SPERIMENTAZIONE NUMERICA

Il minimal spanning tree per grafi euclidei

Mario Correddu

INTRODUZIONE

Dato $G = (V, E, W)$ un grafo pesato il problema del minimal spanning tree può essere enunciato come trovare un sottografo aciclico (quindi un albero) $T = (V, E', W)$ i cui archi siano incidenti con tutti i vertici del grafo G e che minimizzi

$$\sum_{e \in E'} W(e)$$

In questo lavoro verranno considerati grafi euclidei, ovvero grafi completi i cui nodi sono punti in $\mathbb{R}^m$ e i pesi degli archi sono dati dalla distanza euclidea fra i nodi che collegano.

Il report è diviso in due parti:

- nella prima saranno esposti e confrontati un metodo per calcolare il MST e poi tre euristiche basate sul paradigma del divide et impera, due derivanti dall'algoritmo presentato in [2] e una non trovata in letteratura, che permettano di calcolare degli spanning tree in un tempo minore. L'utilizzo delle euristiche infatti può essere utile nelle applicazioni se si ottiene una buona approssimazione del MST, per ridurre i tempi di esecuzione. Infatti nonostante in dimensione 2 siano noti algoritmi esatti che risolvono il problema in tempo $O(nlog(n))$, ad esempio utilizzando la triangolazione di Delaunay [4], in dimensione maggiore non sono noti algoritmi esatti altrettanto rapidi.

- nella seconda saranno messe a confronto le proprietà del MST rispetto alla variante bipartita del problema, ovvero dove i nodi sono divisi in due sottoinsiemi e gli unici archi presenti nel grafo collegano i punti da un sottoinseme all'altro. Alcuni studi sul MST nel caso di grafi euclidei sono stati svolti nella tesi di Riva [3] per grafi in dimensione 1 e 2.

I test sono stati eseguiti in macchina matlab versione 2020b, considerando punti in $[0,1]^m$ scelti con distribuzione uniforme.

EMST e euristiche

Algoritmi esatti

L'algoritmo MSTJ, utilizzato per il calcolo del MST, è un'applicazione dell'algoritmo di Jarnik-Prim per grafi euclidei. Esso è un algoritmo Greedy che a ogni passo aggiunge un nodo, non ancora inserito, con distanza minima dall'albero costruito fino a quel momento. L'algoritmo si basa su una gestione tramite una coda di priorità dei nodi in modo da essere asintoticamente migliore di una ricerca del minimo ad ogni passo.

In questo modo usando un heap binario per rappresentare la coda di priorità, si ottiene un costo asintotico di $O(|E|log(n)$ passi, dove $n = |V|$. Poichè il grafo è euclideo, ogni nodo è connesso a tutti gli altri e quindi il grafo ha n^2 archi e il costo asintotico dell'algoritmo al caso pessimo è di $O(n^2 log(n))$ passi.

Per l'implementazione è stata sfruttata la struttura di grafo euclideo calcolando solo i pesi degli archi incidenti al nodo che viene aggiunto a ogni iterazione, in maniera tale da non calcolare già dalla prima iterazione tutti gli archi presenti nel grafo, rendendo così il costo teorico in termini di spazio $O(n)$ invece che $O(n^2)$.

Verranno confrontate anche le performance dell'algoritmo per il minimum spanning tree già presente in Matlab, dato dalla funzione minspantree. Tale funzione dato un grafo restituisce un corrispettivo minimal spanning tree. Tuttavia per la costruzione del grafo è necessario calcolare la matrice di adiacenza, rendendo così lo spazio utilizzato $O(n^2)$.

Euristiche

Di seguito un presentazione di tre euristiche basate sul paradigma divide et impera:

MSTeusort: dato X, il vettore di lunghezza n che contiene punti in $\mathbb{R}^m$, e cut$\in \mathbb{N}$, MSTeusort ordina X lungo la coordinata cut (mod m) e lo divide a metà in due sottoinsiemi di $\frac{n}{2}$ nodi ciascuno. A questo punto calcola uno spanning tree all'interno di ciascuno due sottografi generati dai due sottinesiemi e infine aggiunge un arco che colleghi i due alberi, scelto fra gli archi che collegano i $\frac{\sqrt{n}}{m}$ nodi più vicini all'iperpiano affine che separa i due sottografi.

fissato $cut \in \mathbb{N}$
procedure MSTsort(X,i)
T = ∅;
cut=cut mod m;
sia n il numero di punti in X
ordina X secondo la cut-esima componente dei vettori che lo compongono
X1=X(1:$\lfloor \frac{n}{2} \rfloor$);
X2=X($\lfloor \frac{n}{2} \rfloor$+1:n);
trova l'arco di lunghezza minima che collega gli ultimi $\frac{\sqrt{n}}{2}m$ nodi di X1 e i primi $\frac{\sqrt{n}}{2}m$ nodi di X2 e lo aggiunge a T
T1=MSTsort(X1,i+1);
T2=MSTsort(X2,i+1);
$T = T \cup T1 \cup T2$;
end
output T

L'algoritmo ha un tempo di esecuzione $O(n(logn)^2)$.

Le euristiche seguenti sono delle versioni opportunamente modificate dall'algoritmo presentato in [2], che applica il paradigma del divide et impera per costruire uno spanning tree di altezza fissata.

MSTeuh costruisce uno spanning tree a partire da un nodo s : dopo aver diviso la regione in sottoregioni costituite da cubi $m \times m$ di lato $\frac{L}{\left\lfloor n^{1-\frac{1}{m}} \right\rfloor^{\frac{1}{m}}}$, dove L è il lato di un m-cubo che contiene tutti i nodi, per ogni cubo seleziona un nodo, aggiunge l'arco che lo collega ad s all'albero e costruisce uno spanning tree all'interno del cubo con radice tale nodo.

scelto s a caso fra i punti di X
procedure MSTeuh(X, s)
T = ∅;
$k = \left\lfloor |G|^{1-\frac{1}{m}} \right\rfloor$;
Sia L il lato di un m-cubo che contiene tutti i nodi in X,
Dividi il m-cubo in m-cubi di lato $\frac{L}{\left\lfloor k^{\frac{1}{m}} \right\rfloor}$;
Sia k' il numero di m-cubi e siano X_i i punti di X nell'i-esimo cubo, $1 \leq i \leq k'$;

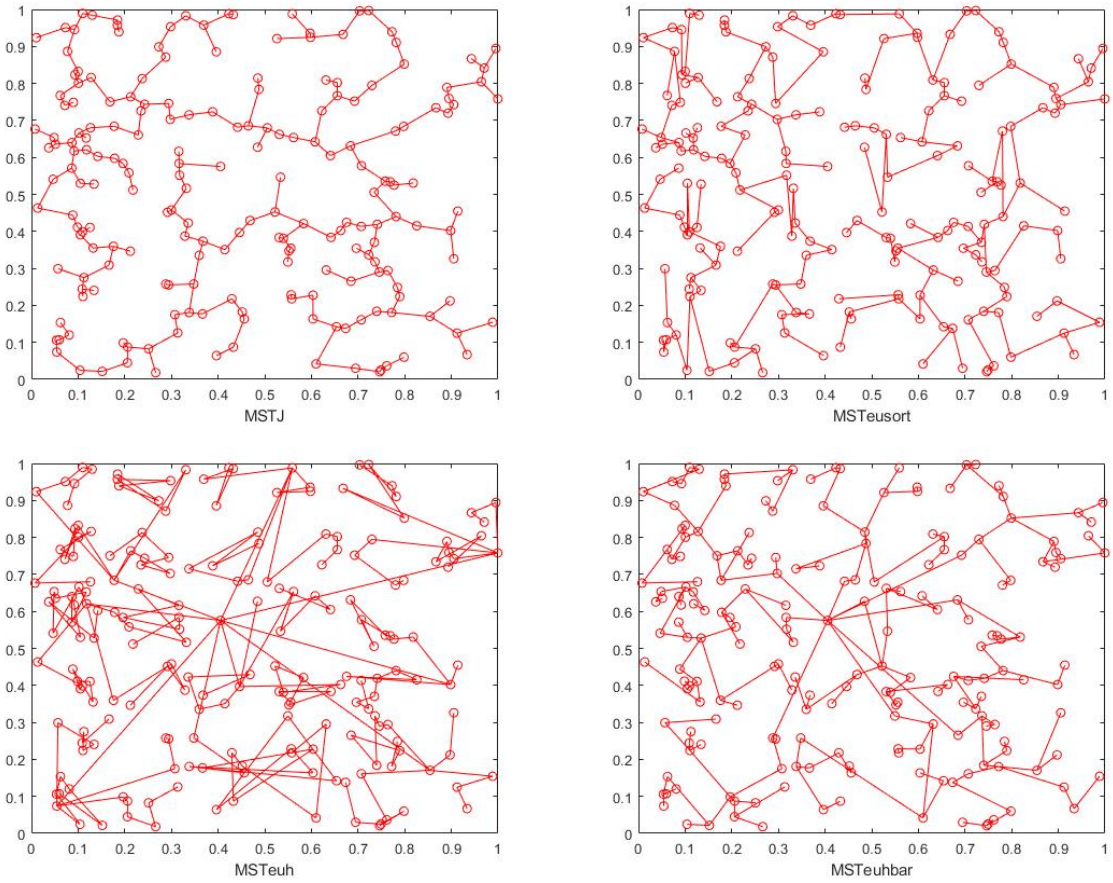
for i from 1 to k'
if $|X_i| \geq 1$ then
scegli un punto s in X_i;
$T = T \cup (s_i, s)$;
if $|X_i| > 1$ then
$T = T \cup MSTeuh(X_i, s_i)$;
end;
end;
output T

MSTeuhbar si basa sulla stessa suddivisione in sottoregioni di MSTeuh, ma pone come nodo radice di ogni sottoalbero il nodo che minimizza la distanza dal baricentro dei nodi della sottoregione. Ogni sottoalbero viene poi collegato al nodo s scegliendo il punto che minimizza la distanza fra gli stessi.

scelto s fra i nodi di G:
procedure MSTeuhbar(G,s)
T = $\emptyset$;
$k = \left\lfloor |G|^{1-\frac{1}{m}} \right\rfloor$
Sia L il lato di un cubo che contiene tutti i nodi in S,
Dividi il m-cubo in m-cubi di lato $\frac{L}{\left\lfloor k^{\frac{1}{m}} \right\rfloor}$;
Sia k' il numero di m-cubi e siano S_i i punti di S nell'i-esimo cubo, $1 \leq i \leq k'$;
for i from 1 to k'
if $|S_i| \geq 1$ then
sia v_i il punto di S_i che minimizza la distanza da s;
$T = T \cup (v_i, s)$;
if $S_i > 1$ then
calcola il baricentro dei punti in S_i;
sia s_i il punto di S_i che minimizza la la distanza dal baricentro;
$T = T \cup MSTeuh(S_i, s_i)$;
end
end
output T

Sia MSTeuh che MSTeuhbar hanno un costo teorico di $O(nloglog(n))$ passi al caso ottimo e $O(n^2)$ al caso pessimo.

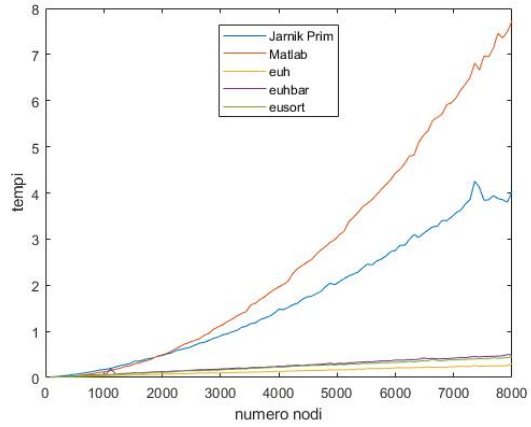
Figure 1: Esempi di alberi prodotti dagli algoritmi, dimensione 2, n=200



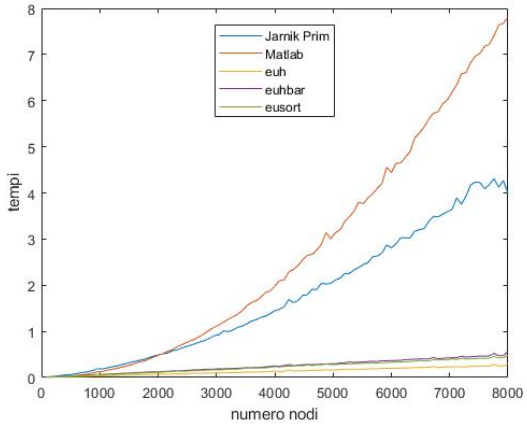
Test

Sono stati condotti 2 tipi di test, uno che confronta i tempi di esecuzione e uno che confronta le proprietà dei grafi ottenuti con i vari algoritmi, ovvero peso, frequenza dei nodi di grado da 1 a 5 e dei nodi di grado maggiore di 5. I test sono stati eseguiti nel caso in cui i nodi siano in $\mathbb{R}^m$ con $m = 2,3,7$ e 13 e con numero di nodi fino a 8000 e fino a 200000 per le euristiche in dimensione 2,3 e 7.

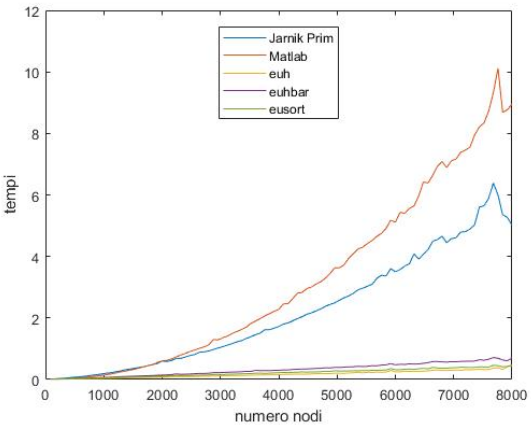
Figure 2: Tempi di esecuzione



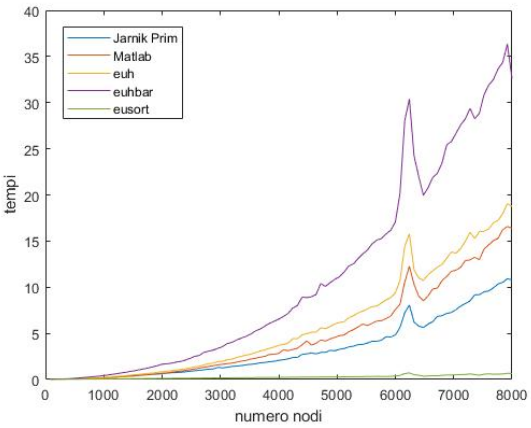
Dim 2



Dim 3



Dim 7



Dim 13

Confrontando i risultati ottenuti per i tempi fino a 8000 nodi , si notano delle differenze di comportamento, al crescere di n, fra i due algoritmi esatti. Infatti sotto una soglia dipendente da m, l'implementazione dell'algoritmo di Jarnik-Prim fornita da Matlab risulta la più rapida,

mentre, superata quella soglia, è MSTJ ad essere più performante, con una differenza nei tempi che aumenta al crescere di n. Tuttavia non conoscendo il codice sorgente, si può solo congetturare che la causa sia una diversa gestione della coda di priorità di minspantree rispetto a MSTJ. In aggiunta il costo in termini di spazio di MSTJ uguale a $O(n)$, lo rende preferibile a euclmatlabMST che invece ha un costo di $O(n^2)$.

Osservando invece le euristiche si nota invece una performance dipendente da m per gli algoritmi euh e euhbar. In particolare sembrerebbe che mentre per m=1,2,7 mantengano un costo temporale associabile a $O(nlog(logn))$, per m=13 il tempo di esecuzione degli algoritmi aumenti in modo considerevole rendendoli peggiori degli algoritmi esatti. Questo fenomeno parrebbe dipendere dal modo in cui viene suddivisa la regione, in cubi di lato $\frac{L}{k^{\frac{1}{m}}}$ dove $k = \lfloor |G|^{1-1/m} \rfloor$. Infatti ponendo $n = 8000$ e $m = 13$, si ha $k^{\frac{1}{m}} = 1$ difatto rendendo la suddivisione in cubi inesistente e causando n ricorsioni. In questo modo gli algoritmi si rivelano meno performanti nel caso di n non sufficientemente elevati rispetto a m, rendendoli nella pratica $O(n^2)$ in termini di spazio e tempo.

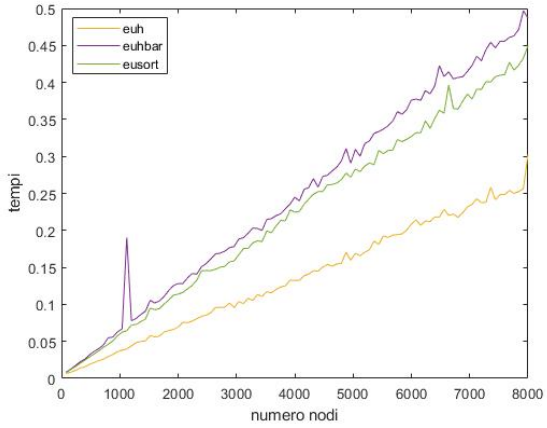
Alla divisione in cubi potrebbe essere attribuito anche l'aumento considerevole dei tempi di esecuzione degli algoritmi MSTeuh e MSTeuhbar in dimensione 7 dove, per $n = 84000$, i tempi risultano essere più del doppio rispetto a $n = 82000$. Infatti, si ha che per $n = 84000$, $\left\lfloor k^{\frac{1}{m}} \right\rfloor$ risulta essere 4, portando una suddivisione iniziale dello spazio in molte più sottoregioni rispetto a $n = 82000$ dove $\left\lfloor k^{\frac{1}{m}} \right\rfloor$ invece è 3.

Al contrario MSTeusort sembra mantenere un comportamento stabile rispetto a m mantenendo un tempo di esecuzione sempre minore rispetto agli algoritmi esatti.

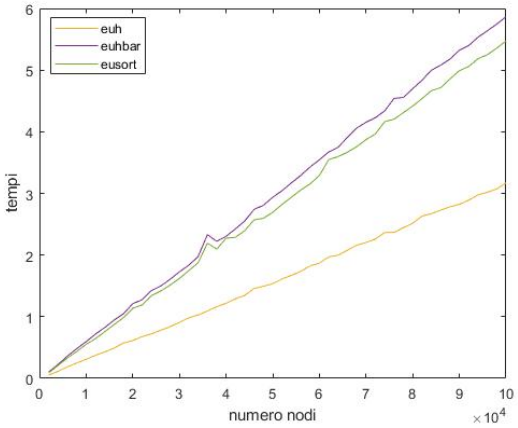
Quindi sembrerebbe che in dimensione bassa rispetto al numero di nodi, l'euristica più rapida sia MSTeuh mentre in caso contrario pare essere MSTeusort.

Figure 3: Tempi di esecuzione delle euristiche in dimensione 2,3 e 7

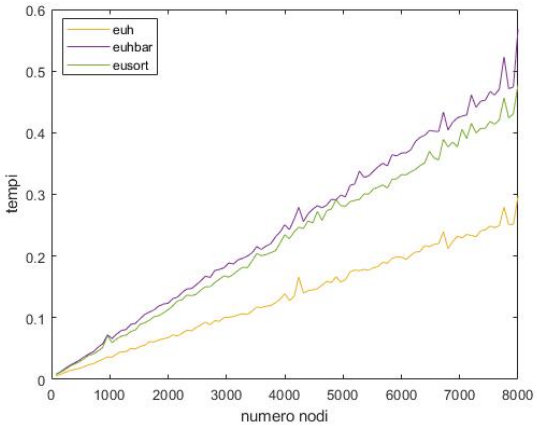
n=8000 n=100000



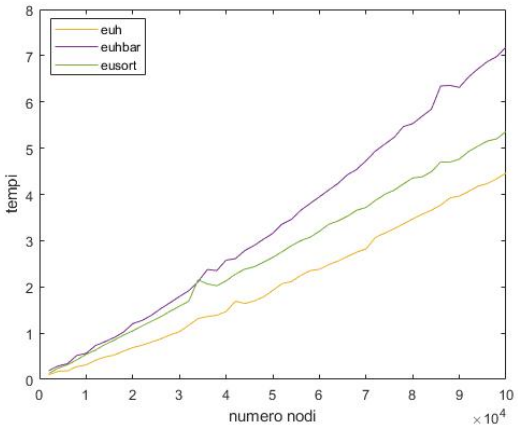
Dim 2



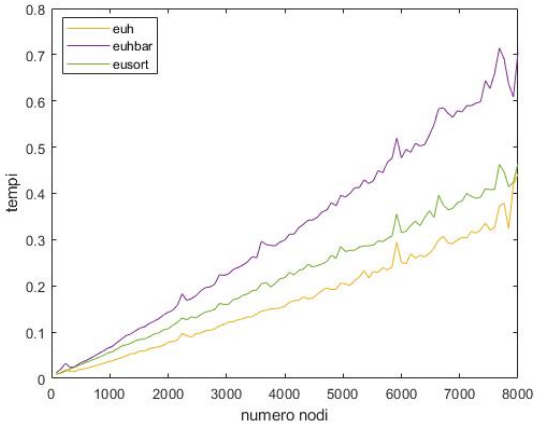
Dim 2



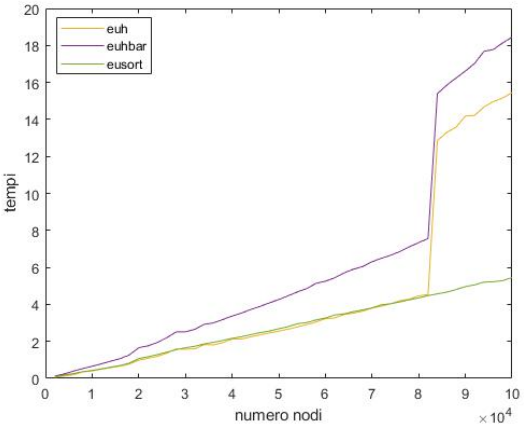
Dim 3



Dim 3



Dim 7



Dim 7

Osservando i pesi si nota come le euristiche producano alberi il cui peso presenta un andamento simile al peso del MST, mantenendo un costo assimilibile al costo teorico di $O(n^{1-\frac{1}{m}})$ per ogni m. Per una dimostrazione dell'andamento teorico del peso del MST si veda [5].

Si registrano, però, nuovamente delle differenze dipendenti da m. Infatti per m=2,3 MSTeusort ha prodotto lo spanning tree di peso minore fra le euristiche, con alberi di peso al più 1.5 volte quello del MST in dimensione 2 e 1.64 in dimensione 3. Invece per m=13 è MSTeuhbar a restituire l'albero di peso minore, al più 1.1 volte il peso del MST. Per m=7 invece nessuna fra MSTeusort e MSTeubar sembra essere preferibile rispetto all'altra.

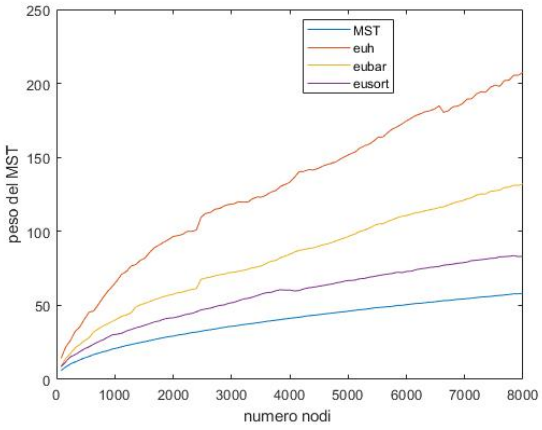
Questa differenza di comportamento sembra essere dovuta di nuovo al fatto che per m=13 a ogni iterazione la presenza di un solo cubo renda l'algoritmo MSTeuhbar una versione modificata di un'algoritmo greedy per generare il MST, dove, invece che scegliere il punto con distanza minima da tutti i nodi già presenti nell'albero, sceglie il nodo di distanza minima dal baricentro dei punti non ancora inseriti, che sarà spesso il punto più vicino al nodo aggiunto all'iterazione precendete.

Si nota anche una differenza di andamento di MSTeuhbar in dimensione 7, di nuovo nel passaggio fra $n = 82000$ e $n = 84000$, che si direbbe essere causata, come per i tempi, dall'aumento del numero di cubi iniziale. Sembrerebbe che questo però non condizioni in modo evidente il peso dell'albero prodotto da MSTeuh, forse a causa dalla scelta casuale dei nodi radice per ogni suddivisione in cubi. Inoltre, in dimensione 3, sia MSTeuh che MSTeuhbar presentano alberi con pesi caratterizzati da un evidente crescita "a scalini", che sembrerebbe, ancora una volta, essere causata dai cambiamenti di $\left\lfloor k^{\frac{1}{m}} \right\rfloor$ dovuti all'utilizzo della parte intera per il suo calcolo (e quindi all'aumento repentino del numero di cubi generati al primo passo per particolari n), ad esempio da 4 a 5 per $n = 3120$ e $n = 3200$ e da 5 a 6 per $n = 6320$ e $n = 6400$ rispettivamente.

Infine notiamo che per ogni m è MSTeuh che genera l'albero di peso maggiore, anche 2 volte il peso del MST.

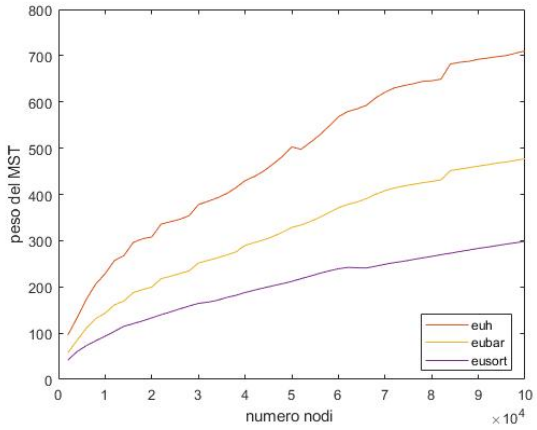
Figure 4: Peso degli alberi

n=8000

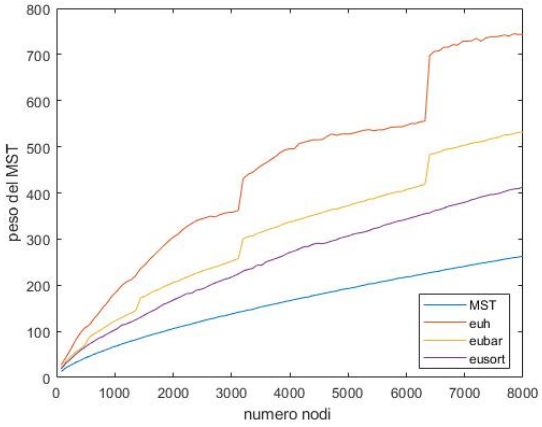


Dim 2

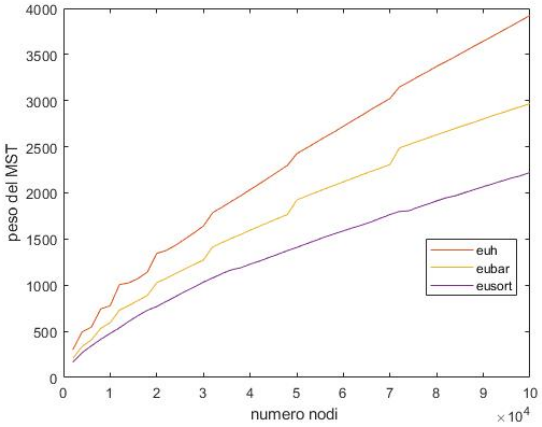
n=100000



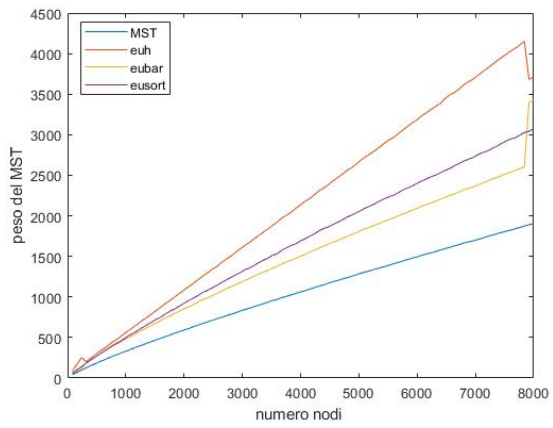
Dim 2



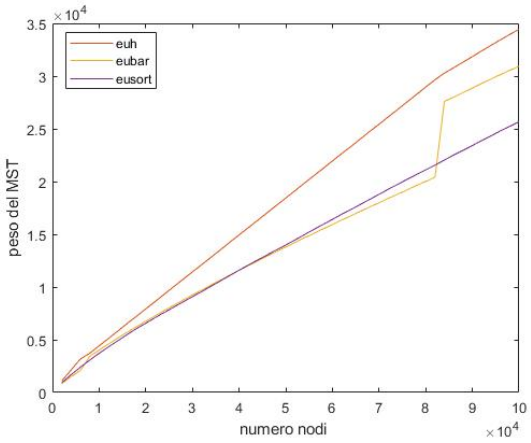
Dim 3



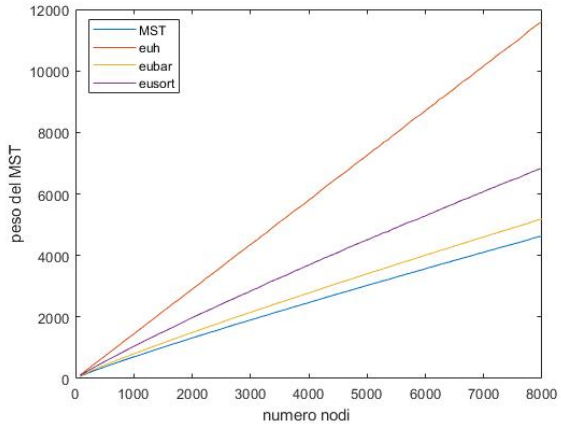
Dim 3



Dim 7



Dim 7



Dim 13

Osservando i gradi degli alberi generati tramite le euristiche, si evidenzia come in dimensione 2 e , soprattutto, in dimensione 3 l'euristica eusort genera un albero le cui frequenze dei nodi di grado k si avvicninano molto a quella del MST per ogni k, con un errore medio che non supera 0.05 per la dimensione 2 e 0.02 per la dimensione 3. Invece in dimensione 13 è MSTeuhbar ad approssimare i gradi del MST, con un errore assoluto medio che non supera 0.03 per ogni grado, anche se riteniamo che questo dipenda di nuovo dal comportamento sopracitato.

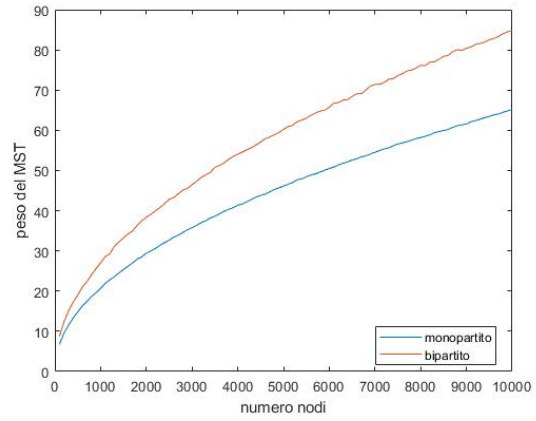
Caso bipartito

In questa seconda parte viene confrontato il comportamento del MST di un grafo euclideo con il MST nel caso di un grafo euclideo bipartito.

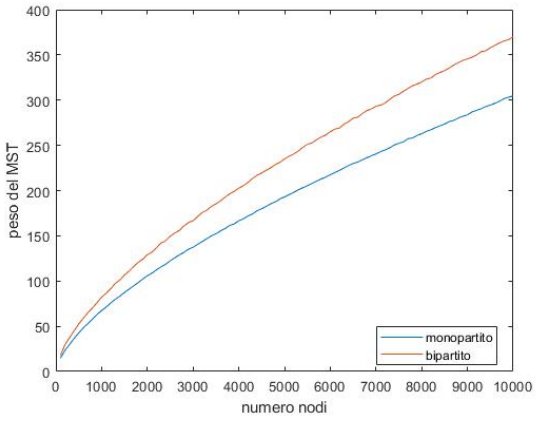
Per il calcolo dei MST è stato usato MSTJ nel caso monopartito, e una sua variante, MSTJbi, che tiene conto della suddivisione dei nodi, ponendo a ∞ il peso degli archi che collegano i nodi appartenenti allo stesso gruppo, per il caso bipartito.

Dai test, si evince che il peso nel caso bipartito risulti mediamente maggiore rispetto a quello monopartito, ma le due crescite siano comparabili, mantenendo l'andamento asintotico noto per il caso monopartito (si veda [5]) di $O(n^{1-\frac{1}{m}})$ in dimensione m. Inoltre i due pesi sembrano tendere a coincidere all'aumentare della dimensione passando da una differenza relativa media del 30% in dimensione 2 a 7% in dimensione 13.

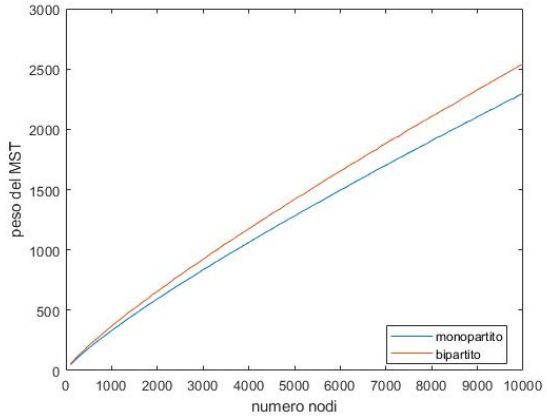
Figure 5: Peso dei MST



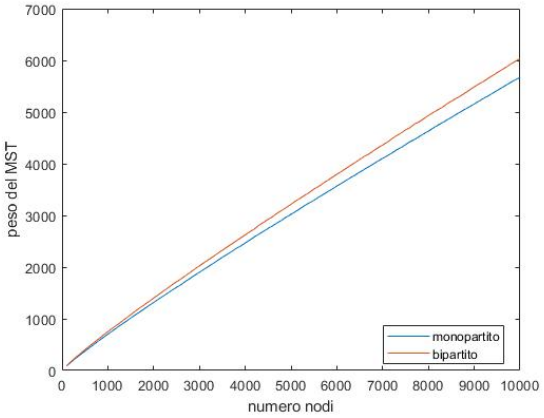
Dim 2



Dim 3



Dim 7

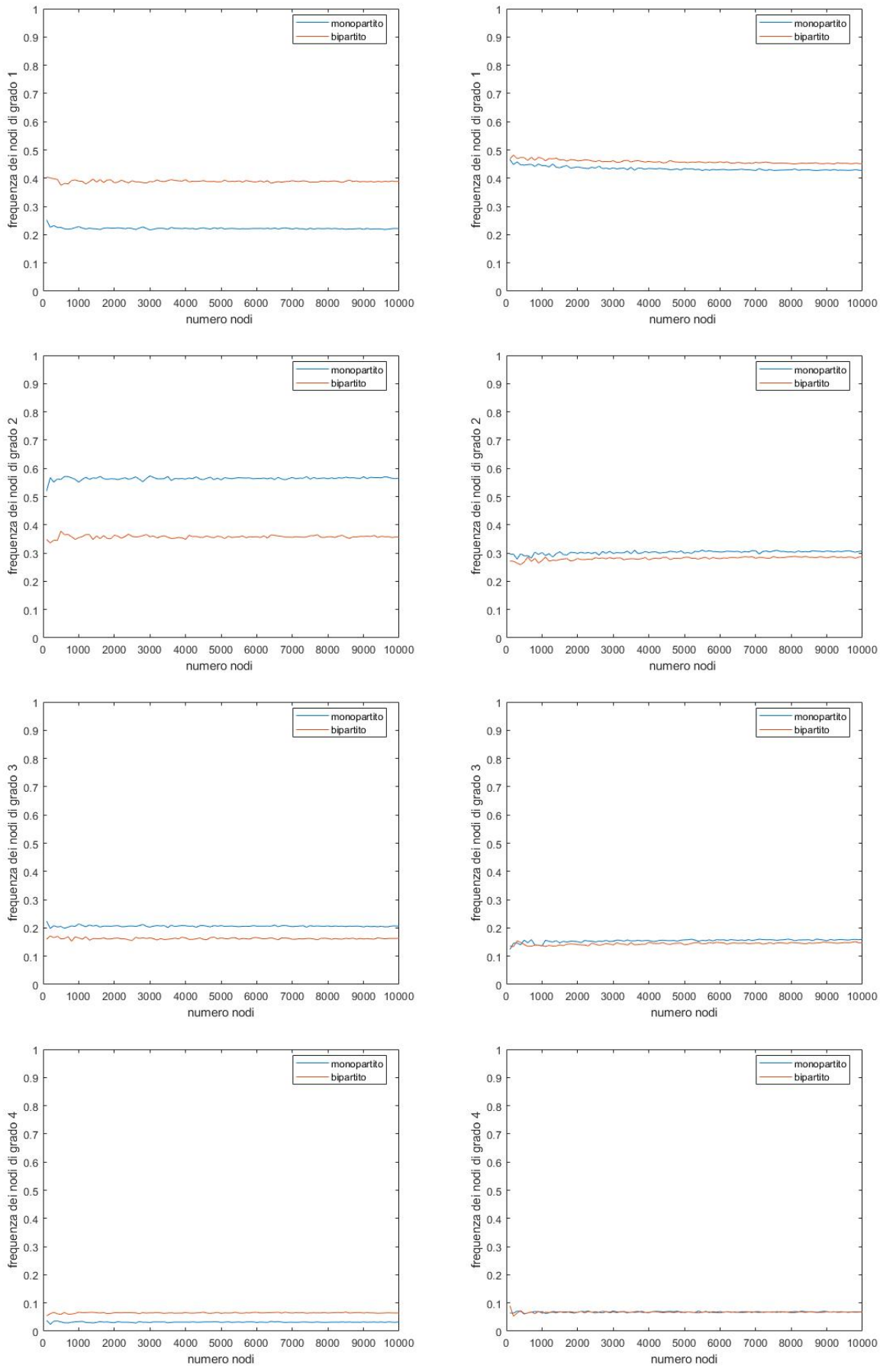


Dim 13

La distribuzione dei gradi invece appare diversa: infatti, nonostante in entrambi i casi sembra esserci una frequenza media per i nodi di ciascun grado alla quale ogni MST sembra tendere (nel caso monopartito questo fatto è supportato da risultati teorici, si veda [1]), nel caso bipartito e monopartito queste sono diverse come evidenziato dai grafici:

Figure 6: Frequenza dei nodi di grado 1,2,3,4

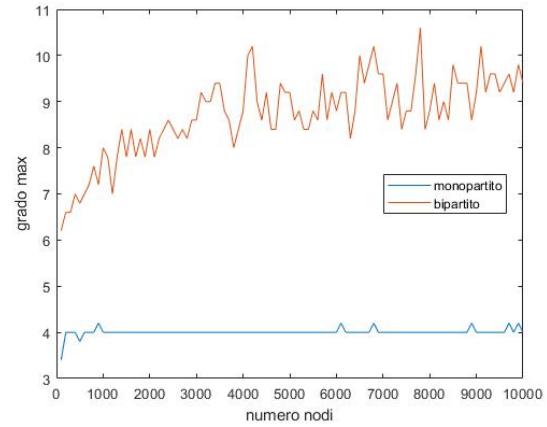
Dim 2 Dim 13



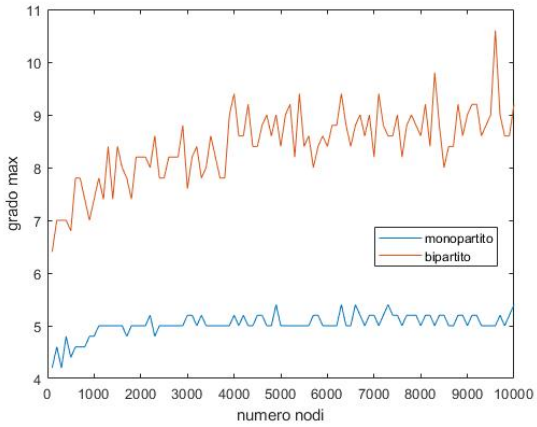
Tuttavia, anche in questo caso, pare che all'aumentare di m le differenze fra le frequenze del caso monopartito e del caso bipartito, sembrano ridursi.

Infine, nel caso bipartito il fatto che i gradi dei nodi non siano limitati da una costante dipendente dalla dimensione come invece nel caso monopartito(si veda [1], Lemma 4), si riflette nei risulati dei test in dimensione 2, dove il grado massimo dei nodi nel caso bipartito supera il limite teorico del caso monopartito, 6. Inoltre sembrerebbe che il fatto che i punti siano scelti in modo casuale, porti il grado massimo a crescere molto lentamente all'aumentare di n. All'aumentare di m invece, il grado massimo,cresce nel caso monopartito, ma assume valori inferiori rispetto ai limiti teorici, mentre sembra verificarsi una lieve crescita anche nel caso bipartito. Come per i pesi e le frequenze, anche per il grado massimo sembrano ridursi le differenze fra grafo monopartito e bipartito all'aumentare di m.

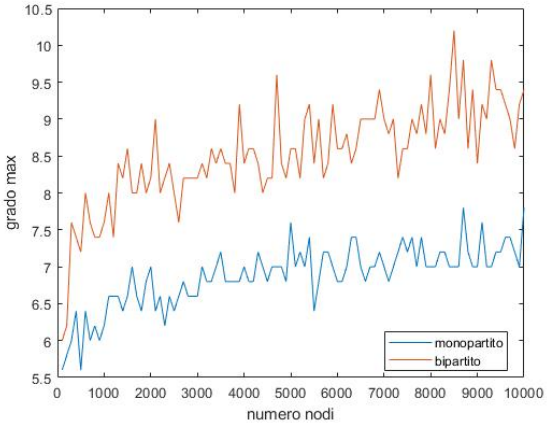
Figure 7: Grado massimo dei nodi



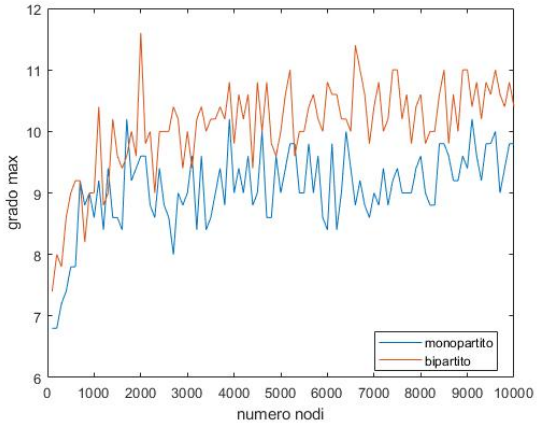
Dim 2



Dim 3



Dim 7



Dim 13

Bibliografia

1 Aldous, D., Steele, J.M. Asymptotics for Euclidean minimal spanning trees on random points. Probab. Th. Rel. Fields 92, 247–258 (1992).

2 Clementi A.E.F., Di Ianni M., Monti A., Lauria M., Rossi G., Silvestri R. (2005) Divide and Conquer Is Almost Optimal for the Bounded-Hop MST Problem on Random Euclidean Instances. In: Pelc A., Raynal M. (eds) Structural Information and Communication Complexity. SIROCCO 2005. Lecture Notes in Computer Science, vol 3499. Springer, Berlin, Heidelberg.

3 Riva A., 2019, The random minimum spanning tree problem, tesi di laurea magistrale, Università degli studi di Milano.

4 M. Shamos, D. Hoey. "Closest-point problems". Proceedings of the 16th Annual Symposium on Foundations of Computer Science (1975), pp. 151-162.

5 J.M. Steele, (1988) "Growth Rates of Euclidean Minimal Spanning Trees with Power Weighted Edges", Annals of Prob., 1767-1787.