09935	1.06077	1.00044

Fattore di approssimazione delle istanze con β = 0.75

β 0.80

2MST	R2MST	CH	CC
1.12009	1.12218	1.06982	1.00123
1.12650	1.11494	1.07285	1.00128
1.12233	1.12202	1.07915	1.00129
1.11520	1.12228	1.07662	1.00104
1.11581	1.12072	1.07010	1.00122
1.11774	1.11582	1.07329	1.00103
1.12633	1.12698	1.07453	1.00063
1.12627	1.12663	1.06870	1.00065
1.12212	1.12190	1.07574	1.00182
1.11548	1.11695	1.07210	1.00187

Fattore di approssimazione delle istanze con β = 0.80

β 0.85

2MST	R2MST	CH	CC
1.12763	1.13438	1.08365	1
1.14284	1.13845	1.09252	1.00052
1.13938	1.14327	1.07671	1.00064
1.15081	1.14508	1.08227	1.00211
1.12857	1.12715	1.07930	1.00155
1.14853	1.14442	1.08489	1.00105
1.14906	1.14717	1.09385	1.00306
1.14594	1.15249	1.08526	1.00210
1.14035	1.13962	1.08343	1.00089
1.14734	1.14704	1.08591	1.00193

Fattore di approssimazione delle istanze con β = 0.85

β 0.90

2MST	R2MST	CH	CC
1.16218	1.16654	1.09223	1.00196
1.16368	1.16916	1.09444	1.00053
1.15841	1.16067	1.09749	1.00076
1.16622	1.16601	1.10126	1.00165
1.16021	1.16829	1.10761	1.00207
1.16141	1.16110	1.10143	1.00175
1.15570	1.14455	1.09961	1.00058
1.17271	1.17293	1.10144	1.00234
1.17075	1.17049	1.10155	1
1.16239	1.16885	1.09681	1

Fattore di approssimazione delle istanze con β = 0.90

β 0.95

2MST	R2MST	CH	CC
1.18859	1.18405	1.11942	1.00098
1.18330	1.18535	1.10916	1.00141
1.17477	1.18442	1.11766	1.00186
1.19100	1.18976	1.12628	1.00097
1.17673	1.17469	1.11071	1.00146
1.17372	1.17910	1.10607	1.00040
1.18637	1.18420	1.11941	1
1.18414	1.18872	1.10803	1.00192
1.18615	1.18812	1.11092	1.00136
1.18930	1.18694	1.11418	1.00257

Fattore di approssimazione delle istanze con β = 0.95

9. Appendice 2

β	2MST	R2MST	CH	CC
0.55	1.19801	1.1	1.12383	1.07407
0.60	1.38461	1.2	1.22727	1.16666
0.65	1.55045	1.3	1.31007	1.28571
0.70	1.68965	1.4	1.37383	1.44444
0.75	1.8	1.5	1.42105	1.66666
0.80	1.88235	1.6	1.45454	2
0.85	1.93959	1.7	1.47700	2.55555
0.90	1.97560	1.8	1.49079	3.66666
0.95	1.99447	1.9	1.49792	7

Upper bound degli algoritmi calcolato sui valori di β interessati

10. Appendice 3

TSPLIB95	2MST	R2MST	CH	CC
bayg29	1.37639	1.37267	1.08509	1.20993
bays29	1.30297	1.19950	1.09554	1.30841
brazil58	1.20618	1.15306	1.06154	1.88682
fri26	1.20917	1.14941	1.05336	1.19530
gr120	1.44742	1.28954	1.12863	1.93849
gr17	1.15875	1.11558	1.06666	1.42973
gr24	1.40723	1.23742	1.13915	1.36949
gr48	1.37177	1.36424	1.12009	1.28695
hk48	1.24674	1.28348	1.12416	1.51827
pa561	1.38291	1.39739	1.14585	1.36988
si1032	1.07123	1.05538	1.01814	1.04070
si175	1.08202	1.05722	1.02821	1.09113
si535	1.07973	1.06066	1.03785	1.19952
swiss42	1.35428	1.28358	1.08562	1.51924

Fattore di approssimazione di alcune istanze di TSPLIB95

TSPLIB95	2MST	R2MST	CH	CC
bayg29	0:00.04	0:00.01	0:00.06	0:00.05
bays29	0:00.00	0:00.00	0:00.00	0:00.01
brazil58	0:00.00	0:00.00	0:00.03	0:00.07
fri26	0:00.00	0:00.00	0:00.00	0:00.01
gr120	0:00.04	0:00.04	0:00.04	0:00.86
gr17	0:00.00	0:00.00	0:00.00	0:00.00
gr24	0:00.00	0:00.00	0:00.00	0:00.00
gr48	0:00.00	0:00.00	0:00.00	0:00.03
hk48	0:00.01	0:00.00	0:00.00	0:00.03
pa561	0:05.67	0:05.47	0:05.85	9:19.78
si1032	0:32.30	0:32.21	0:32.70	1:30:46
si175	0:00.15	0:00.15	0:00.16	0:04.04
si535	0:04.87	0:04.37	0:05.28	6:07.87
swiss42	0:00.00	0:00.00	0:00.00	0:00.02

Tempi di esecuzione su alcune istanze di TSPLIB95

11. Appendice 4

L1 Space

istanza	2MST	R2MST	CH	CC
0	1.44552	1.40774	1.13910	3.71646
1	1.41866	1.39143	1.14158	3.42857
2	1.41581	1.37238	1.13051	3.95415
3	1.42493	1.36586	1.13427	3.84556
4	1.43516	1.41569	1.13899	3.56490
5	1.40746	1.38766	1.13564	3.43474
6	1.42332	1.35268	1.13890	4.09736
7	1.42683	1.38306	1.13426	3.68388
8	1.42464	1.39594	1.15500	4.21936
9	1.43592	1.37766	1.14439	4.33861
media	**1.42582**	**1.38501**	**1.13926**	**3.82835**

Fattore di approssimazione di alcune istanze generate da spazi L1

L2 Space

istanza	2MST	R2MST	CH	CC
0	1.42734	1.38315	1.12505	3.52313
1	1.42317	1.38142	1.13771	3.47525
2	1.41874	1.38705	1.14055	3.74193
3	1.37355	1.37312	1.13318	4.00902
4	1.39208	1.36643	1.11957	3.87926
5	1.43569	1.38647	1.12716	3.65201
6	1.39647	1.35952	1.11366	3.72187
7	1.39153	1.36232	1.12152	3.49841
8	1.40138	1.36838	1.11973	3.23369
9	1.40391	1.37759	1.12246	3.82659
media	**1.40638**	**1.37481**	**1.12605**	**3.65611**

Fattore di approssimazione di alcune istanze generate da spazi L2

LInf Space

istanza	2MST	R2MST	CH	CC
0	1.41377	1.38057	1.13459	3.41124
1	1.41857	1.39661	1.13866	3.72835
2	1.43202	1.38400	1.12837	3.52494
3	1.42602	1.38930	1.12921	3.50383
4	1.41943	1.35343	1.13128	4.07238
5	1.45133	1.36620	1.14398	3.73247
6	1.40784	1.38834	1.13907	3.59415
7	1.42360	1.35584	1.12708	4.02534
8	1.42916	1.40182	1.12569	3.43286
9	1.44365	1.40033	1.14378	3.75085
media	**1.42653**	**1.38164**	**1.13417**	**3.67764**

Fattore di approssimazione di alcune istanze generate da spazi LInf