Università di Palermo - Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Corso di INTELLIGENZA ARTIFICIALE

Prof. Salvatore Gaglio

Il problema di Steiner nelle reti

all. ing. Giuseppe Lo Presti
matricola 0322605

Relatori:
ing. Giuseppe Di Fatta
dott. Giuseppe Lo Re

Anno Accademico 1999 – 2000

Palermo, Settembre 2000

Si ringrazia per la collaborazione il CE.R.E. - Centro di studio sulle Reti di Elaboratori.

SOMMARIO

1. Introduzione

Negli ultimi anni, l'evoluzione delle tecnologie legate alla telematica ha portato ad una diffusione su larga scala delle reti di calcolatori, le quali si sono sviluppate fino alla copertura quasi mondiale dell'attuale rete Internet. In questo scenario, grazie al sempre crescente numero di utenti, sono stati sviluppati un grande numero di studi teorici e di applicazioni specifiche per le reti di computer.

Questo lavoro si propone di affrontare il cosiddetto *problema di Steiner*, che si inquadra nell'ambito della teoria dei grafi, con particolare riferimento alle reti di computer. Il problema di Steiner è in generale un problema di interconnessione al minimo costo possibile, ed è di importanza centrale, come vedremo, nell'attuale panorama delle telecomunicazioni.

Nel capitolo 2 verrà definito rigorosamente il problema in termini formali e verranno esaminate alcune tecniche, tratte dalla letteratura, che consentono di calcolare una soluzione. Successivamente, nel capitolo 3 sarà presentato il sistema implementato che consente di sperimentare le prestazioni delle varie tecniche esistenti su una piattaforma comune; i risultati della sperimentazione condotta in laboratorio sono presentati nel capitolo 4, dove verranno tratte alcune conclusioni sulle prestazioni di queste tecniche. Infine, nel capitolo 5 sono illustrate brevemente alcune applicazioni del problema di Steiner in altri settori.

1.1. Il multicasting

Per caratterizzare i modi possibili di comunicazione su una rete di computer, si può prendere in considerazione il numero di destinatari dal mittente: così, la comunicazione uno a uno viene detta *unicast*, mentre la comunicazione da uno a tutti viene detta *broadcast*; si definisce *multicast* la trasmissione simultanea di dati da una sorgente verso molte destinazioni.

Nonostante le attuali reti telematiche ed Internet offrano un supporto piuttosto limitato a questo tipo di comunicazione, negli ultimi anni il problema sta suscitando notevoli interessi soprattutto per le applicazioni di tipo multimediale, quali la teleconferenza, l'insegnamento a distanza, o la trasmissione di un evento video, che stanno diventando determinanti nel panorama della telematica odierna. Queste applicazioni coinvolgono grandi quantità di dati video ed audio, pertanto trarrebbero notevoli benefici da un multicasting svolto in modo efficiente, dove l'efficienza è mirata principalmente all'utilizzo ottimale delle risorse di banda offerte dalla rete, tenuto conto delle esigenze dei destinatari.

La soluzione tipica adottata generalmente dai protocolli di multicast consiste nel generare un albero che dalla sorgente raggiunga tutte le destinazioni necessarie; è possibile ottimizzare in più direzioni tale costruzione, in funzione dei diversi vincoli imposti dal tipo di applicazione in

esame: minimizzando il costo totale dell'albero, cioè la somma dei costi di tutti i collegamenti impiegati, dove il costo è un parametro noto; oppure, nel caso di applicazioni interattive *real-time*, è necessario mantenere il ritardo massimo sotto una determinata soglia, fissata generalmente dal requisito di QoS richiesto dal destinatario, pertanto si può minimizzare il ritardo dalla sorgente, oltre che il costo; un ulteriore vincolo studiato in [Bau96] riguarda il grado massimo consentito dei nodi dell'albero.

Il caso più generale di *video multicast*, che costituisce l'applicazione di riferimento, può prevedere ulteriori vincoli, rappresentanti scenari via via più realistici. Ad esempio, se come accade normalmente la topologia della rete non è completamente nota al sorgente, può essere necessario distribuire la generazione dell'albero di multicast tra i nodi partecipanti alla trasmissione o tra tutti i nodi della rete, utilizzando un opportuno algoritmo distribuito; oppure può accadere che alcuni utenti destinatari si uniscano al gruppo o lo abbandonino durante la trasmissione: questo comporta il riarrangiamento on-line dell'albero di multicast quando le modifiche effettuate rendono l'albero troppo inefficiente. Infine si può tenere conto delle diverse richieste di banda dei vari destinatari, in particolare nel caso in cui la rete disponga di un meccanismo di prenotazione delle risorse: in questo caso i costi dei collegamenti diventano funzione della banda effettivamente impegnata.

Per alcuni degli scenari citati sono stati sviluppati diversi protocolli di multicast ad-hoc (cfr. ad esempio *ARIES* [Bau97], *CRCDM* [RMM99], ecc.); la loro base comune è in ogni caso la generazione dell'albero di multicast, problema su cui ci concentreremo nel seguito tralasciando i vincoli aggiuntivi. L'obiettivo sarà quindi quello di generare un albero di multicast che dal sorgente raggiunga tutte le destinazioni e che abbia il costo minimo. Nel prossimo capitolo si vedrà come questo problema sia un'istanza del problema di Steiner nelle reti.

2. Il problema di Steiner

Come già accennato, il problema di Steiner è un problema di interconnessione ottima, nell'ambito di una topologia assegnata. Le varianti topologiche più importanti sono:

- *problema di Steiner nelle reti*: la topologia è data da un grafo G, e la metrica è indotta dalla funzione di costo w_{ij} degli archi del grafo; questa è anche la versione più generale del problema, ed è quella trattata in questa sede;
- *problema di Steiner rettilineo*: la topologia è data dal piano $\mathbf{R}^2$ con la metrica indotta dalla distanza Manhattan[1];
- *problema di Steiner euclideo*: anche in questo caso la topologia è data dal piano $\mathbf{R}^2$, con la metrica indotta dalla distanza euclidea;

Ciò premesso, la definizione formale del problema di Steiner nelle reti è la seguente:

DATI una rete connessa e non diretta $G = (V, E, w)$, $w : E \to \mathbf{R}$, e un insieme $Z \subseteq V$,

TROVARE il sottografo connesso $G_S = (V_S, E_S, w)$ di G tale che:

a) $Z \subseteq V_S$;

b) $w(G_S) = \sum_{(i,j) \in E_S} w(i, j)$ sia minimo.

L'insieme Z è detto, con riferimento al problema trattato, insieme di multicast, mentre l'insieme $V_S - Z$ viene denotato con S ed è costituito dai nodi detti di Steiner; se la funzione di costo $w(i, j)$ degli archi assume sempre valori positivi il sottografo soluzione è un albero, detto **albero di Steiner**, che verrà indicato nel seguito con T_Z, essendo questo il caso cui ci si riferirà normalmente. Si pone inoltre $n = |V|$, $m = |E|$, $p = |Z|$, $s = |S|$.

Si può notare che alcuni casi limite del problema così esposto ricadono in problemi ben noti della teoria dei grafi:

- $p = 2$: si tratta di trovare il percorso più breve che connetta i due nodi dell'insieme Z; il problema viene risolto dall'algoritmo di Dijkstra, che restituisce un albero di cammini minimi da un nodo verso tutti gli altri nodi della rete;
- $p = n$, cioè $Z = V$: si tratta di calcolare il *minimum spanning tree* o MST del grafo dato; in questo caso il problema viene risolto dall'algoritmo di Prim o dalla variante di Kruskal [CLR97].

[1] La distanza Manhattan tra due punti, data da $d = |x_2 - x_1| + |y_2 - y_1|$, è una misura di distanza per i casi in cui è proibito uno spostamento in diagonale.

D’altra parte, sebbene questi casi limite vengano risolti in tempo polinomiale dagli algoritmi citati, è stato dimostrato che il problema generale con $2 < p < n$ è **NP-completo**, e resta tale anche quando la rete è planare o ha tutti gli archi di costo unitario; dunque anche se per problemi di piccole dimensioni si riesce a trovare la soluzione ottima tramite una ricerca esaustiva con un tempo esponenziale, è essenziale trovare tecniche euristiche per calcolare, in tempo polinomiale, una soluzione ammissibile quanto più vicina possibile alla soluzione ottima. Tali tecniche generalmente si basano anche sugli algoritmi di Dijkstra e di Prim, che costituiscono di fatto gli strumenti di base per il calcolo dell’albero finale.

Per misurare e confrontare le prestazioni delle varie euristiche si considerano generalmente due figure di merito dipendenti solo dall’algoritmo in esame:

- la *competitività*[2], che rappresenta una misura della bontà della soluzione trovata rispetto alla soluzione ottimale; è definita infatti come segue:

  $$c_H = w(T_H) / w(T_Z)$$

  dove T_H è l’albero restituito dall’euristica H. Per questo parametro sono noti per via analitica dei limiti superiori caratteristici di ogni euristica. Poiché però risulta proibitivo, per quanto detto, calcolare la soluzione ottima per problemi di grandi dimensioni, in questi casi si ridefinisce la competitività come $c'_H = w(T_H) / \min\{w(T)\}$, dove il minimo è calcolato sull’insieme di tutti i risultati delle euristiche prese in considerazione;
- la *complessità computazionale*, espressa come è noto dalla notazione $O(f(n))$, misura l’ordine di grandezza del tempo di calcolo rispetto alle dimensioni del problema, indipendentemente dalla bontà della soluzione trovata.

Nei prossimi paragrafi queste proprietà verranno analizzate con riferimento alle euristiche più importanti pubblicate in letteratura.

2.1. Algoritmi ottimi

2.1.1. Strategie di ricerca della soluzione

Poiché come accennato il problema di Steiner è NP-completo, qualsiasi strategia di ricerca deve tenere conto di uno spazio di ricerca di dimensioni esponenziali rispetto ad n. In particolare, esso è costituito da tutte le possibili scelte di $V_S \subseteq V$ per le quali il sottografo di G indotto da V_S è connesso, dove si è in presenza di una soluzione ammissibile solo se V_S contiene Z, cioè al più in 2^{n-p} casi nel caso peggiore.

In questo problema, inoltre, vi è un ulteriore elemento che complica la ricerca: lo spazio di ricerca è poco strutturato, poiché attualmente non si è trovata nessuna funzione di valutazione

[2] *cost competitiveness* nella letteratura anglosassone.

che possa guidare la ricerca e la costruzione dell'albero soluzione in maniera informata, stimando una "distanza" dall'obiettivo. In altre parole, non esiste nessun punto dello spazio di ricerca che si possa configurare come punto "più vicino" al risultato rispetto al punto iniziale. In più, non si dispone di un test di ottimalità che consenta di arrestare la ricerca, pertanto ogni algoritmo ottimo è obbligato a estendere effettivamente la ricerca a tutte le soluzioni ammissibili, per avere la certezza di essere completo. Ecco che un algoritmo ottimo non può basarsi sui classici algoritmi di ricerca informata quali A* o i suoi derivati, e si deve comportare come una ricerca non informata in ampiezza. D'altra parte, in questo problema la soluzione è tutta nella configurazione finale ed è indipendente dal percorso di ricerca compiuto per ottenerla; un algoritmo a miglioramento iterativo, seppure non ottimo, sembrerebbe allora preferibile, e in effetti esistono tecniche euristiche di questo tipo: però non si dispone del gradiente della funzione di costo, pertanto può risultare estremamente difficile mettere a punto un algoritmo, ad esempio basato sul *simulated annealing*, che sia competitivo con i metodi euristici disponibili, dei quali si tratterà più avanti.

Un modo per diminuire la dimensione del problema indipendentemente dall'algoritmo utilizzato, consiste nell'applicare alcune tecniche di riduzione, le quali sfruttando le caratteristiche del grafo e dell'insieme Z in esame conducono ad un problema equivalente di dimensioni minori sia in n sia in p, la cui soluzione consente di trovare immediatamente la soluzione originaria; in particolare, queste tecniche identificano sia archi e S-vertici che fanno parte dell'albero di Steiner ottimo, sia archi e S-vertici che possono essere eliminati, ed hanno tutte complessità al più polinomiale in n. Si è verificato che il problema equivalente risulta ridotto in alcuni casi del 60% in n e del 20% in p, il che comporta risparmi decisamente elevati nel tempo di calcolo degli algoritmi ottimi, che verranno illustrati nei prossimi paragrafi.

2.1.2. Spanning Tree Enumeration Algorithm (STEA)

Questo algoritmo, proposto da Hakimi, è molto semplice nella sua formulazione: l'albero di Steiner ottimo può essere trovato enumerando tutti i MST delle sottoreti connesse di G, indotte dai sottoinsiemi V_S di V tali che $Z \subseteq V_S \subseteq V$. È quindi paragonabile ad una ricerca non informata in uno spazio di ricerca organizzato come un albero di profondità 1 e fattore di ramificazione $b = 2^{n-p}$ nel caso peggiore. La complessità spaziale di questo algoritmo è $O(n)$, dato che deve essere memorizzato solo il MST migliore durante l'enumerazione, mentre la complessità temporale è $O(p^2 2^{n-p})$, corrispondente al numero di MST da calcolare nel caso peggiore; pertanto STEA è polinomiale in p ed esponenziale in n, fatto che limita notevolmente il range di problemi cui è applicabile[3].

[3] Ad esempio, è stata necessaria più di mezz'ora di calcolo per un caso con $n = 200$ e $p = 10$, ridotto con le citate tecniche a $n = 70$ e $p = 7$.

2.1.3. Dynamic Programming Algorithm (DPA)

L'algoritmo presentato in questo paragrafo, proposto da Dreyfus e Wagner [DW72], utilizza di un approccio alla risoluzione dei problemi di decisione sviluppato alla fine degli anni '50 da R. Bellman[4] e noto con il nome di **programmazione dinamica**. Essa si avvale di una proprietà di separabilità delle politiche ottime, secondo cui *una politica ottima è costituita da una sequenza di azioni tali che, effettuata la prima azione, le azioni restanti costituiscono una politica ottima per lo stato raggiunto dopo la prima azione*; questo principio molto generale si applica con successo in svariati settori applicativi, dal riconoscimento del parlato, alla demodulazione ottima di un segnale modulato (algoritmo di Viterbi), al controllo ottimo, nonché alla ricerca dei cammini minimi in una rete; infatti, la semplice enumerazione esaustiva di tutte le politiche di lunghezza n, di complessità $O(a^n)$ se vi sono a azioni possibili ad ogni passo, viene ridotta a $O(ans)$, con s pari al numero di stati raggiungibili dopo ogni azione, normalmente abbastanza limitato. Quindi la programmazione dinamica consente di costruire efficacemente una soluzione ottima quando sono disponibili le soluzioni ottime dei sottoproblemi del problema originario; queste a loro volta vengono costruite da istanze ancor più piccole, iniziando da istanze di dimensioni tali che si possono risolvere direttamente.

In questo caso, si utilizza l'unico principio guida ottimo nella costruzione dell'albero, cioè il fatto che il percorso che connette due punti di diramazione consecutivi nell'albero coincide con il cammino minimo tra quei nodi, esteso secondo una regola che gli autori definiscono *proprietà di decomposizione ottima* dell'albero di Steiner: per ogni nodo $z \in Z$, se $p \geq 3$ è possibile trovare un nodo $s \in S$ e un insieme $D \subset Z$, tale che l'albero T_Z sia costituito da tre componenti:
- la prima componente è l'albero di Steiner per l'insieme $D \cup \{s\}$;
- la seconda è l'albero di Steiner per l'insieme $(Z - D - \{z\}) \cup \{s\}$;
- l'ultima connette z e s lungo il cammino minimo, cioè è ancora l'albero di Steiner per $\{s, z\}$.

Se allora si calcolano gli alberi di Steiner per tutti gli insiemi D di dimensione k, si possono usare queste soluzioni per gli insiemi di dimensione $k + 1$, aggiungendo di volta in volta un nodo $z \in Z$ fino a coprire l'intero insieme, trovando così la soluzione cercata in p passi di decisione. Il numero di stati s dopo ogni passo è ancora esponenziale in p, essendo correlato ai sottoinsiemi D di Z, ma non si hanno più $O(2^n)$ politiche da esplorare.

Questa strategia risulta pertanto paragonabile ad una ricerca non informata in ampiezza con profondità pari a p. Infatti, la complessità spaziale è adesso $O(n\, 2^{p-1})$, dato che è necessario memorizzare le soluzioni intermedie per ogni sottoinsieme D, mentre la complessità temporale risulta come previsto $O(n^3 + n^2(2^{p-1} - p - 1) + n(3^{p-1} - 2^p + 3)/2)$, dove il primo termine è respon-

[4] R. BELLMAN, *Dynamic Programming*, Princeton University Press, Princeton, 1957.

sabile del calcolo di tutte le distanze minime da ogni nodo e gli altri sono dovuti all'enumerazione esaustiva dei sottoinsiemi *D* dell'insieme *Z*. Essendo la complessità solo polinomiale in *n*, è possibile eseguire questo algoritmo su problemi relativamente più grandi[5]; d'altra parte, questo vantaggio è stato ottenuto a spese dello spazio di memoria richiesto per conservare le soluzioni intermedie, che diventa insostenibile già con $p = 20$ ed $n = 200$: i limiti di applicabilità di questo algoritmo sono quindi dettati, come peraltro spesso accade in questi casi, dalla memoria disponibile prima che dal tempo di calcolo.

L'algoritmo DPA è esposto di seguito:

```
function DPA(d, Z) returns il costo dell'albero di Steiner ottimo per l'insieme Z
    inputs: d(i, j), costo dei cammini minimi (DIJKSTRA) tra i nodi della rete
            Z, l'insieme di multicast
    locals: C, l'insieme Z meno un nodo
            D, sottoinsieme di multicast corrente
            E, ogni sottoinsieme proprio dell'insieme D o C
            S[A, i], costo dell'albero di Steiner ottimo che connette il nodo i all'insieme A

  q ← un nodo Z arbitrario
  C ← Z – {q}
  for each t in C do
      for each j in V do
          S[{t}, j] ← d(t, j);          // calcola gli alberi di dimensione 1
  for m = 2 to |C| – 1  do
      for each D tale che D ⊂ C and |D| = m do
          for each j in V do            // inizializza gli alberi di dimensione m
              S[D, j] ← ∞
          for each j in V do
              u ← ∞
              for each E tale che D[1] ∈ E and E ⊂ D do      // utilizza tutti gli alberi
                  u ← min(u, S[E, j] + S[D – E, j])         // fino alla dim. m-1
              for each i in V do
                  S[D, i] ← min(S[D, i], d(i, j) + u)
  w ← ∞
  for each j in V do
      u ← ∞
      for each E tale che C[1] ∈ E and E ⊂ C do
          u ← min(u, S[E, j] + S[C – E, j])          // utilizza tutti gli alberi calcolati finora
      w ← min(w, d(q, j) + u)
  return w
```

L'avanzamento della ricerca secondo i passi visti prima risulta evidente dai commenti a margine: in particolare, i costi degli alberi per gli insiemi *D* di cardinalità *m* vengono calcolati dai costi di tutti gli alberi precedenti, di cardinalità da 2 a $m – 1$; è interessante notare come la

[5] Nello stesso esempio già citato, bastano solo 5 secondi di calcolo senza applicare le riduzioni.

procedura sia indipendente dall'ordine scelto per enumerare gli insiemi *D*, purché si esauriscano tutti quelli di una cardinalità prima di passare alla successiva.

Si noti che al termine dell'esecuzione di questo algoritmo è noto soltanto il costo complessivo dell'albero, non la sua topologia. Per ottenere l'albero effettivo, è necessario memorizzare, durante le fasi di calcolo dei minimi, anche i nodi per i quali si verifica il minimo; così, con un processo a ritroso – tipico degli algoritmi di programmazione dinamica – si può ricostruire l'albero dal percorso delle scelte effettuate. Questo comporta un'ulteriore richiesta di spazio di memoria, comunque trascurabile rispetto al valore già detto, mentre il tempo di calcolo aggiuntivo necessario non è esponenziale in quanto non si tratta più di una ricerca. In conclusione, il DPA riesce a trovare la soluzione ottima con una complessità non eccessivamente elevata nei casi con $p << n$.

2.2. Euristiche per il problema di Steiner

Sebbene DPA sia un algoritmo ottimo relativamente veloce, questo è pur sempre inapplicabile a molti problemi reali, dove è necessario ricorrere a metodi euristici veloci per il calcolo di una soluzione ammissibile. In questo paragrafo vengono presentati alcuni tra questi metodi: tutti hanno una complessità computazionale lineare nello spazio occupato e polinomiale nel tempo impiegato, e seguono un approccio di tipo *greedy* per la costruzione della soluzione, restituendo un albero soluzione non appena viene terminata la sua costruzione, senza mai effettuare *backtracking*. In ogni caso non vengono utilizzate le tecniche di riduzione viste in precedenza, dato che non porterebbero ad alcun vantaggio in termini di tempo di esecuzione.

Di ogni euristica verrà illustrato il principio guida, peraltro spesso molto semplice, e l'algoritmo nelle sue linee essenziali, seguito da una analisi sia della complessità computazionale sia della competitività; questa rassegna non vuole essere esaustiva, anche perché è possibile definire nuove tecniche da varianti di tecniche esistenti, e d'altra parte la ricerca in questo settore è ancora relativamente attiva.

2.2.1. Pruned Dijkstra Heuristic (PDH)

Una delle euristiche più semplici ed immediate è l'euristica *pruned Dijkstra*, nota anche come *merged shortest paths*. Questa euristica deriva direttamente dall'algoritmo di Dijkstra applicato al nodo sorgente, che restituisce come è noto l'albero dei percorsi minimi dalla radice verso gli altri nodi; se si effettua ripetutamente il pruning delle foglie non appartenenti a *Z* finché ne esistono, si ottiene un albero di Steiner per i nodi *Z*.

Il costo computazionale di questa euristica è estremamente basso, pari a quello dell'algoritmo di Dijkstra, $O(n^2)$. Il limite superiore per la competitività risulta essere pari a *z*

[Voss92], risultato in molti casi inaccettabile, anche se esistono tecniche di post-processing che riescono a migliorare qualunque albero, come si vedrà in seguito.

Anche l'algoritmo di Prim si presta ad un'euristica simile, detta *pruned MST*, ma sebbene la complessità computazionale sia sempre $O(n^2)$, cioè quella dell'algoritmo di Prim, il limite superiore per la competitività è *s*, dunque ancora peggiore della *PDH*.

2.2.2. Distance Network Heuristic (DNH)

Questa euristica è stata proposta indipendentemente da Kou *et al.*, Plesnik ed El-Arbi, ed utilizza ancora il MST come elemento centrale, questa volta però in modo più efficiente, ottenendo prestazioni notevolmente superiori e una complessità computazionale relativamente bassa. L'algoritmo comporta la generazione della *rete delle distanze* dei nodi *Z*, da cui il nome dell'euristica, la quale è un grafo totalmente connesso che ha per vertici i nodi *Z* e i cui archi hanno un costo pari al costo del cammino minimo tra i nodi della rete di partenza. La pseudocodifica dell'algoritmo è mostrata di seguito:

function *DNH*(*G*, *Z*) **returns** T_{DNH}, albero di Steiner per *Z* nel grafo *G*
  locals *T*, albero di Steiner
    DN, distance network indotto dall'insieme *Z*
    G_Z, sottografo
  $DN \leftarrow$ *MAKE-DISTANCE-NETWORK*(*Z*)
  $T \leftarrow MST(DN)$
  $G_Z \leftarrow \varnothing$
  for each *e* **in** *T* **do**
    $G_Z \leftarrow G_Z \cup$ cammino minimo in *G* che connette gli estremi di *e*
  $T \leftarrow MST(G_Z)$
  elimina da *T* tutte le foglie $\notin Z$
  return *T*

È stato dimostrato [Win87] che per la competitività di DNH vale la relazione:

$$c_{\text{DNH}} \leq 2\left(1-\frac{1}{l}\right)$$

l è il numero di foglie dell'albero, sempre minore o uguale a *p*.

Infine, il costo computazionale è dato da $O(pn^2)$, dato che il calcolo della rete delle distanze comporta l'esecuzione dell'algoritmo di Dijkstra per *p* volte, mentre i passi successivi influiscono al massimo per $O(p^2)$ al costo totale.

2.2.3. Shortest Path Heuristic (SPH)

L'euristica SPH, proposta da Takahashi e Matsuyama [TM80], produce dei risultati molto buoni nonostante la semplicità della strategia di costruzione. Infatti, SPH inizializza l'albero con un nodo *Z* arbitrario, ad esempio il sorgente; successivamente unisce i nodi di multicast secondo

la loro distanza dall'albero, finché non vengono connessi tutti. In particolare, l'algoritmo si può delineare come segue:

```
function SPH(G, Z) returns T_SPH, albero di Steiner per Z nel grafo G
    locals T, albero di Steiner
    T ← {z_0}
    while(T non contiene tutti i nodi Z)
        connetti T con il nodo in Z – T più vicino tramite il cammino più breve
    return T
```

È noto [Win87] che la competitività di SPH soddisfa la relazione:

$$c_{\mathrm{SPH}} \leq 2\left(1-\frac{1}{p}\right).$$

Inoltre, il costo computazionale nel caso peggiore è dato da $O(pn^2 + np^2)$, dove il primo termine dipende dal calcolo delle distanze minime di ogni nodo Z dall'albero, mentre il secondo termine tiene conto del costo di ricerca del nodo migliore da unire ad ogni passo; per dimostrare questo secondo termine, si consideri il ciclo principale dell'euristica: esso viene eseguito p volte per esaurire l'insieme Z, cercando ogni volta la distanza minima tra T (costituito da $O(n)$ nodi) e i restanti nodi Z (al più p), per un totale di $O(np^2)$ confronti.

Questa euristica ha originato diverse varianti, che tendono ad eliminare la dipendenza del risultato dalla scelta del nodo iniziale; le più importanti sono la K-SPH e la SPH-Z, analizzate nel seguito.

2.2.4. Kruskal based – Shortest Path Heuristic (K-SPH)

La K-SPH è una variante della SPH ispirata all'algoritmo di Kruskal per il calcolo del MST: anziché partire con un nodo, K-SPH gestisce una "foresta" di alberi, i quali vengono uniti a formare l'albero di Steiner. L'algoritmo è infatti il seguente:

```
function K-SPH(G, Z) returns T_KSPH, albero di Steiner per Z nel grafo G
    locals F, foresta dei sottoalberi della soluzione
    F ← Z
    while(F non è connessa)
        connetti i due sottoalberi più vicini in F tramite il loro cammino più breve
    return F
```

Poiché la crescita dell'albero avviene senza privilegiare nessun nodo, le prestazioni di K-SPH sono generalmente migliori, come si vedrà nel capitolo 4. D'altra parte, il costo computazionale nel caso peggiore è $O(n^3 + pn^2)$, dato che bisogna eseguire l'algoritmo di Dijkstra per tutti i nodi dell'albero ed eseguire la ricerca dei due sottoalberi più vicini, anche se una valutazione più rigorosa è $O(n^2(p + s) + p(p + s)^2)$, con $p + s << n$ nei casi in esame. Infine, si può dimostrare che la competitività ha lo stesso limite superiore della SPH.

Infine, poiché la crescita di ogni sottoalbero è relativamente indipendente dagli altri, l'euristica K-SPH si presta bene ad una implementazione distribuita, ove il protocollo di multicast lo richiedesse.

2.2.5. Repetitive Shortest Path Heuristics

Un altro metodo per eliminare la dipendenza di SPH dalla scelta del nodo iniziale è quello di ripetere l'algoritmo SPH per diversi nodi iniziali e scegliere il risultato migliore; in particolare, le euristiche ripetitive più comuni sono:

- SPH-Z: ripete SPH p volte, una per ogni nodo Z; il costo computazionale risulta quindi pari a $O(pn^2 + np^3)$, tenuto conto che i dati sulle distanze minime vengono calcolati una volta soltanto.
- SPH-V: ripete SPH n volte, una per ogni nodo della rete; se il nodo iniziale non è di multicast, viene temporaneamente unito a Z, e poi viene eliminato. In questo caso, il costo computazionale risulta pari a $O(n^3 + n^2p^2)$, comportando il calcolo di *tutte* le distanze minime, ed è l'euristica più costosa presa in considerazione.
- SPH-ZZ: per ogni coppia di nodi Z, inizializza l'albero con il loro cammino minimo, poi esegue SPH; dunque ripete SPH per $p(p\text{-}1)$ volte, con un costo computazionale dell'ordine di $O(pn^2 + np^4)$.

Sebbene la competitività di tutte le euristiche ripetitive abbia lo stesso limite teorico della SPH, le prestazioni effettive risultano spesso notevolmente migliori; d'altra parte l'elevato costo computazionale le configura come soluzione subottima di confronto piuttosto che come metodo da implementare in un protocollo di multicast reale.

2.2.6. Average Distance Heuristic (ADH)

Questa euristica, proposta da Rayward-Smith e Clare [Ray83], procede similmente alla K-SPH, ma è dotata di un meccanismo più sofisticato per la scelta dei sottoalberi da unire: anziché unire direttamente i due sottoalberi più vicini, viene scelto un nodo intermedio v^* da cui passare, tale che sia il più "centrale" per un gruppo di sottoalberi, in modo da favorire il successivo collegamento degli altri alberi. La misura di centralità suggerita dagli autori consiste nel calcolo di una funzione di merito per ogni nodo $v \in V$ secondo il seguente procedimento:

1. si calcolino le distanze di v da tutti i sottoalberi correnti;
2. si ordinino in ordine decrescente; sia $d_i(v)$ la distanza di v da T_i, con $d_i(v) \le d_{i+1}(v)$; se $v \in Z$, si avrà $d_1(v) = 0$;
3. sia $f(v) = \min_{2 \le r \le k} \left\{ \sum_{i=1}^{r} \frac{d_i(v)}{r-1} \right\}$, dove k è il numero dei sottoalberi, inizialmente pari a p e via via decrescente fino a 2; se $v \in Z$, si può dimostrare che $f(v) = d_2(v)$;

ciò premesso, si sceglie v^* in modo da minimizzare $f(v)$ su tutti i vertici della rete, e si uniscono i due sottoalberi più vicini a v^* attraverso v^* stesso. Quando $v^* \in Z$, ADH si comporta esattamente come K-SPH, in quanto K-SPH minimizza $f(v) = d_2(v)$ per tutti i $v \in Z$; altrimenti, v^* è il nodo "più centrale", secondo ADH, per gli r sottoalberi più vicini: la struttura della $f(v)$ suggerisce che nelle successive iterazioni tali sottoalberi vengano uniti attraverso lo stesso nodo v^*, distribuendo il costo complessivo $\sum_{i=1}^{r} d_i(v)$ su $r - 1$ join successivi. Pertanto ADH è una generalizzazione di K-SPH, e il suo comportamento è in generale migliore, soprattutto per problemi di grandi dimensioni, dove la semplice strategia delle euristiche tipo SPH mostra i suoi limiti, come si vedrà nel seguito.

Anche se ADH ha in generale prestazioni migliori, è stato dimostrato che la competitività di ADH è limitata superiormente dallo stesso limite di DNH [WI88]; il costo computazionale è $O(n^3 + np^3)$, dato che per ottenere $f(v)$ è necessario calcolare tutti i cammini minimi ed effettuare ripetutamente gli ordinamenti.

2.3. Post-processing

Un'altra categoria di tecniche euristiche, anziché costruire la soluzione dall'insieme Z, inizia da una soluzione già ammissibile e tenta di migliorarla iterativamente ottenendo in ogni caso una soluzione non peggiore di quella iniziale. Queste tecniche sono dette di *post-processing* perché si applicano successivamente all'esecuzione di una qualsiasi euristica tradizionale.

Una semplice tecnica di post-processing si basa sul MST ed è delineata di seguito:

```
function POST-MST(T, Z) returns T', albero di Steiner per i nodi Z con w(T') ≤ w(T)
    inputs T, albero di Steiner ammissibile
    locals SN, sottorete indotta dai nodi in T
    SN ← MAKE-SUBNET(T)
    T ← MST(SN)
    elimina da T tutte le foglie ∉ Z
    return T
```

In questo modo, non vengono mai aggiunti altri nodi, mentre è possibile che alcuni nodi insieme agli archi incidenti vengano eliminati in seguito al rimescolamento prodotto dal MST, pertanto il costo dell'albero finale è certamente non maggiore del costo iniziale. Il costo computazionale particolarmente basso, $O(n^2)$, è dominato dal calcolo del MST di una rete di $p + s$ nodi.

Esistono altri metodi che sono in grado anche di aggiungere nuovi nodi, nel caso sia opportuno, portando spesso a miglioramenti notevoli.

3. Il sistema implementato

3.1. Struttura generale

Per provare e confrontare gli algoritmi suesposti, è stato messo a punto un sistema software che esegua a richiesta dell'utente ciascuna delle euristiche. Tale software, sviluppato in C++, è dotato di una interfaccia essenziale, che acquisisce dalla riga di comando il file che contiene la topologia della rete, il file che contiene l'insieme di multicast e l'algoritmo da utilizzare, e restituisce un file contenente la topologia dell'albero calcolato, secondo opportuni formati; così è possibile eseguire in batch il programma per diverse volte *(cfr. Cap. 4)*.

In tutti i casi, si è assunto che il nodo sorgente è sempre il primo nodo di multicast e diventa la radice dell'albero, e che sono noti la topologia e i costi, implementando una versione centralizzata delle euristiche.

Ciò premesso, il sistema in generale si presenta secondo una struttura a layer *(cfr. fig. 1)*, ognuno dei quali fornisce dei servizi sempre più sofisticati ai layer superiori; al livello più alto è presente il codice delle varie euristiche, che avvalendosi dei servizi sottostanti risulta in tutti casi particolarmente breve e di semplice leggibilità. D'altra parte, la linea guida durante l'implementazione è stata quella della massima chiarezza e modularità del codice piuttosto che l'ottimizzazione per le migliori prestazioni. Pur non di meno, le prestazioni sono risultate più che buone e con alcuni margini di miglioramento.

<table>
<tr><td colspan="2">main: invocazione dell'euristica scelta dall'utente</td></tr>
<tr><td colspan="2">classi per le euristiche: TPDTree, TSPHTree, …</td></tr>
<tr><td rowspan="2">classe TTree</td><td>virtual calcTree, postMST</td></tr>
<tr><td>join, revert; load, store</td></tr>
<tr><td rowspan="2">classe TNet</td><td>makeSubNet, makeDNet</td></tr>
<tr><td>dijkstra, prim; load, store</td></tr>
<tr><td>classe TQueue</td><td>getHead, insert</td></tr>
<tr><td>–</td><td>primitive del linguaggio C++ e I/O di base</td></tr>
</table>

Figura 1 – struttura generale a layer del sistema.

In particolare, le strutture dati utilizzate in tutte le classi sono basate essenzialmente su array mono- e multi-dimensionali, mentre è stata realizzata ad-hoc solo la coda lineare a priorità *TQueue* necessaria per gli algoritmi di Dijkstra e Prim.

È stata sviluppata inizialmente una classe *TNet* che implementa alcuni servizi di base legati alla rete, quali l'I/O su file, il calcolo dei cammini minimi tramite l'algoritmo di Dijkstra, peraltro incapsulato in una funzione che controlla se il cammino o la distanza richiesti non siano già stati calcolati e memorizzati in precedenza per evitare di ricalcolarli, il calcolo del MST tramite l'algoritmo di Prim, e le procedure di alto livello *makeSubNet*() e *makeDNet*() utili per l'euristica DNH e per *PostMST*.

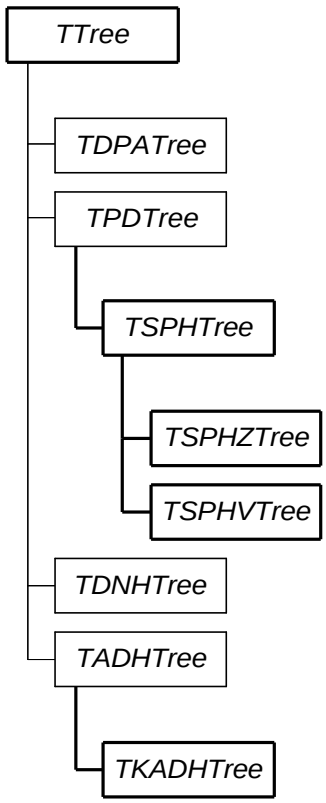


Figura 2 – gerarchia delle classi discendenti da TTree.

Successivamente è stata sviluppata una gerarchia di classi per implementare le varie euristiche, *(v. fig. 2)*. Tutte specializzano la funzione virtuale *calcTree*(), che contiene il corpo dell'euristica, ed ereditano da *TTree* altri servizi specifici per gli alberi, tra cui *join*() che connette due nodi lungo il loro cammino minimo aggiornando tutte le informazioni di stato relative all'albero o ai sottoalberi in formazione, e *revert*(), che "ribalta" un ramo trasformando un nodo in radice e la radice in foglia, necessario quando deve essere gestita una foresta di alberi.

È significativo evidenziare che durante lo sviluppo del sistema, il disegno dell'interfaccia delle classi e l'implementazione delle funzioni di base sono rimasti essenzialmente immutati, mentre l'implementazione dei servizi più avanzati, come *join*(), si è progressivamente evoluta per soddisfare i requisiti di tutte le euristiche; dunque il sistema si è mostrato nel suo complesso flessibile, modulare ed espandibile.

3.2. Gli algoritmi realizzati

Sono stati implementati sia l'algoritmo ottimo DPA, sia le euristiche discusse in precedenza, e cioè PDH, DNH, SPH, SPH-Z, SPH-V, K-SPH, ADH; inoltre è stato implementato il post-processing *POSTMST* in modo da poter essere applicabile al termine di ogni euristica.

In particolare, per l'algoritmo DPA è stata realizzata la sola parte di ricerca, in modo da calcolare il costo dell'albero di Steiner ottimo e confrontarlo con i risultati delle euristiche; la codifica ha presentato essenzialmente il problema di rappresentare efficientemente la tabella S[], indicizzata da un insieme e un nodo, e di poterla enumerare in un certo ordine: per ottenere ciò, si sono utilizzate estensivamente le maschere di bit, in modo che S[d][v] fosse il costo dell'albero di Steiner per il nodo v e il sottoinsieme D di C, $C = Z - \{q\}$, individuato usando i p-1

bit del numero *d* come valori logici di appartenenza dei nodi *Z* all'insieme *D*. Dunque il primo indice è un numero nel range $1 \div 2^{p-1} - 1$ e si rappresenta con *p* bit. Con questa scelta, si è resa necessaria la preclassificazione dei numeri *d* secondo la cardinalità dell'insieme *D* che rappresentano, cioè secondo il numero di bit 1 presenti in *d*; d'altra parte, questo calcolo ha portato un overhead minimo alla procedura del DPA. Le successive operazioni insiemistiche richieste dall'algoritmo sono state implementate tramite operazioni logiche, secondo lo schema seguente (si veda anche il codice in appendice):

operazione	codifica in linguaggio C++
$S[\{t\}, j] \leftarrow d(t, j)$;	`S[1 << t][j] = d(z[t], j);`
i-esimo sottoinsieme *D* di card. *m*	`d = subset[m][i];  //precalcolato`
$D[1]$	`d1 = 1; while(!(d & d1)) d1 <<= 1;`
$D[1] \in E$ **and** $E \subset D$	`(e & d1) && e < d && (e | d) == d`
$S[D - E, j]$	`S[~e & d][j]`
$S[C - E, j]$	`S[~e & (1 << (p-1))-1][j]`

Per quanto riguarda le euristiche, la PDH è stata realizzata a solo scopo di testing iniziale dei livelli di base del sistema, essendo scadente in termini di competitività, e non è stata utilizzata nelle prove sperimentali successive. L'ADH, essendo la più complessa di tutte, ha richiesto una più accurata progettazione delle strutture dati e il più lungo debug: infatti, il calcolo della funzione che valuta la centralità di un nodo comporta la raccolta delle distanze del nodo dai sottoalberi; per consentire questa operazione, è stata attribuita ad ogni nodo un'etichetta di sottoalbero ed è stato definito un array di distanze indicizzato con questa etichetta; così basta un solo ciclo su tutti i nodi dotati dell'etichetta per cercare le suddette distanze; il successivo ordinamento e il calcolo della funzione di centralità – ancora un semplice ciclo – il tutto ripetuto per tutti i nodi della rete, completano un'iterazione dell'euristica, implementata nella funzione *TADHTree*::*getBestCNode*() (cfr. più avanti). La K-SPH è stata implementata quale caso particolare di ADH (cfr. Cap. 2), ereditandone le caratteristiche comuni di crescita di una foresta di sottoalberi, sebbene appartenga alla famiglia delle euristiche *PD-based*; pertanto, è stata definita per questa euristica la classe *TKADHTree*, il cui identificatore richiama questo legame, che specializza la succitata funzione *getBestCNode*() lasciando la sola ricerca dei due alberi più vicini.

Infine, per le euristiche ripetitive si è scelto di effettuare il post-processing al termine di ogni ripetizione, selezionando poi il migliore albero risultante; si ottiene così una competitività ancora migliore, a fronte di un certo incremento del costo computazionale, pari a $O(pn^2)$ per SPH-Z e $O(n^3)$ per SPH-V. D'altra parte, come già accennato, queste euristiche sono da considerare un termine di paragone per la competitività, e le prestazioni sono quindi secondarie.

A titolo esemplificativo e per meglio illustrare lo stile di programmazione, si riporta il codice dell'euristica SPH e di uno stralcio della ADH:

```
long TPDTree::calctree() {          // euristiche PD-based
   int n1, n2;
   costo = 0;
   for(int k = 0; k < nz-1; k++) {   // eseguito esattamente |Z|-1 volte
      getBestPath(n1, n2);           // sceglie un nodo n2 da unire secondo un criterio
      costo += join(n1, n2);         // unisce n2 a T (su n1) lungo lo shortest path
      }
   return costo;
}

void TPDTree::getBestPath(int& n1, int& n2) { ... }      // PDH

void TSPHTree::getBestPath(int& n1, int& n2) {
   long dcorr, dmin = INF;

   for(int v1 = 0; v1 < Net->n; v1++)
      if(attr[v1].tipo == S || attr[v1].tipo == Zin) // per tutti i nodi gia' in T
         for(int i2 = 0; z[i2] != -1; i2++)
            if(attr[z[i2]].tipo == Z && (dcorr = Net->getD(v1,z[i2])) < dmin)
               {
               dmin = dcorr;    // ...sceglie il nodo ancora non in T con la distanza min. da T
               n1 = v1; n2 = z[i2];
               };
}

// ------------------------------------------------------------

int TADHTree::getBestCNode(int& n1, int& n2) {  // restituisce v*; n1, n2 sono i nodi da unire
   int v, v2, i, j, nf, vmin;
   long d, fcorr, fmin = INF;

   for(v = 0; v < Net->n; v++) {
      // calcola le distanze da v ai frammenti correnti
      for(i = 0; i <= nz; i++) dv[i] = INF;
      for(v2 = 0; v2 < Net->n; v2++)
         if(attr[v2].fram != -1 && (d=Net->getD(v, v2)) < dv[attr[v2].fram]) {
            // v2 è un nodo di un sottoalbero e ha finora distanza min. da v => aggiorna
            dv[attr[v2].fram] = d;
            Tv[attr[v2].fram] = v2;
            };

      // ordina dv[f] (e Tv[f])
      ...

      // calcola f(v) secondo la formula:
      nf = 1;      //  nf = # framm. collegabili = r nella formula (qui r è la radice)
      if(dv[0] == 0)
         fcorr = dv[1];          // caso tipo K-SPH
      else {
         fcorr = dv[0];
         do { fcorr += dv[nf++]; }
         while(dv[nf] < INF && fcorr * nf > (fcorr + dv[nf]) * (nf - 1));
                                 // cioè:  ( fcorr / (nf-1) > ( fcorr + dv[nf]) / nf )
         fcorr /= nf - 1;
         };
      if(fcorr < fmin) {      // mette da parte la soluzione migliore
         fmin = fcorr;
         n1 = Tv[0];
         n2 = Tv[1];
```

```
            vmin = v;          // vmin = v*
            };
        };
    return vmin;
}
```

In esso si può notare che la classe *TPDTree* implementa il comportamento di base delle euristiche *greedy* basate sui cammini minimi, mentre la classe *TSPHTree* specializza la funzione virtuale *getBestPath*() introducendo la logica di ricerca propria dell'euristica SPH. Inoltre, è evidente il ciclo di calcolo in ADH delle distanze d_v da v verso i sottoalberi o frammenti e la successiva valutazione della funzione di centralità.

Per il codice completo si veda l'appendice.

3.3. Sviluppi futuri

Il sistema sviluppato è, come si è visto, espandibile in più direzioni; sono previste in particolare alcune evoluzioni, tra cui l'implementazione di altre euristiche e di tecniche di post-processing.

Inoltre, è possibile migliorare le prestazioni grazie a una coda a priorità più efficiente, che abbassa il costo dell'algoritmo di Dijkstra da $O(n^2)$ a $O(m \log n)$, con un risparmio significativo in quanto nelle reti esaminate $m << n^2$ (cfr. Cap. 4), oltre ad alcune migliorie specifiche per ADH, tra cui una routine di ordinamento più efficiente.

Infine, per un miglior interfacciamento con l'utente, è previsto lo sviluppo di un *front-end web-based*, che tra l'altro consentirà l'utilizzo anche in modo remoto, configurando il sistema come una piattaforma di test sia per la didattica che per la ricerca.

4. Prove sperimentali

In questo capitolo verranno analizzate le prestazioni del sistema implementato con riferimento sia alla competitività effettiva sia al tempo impiegato. In particolare, l'analisi della competitività viene svolta tramite un grafico di competitività cumulata, o *cumulative cost competitiveness* nella letteratura anglosassone. Per elaborare tale grafico vengono svolte numerose prove nelle modalità esposte nei prossimi paragrafi e vengono calcolate le competitività di tutte le euristiche per ogni caso; il grafico mostra, al variare dell'ascissa *c*, il numero di volte che ogni euristica ha riportato una competitività non maggiore di *c*. Queste curve tendono asintoticamente al numero di casi esaminati totale già per $c > 1{,}2$; d'altronde, come si è visto la competitività è sempre non peggiore di 2 per i metodi analizzati. In questo confronto, l'algoritmo ottimo produrrebbe una curva coincidente con questo asintoto orizzontale.

Inoltre, si è affiancato a questo grafico un istogramma che mostra la percentuale di successo di ogni euristica, definita come il numero di volte che l'euristica ha ottenuto la soluzione migliore, o quella ottima quando è nota, sul numero di prove totali; questo grafico corrisponde ai punti di ascissa 1 nel grafico di competitività cumulata.

Infine, è stato stimato il tempo di calcolo impiegato in media, oltre alla deviazione standard che in certi casi può risultare relativamente elevata a seconda delle caratteristiche delle reti.

4.1. Le reti simulate

Per ottenere una prova realistica, era necessario generare un elevato numero di reti simulate con una topologia nota e mediamente costante; allo scopo è stato utilizzato il software TIERS [Doar96], che è in grado di produrre topologie gerarchiche simili alle reti di computer reali in base a parametri quali numero di WAN totali, numero di MAN per ogni WAN ecc., ottenendo reti con grado massimo dei nodi e numero totale di link *m* estremamente bassi rispetto a topologie del tutto casuali. Gli insiemi di multicast sono stati scelti prendendo *p* nodi uniformemente distribuiti nella rete, con *p* pari al 5%, 10% e 20% di *n*, e sono stati predisposti dei gruppi di test a *p* ed *n* costanti.

Sono state quindi generate oltre 200 reti con 200 nodi e 10 nodi di multicast, 100 reti con 500 nodi e 25, 50 e 100 nodi *Z*, e infine 100 reti con 1000 nodi e 50, 100 e 200 nodi *Z*, per un totale di oltre 800 reti diverse. In particolare, l'interesse si è rivolto principalmente alle reti di grandi dimensioni (500 e 1000 nodi), in quanto più vicine a problemi reali, mentre sul gruppo di reti da 200 nodi si è potuto eseguire l'algoritmo ottimo DPA, impraticabile su dimensioni maggiori dell'insieme di multicast, che ha consentito di ottenere i grafici di competitività cumulata esatti. La tabella seguente mostra un riepilogo delle caratteristiche delle reti generate:

cardinalità della rete	grado massimo	grado di connettività[6]
200 nodi	13	1,3 %
500 nodi	15	0,5 %
1000 nodi	20 ÷ 25	0,3 %

4.2. Organizzazione delle prove e raccolta dei risultati

Per tutte le reti generate, è stato calcolato l'albero di Steiner secondo tutte le euristiche implementate; in seguito è stato applicato il post-processing *POST-MST*. Tutte le prove sono state eseguite su una workstation *Sun* UltraSPARC 250 MHz equipaggiata con *Sun Solaris* 2.6. Per automatizzare le prove, sono stati messi a punto alcuni script della shell che si occupano di eseguire il programma sulle diverse reti simulate, memorizzando i dati di output significativi in un unico file; di seguito è presentato un estratto del file di output, per una rete da 500 nodi:

```
...
net18
 sph: 974836 - 175 nodi; dopo MST: 973897 - 175 nodi; tempo: 643 msec.
sphz: 973577 - 175 nodi; dopo MST: 972638 - 175 nodi; tempo: 10696 msec.
sphv: 972638 - 175 nodi; tempo: 53194 msec.
 dnh: 973902 - 176 nodi; tempo: 312 msec.
ksph: 974836 - 175 nodi; dopo MST: 973897 - 175 nodi; tempo: 1097 msec.
 adh: 974831 - 174 nodi; dopo MST: 973892 - 174 nodi; tempo: 18563 msec.
...
```

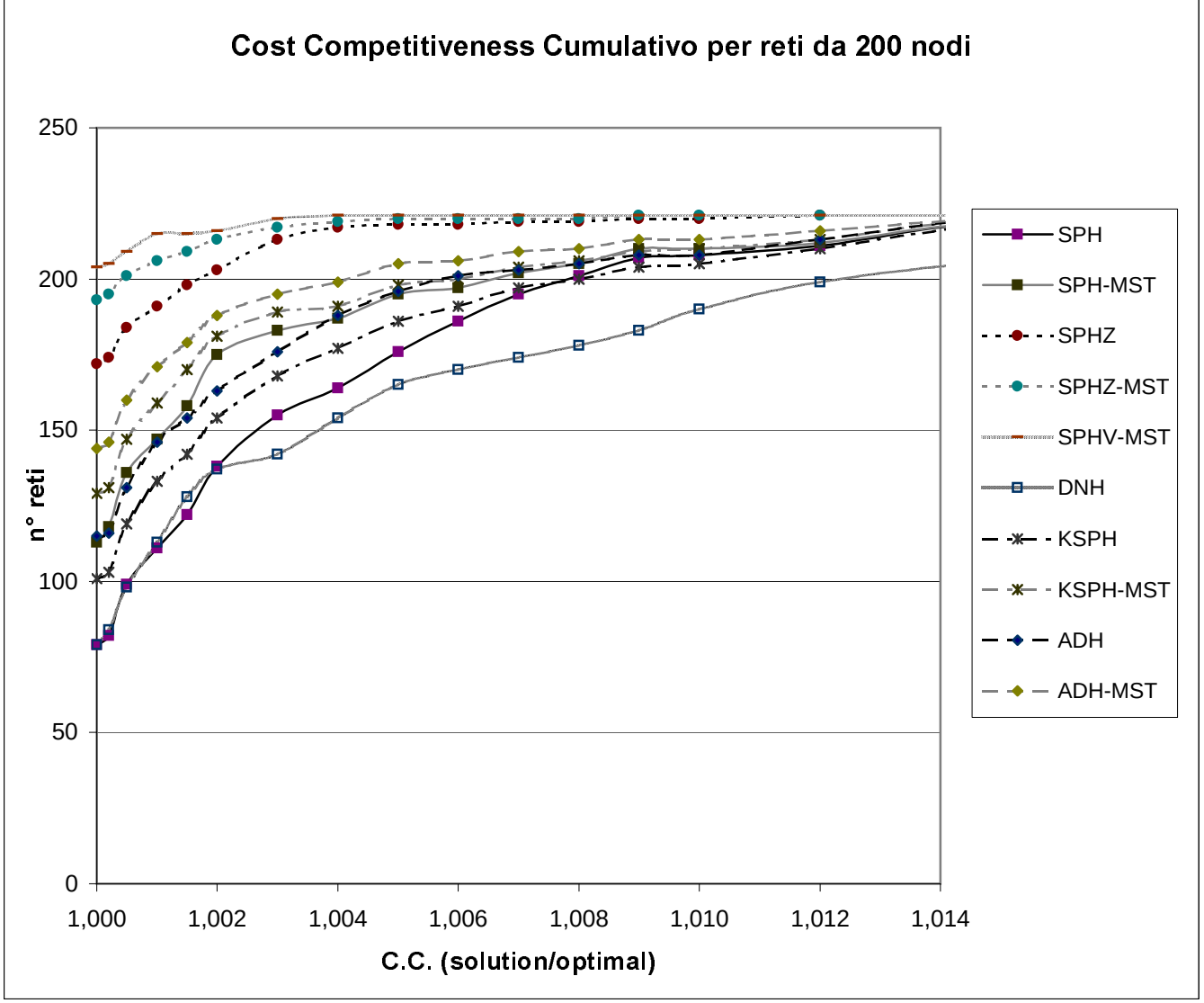
Come si può notare, per ogni euristica vengono restituiti il costo ed il numero di nodi dell'albero, seguiti dagli stessi dati per l'albero ottenuto dopo il post-processing MST; infine, viene restituito il tempo di calcolo dell'euristica escluso il post-processing, tenuto conto che questo risulta sempre trascurabile anche nei casi di dimensioni maggiori.

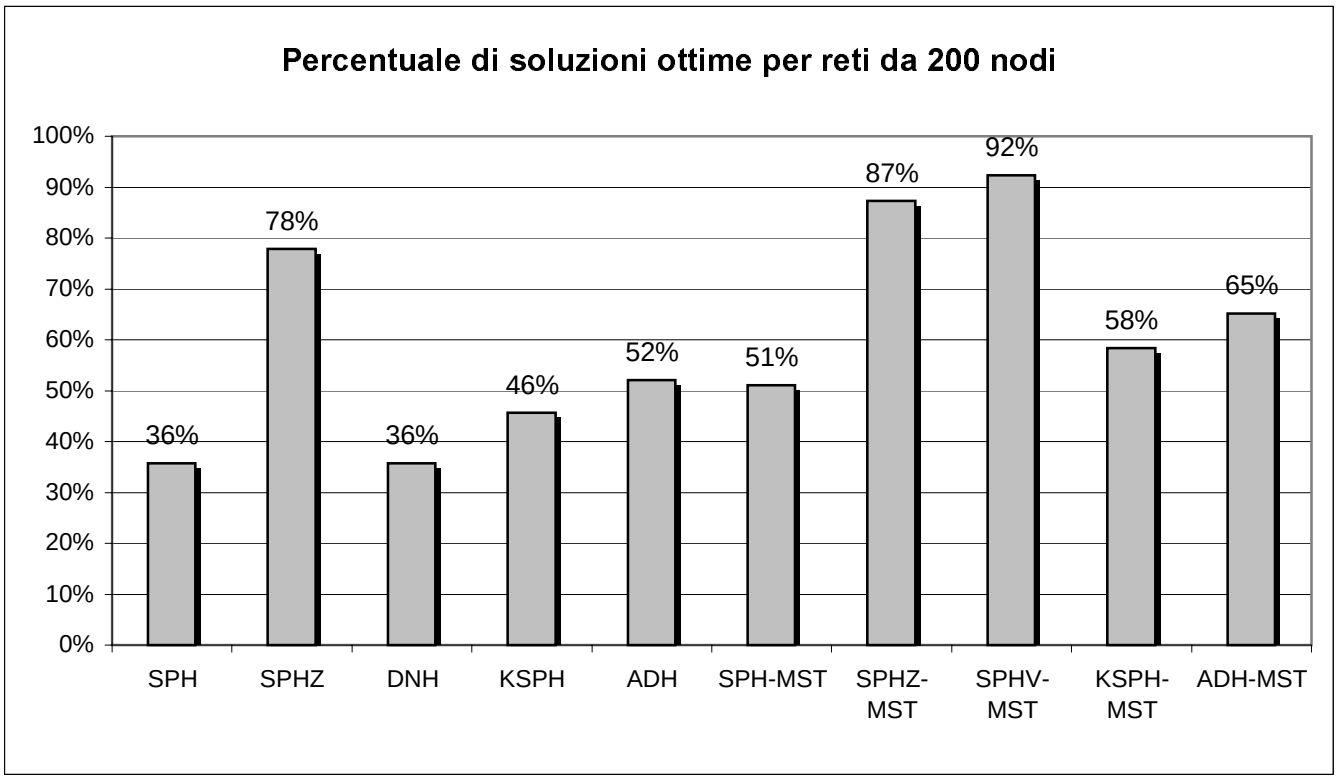
4.3. Risultati sperimentali

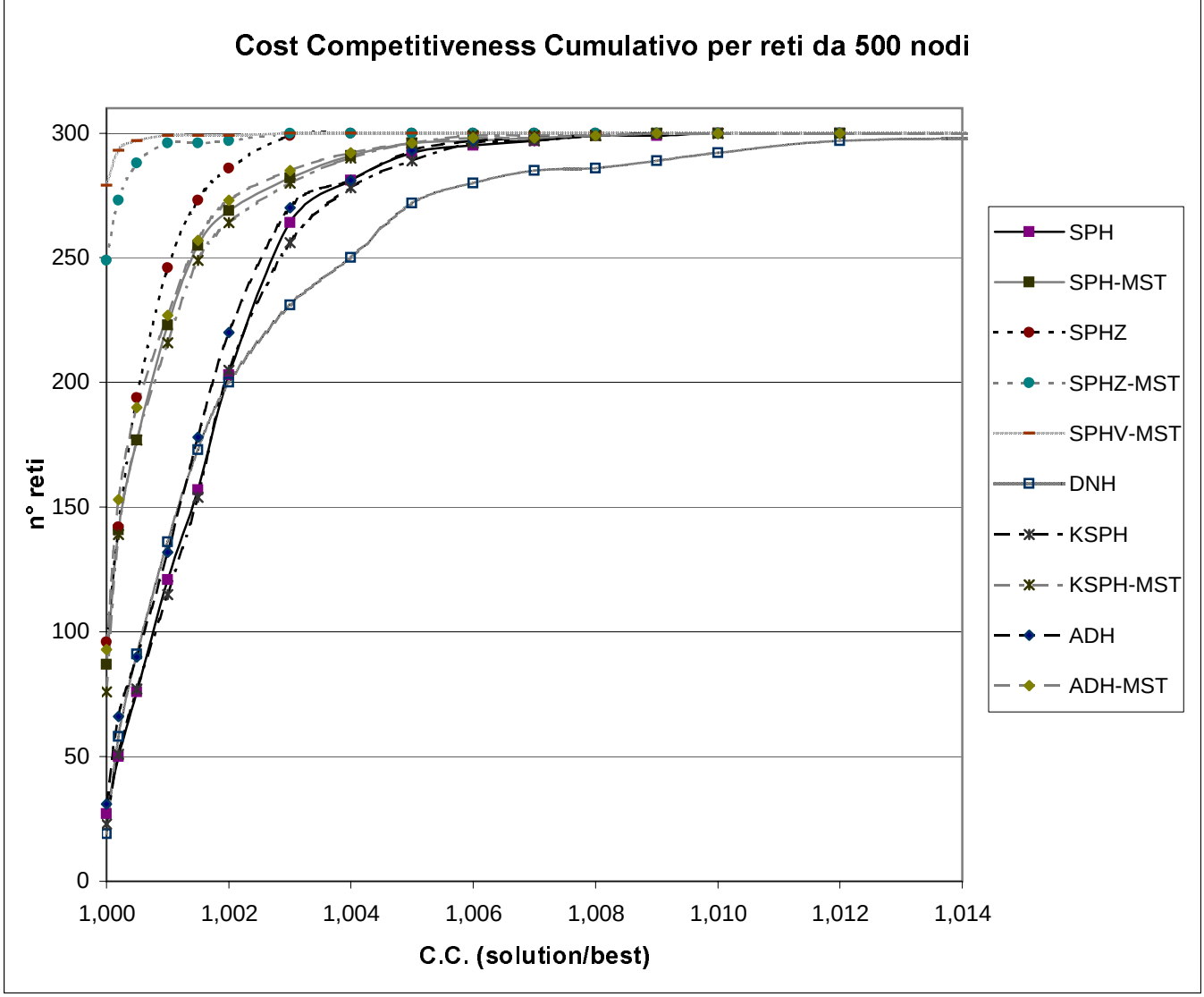
Vengono di seguito presentati i grafici di competitività cumulata e di percentuale di successo, dove vengono messe a confronto le varie euristiche sulle reti da 200, 500 e 1000 nodi. Per SPH-V vengono mostrati solo i risultati dopo *POSTMST* (cfr. par. 3.2), mentre per DNH non si è eseguito il post-processing in quanto incluso nel metodo stesso.

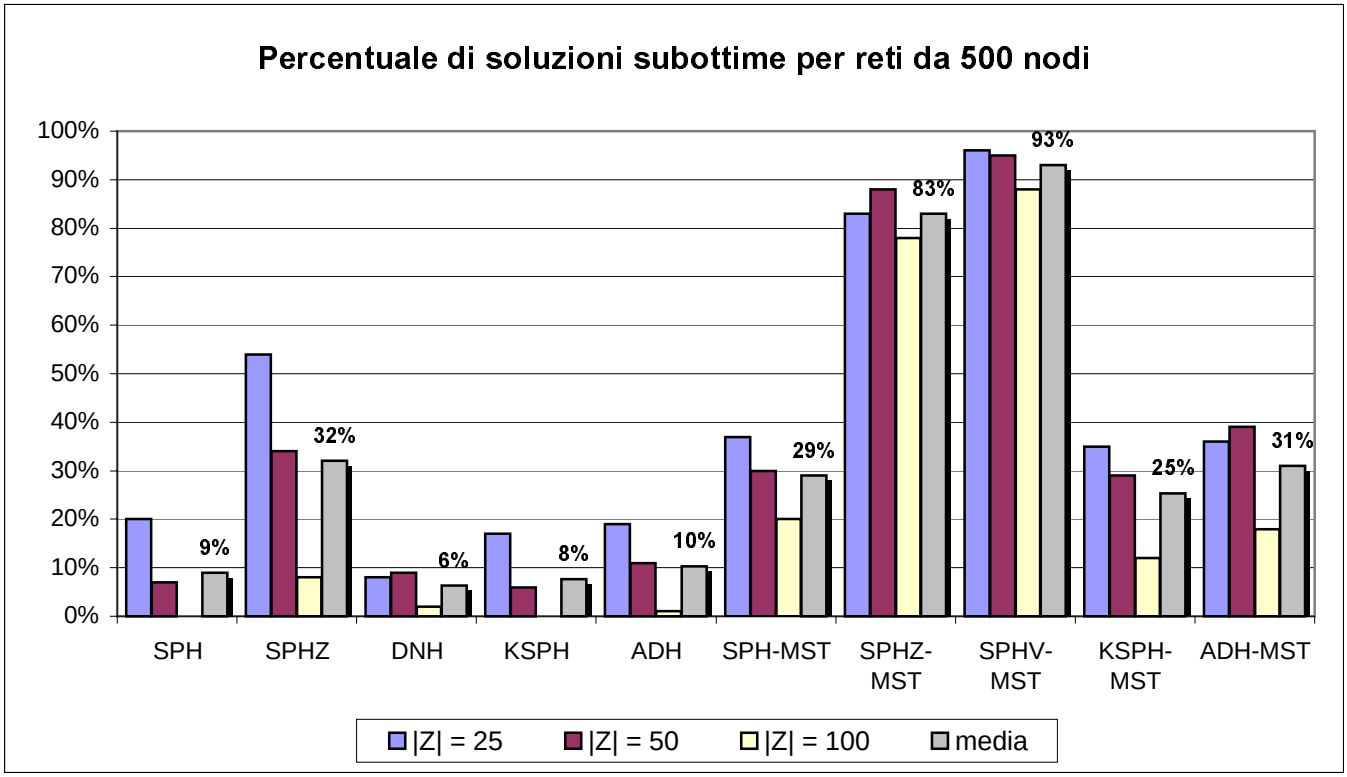
In definitiva, il confronto viene effettuato tra 10 metodi diversi e precisamente: DNH, SPH, SPH + MST, SPH-Z, SPH-Z + MST, SPH-V + MST, K-SPH, K-SPH + MST, ADH, ADH+MST.

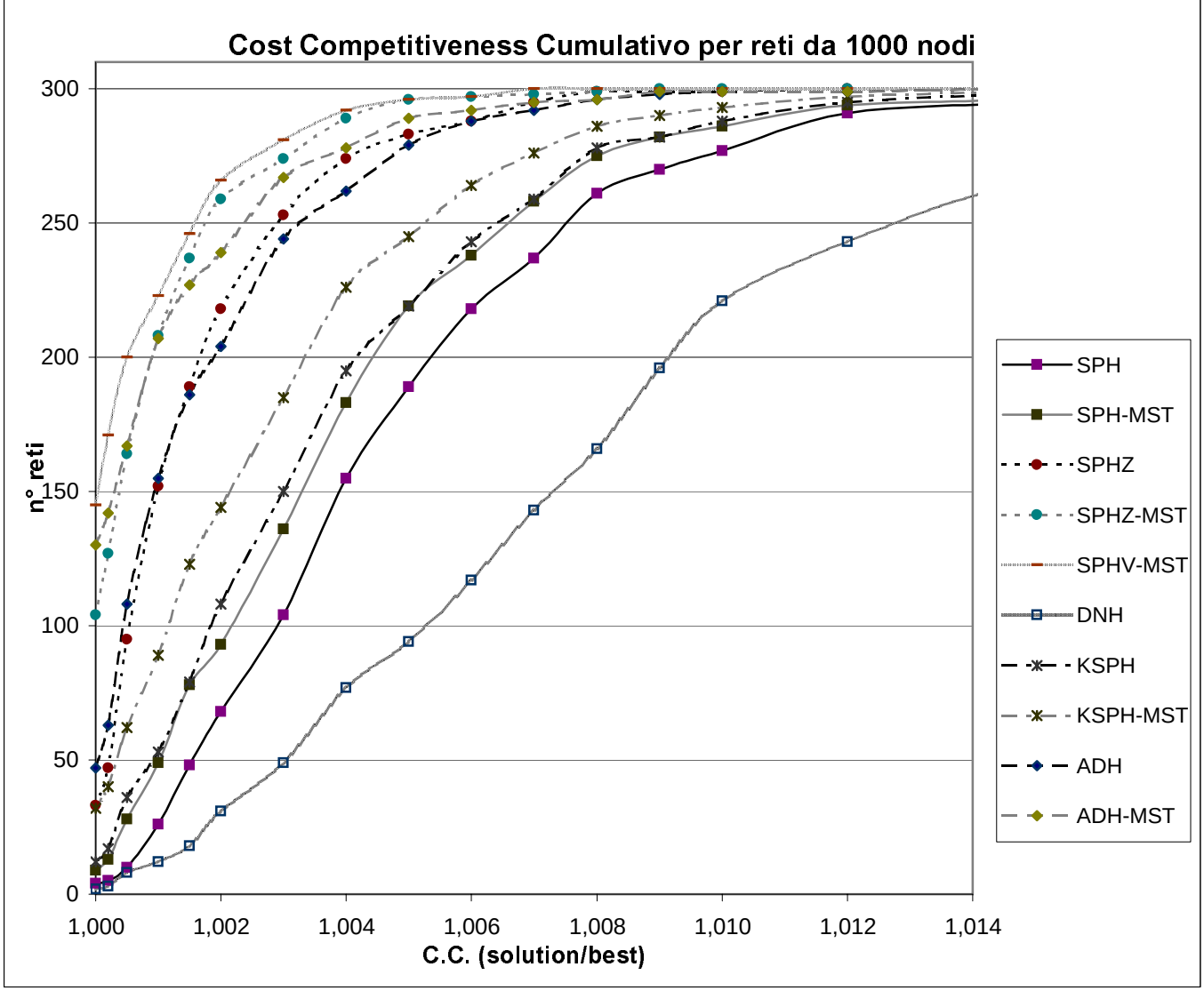
[6] Il *grado di connettività* di una rete è pari al numero di archi diviso il numero massimo di archi se la rete fosse completamente connessa, cioè $n(n-1)/2$ con n nodi.

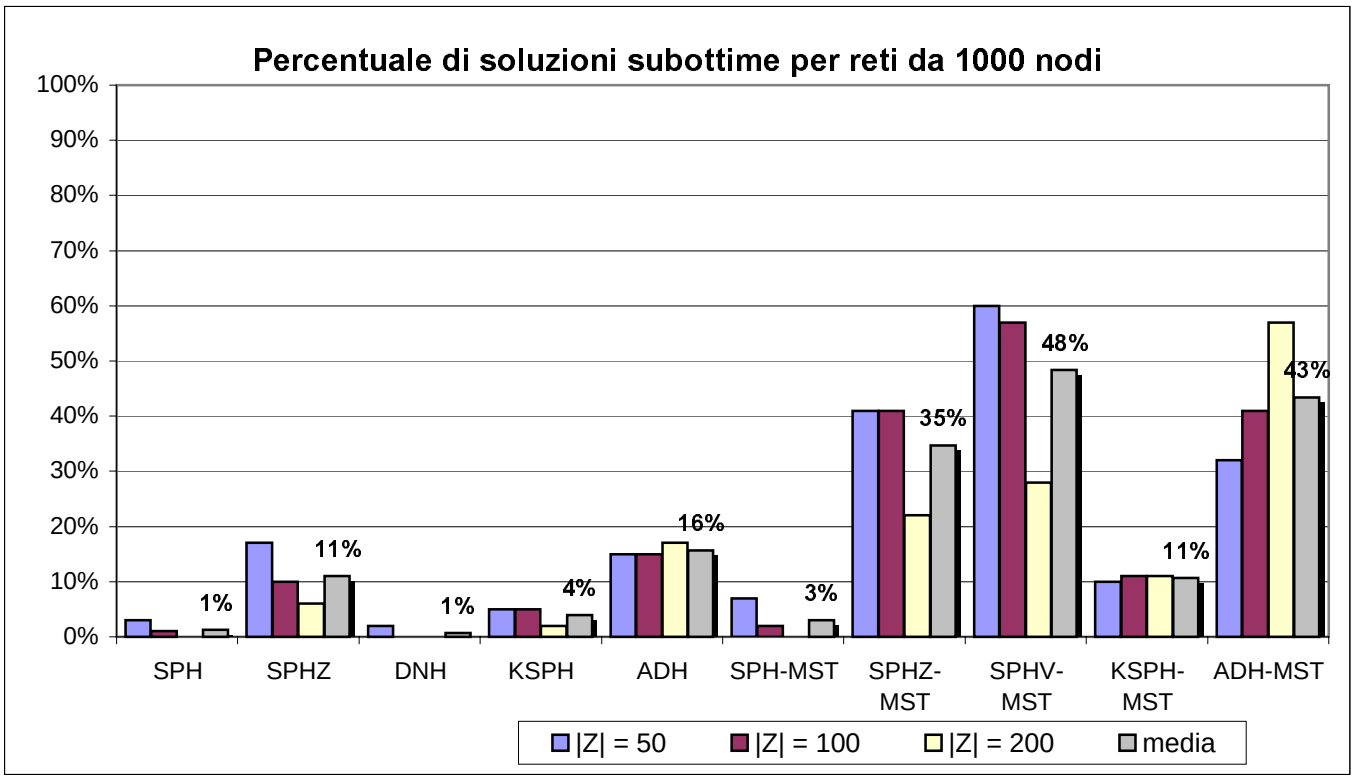

Cost Competitiveness Cumulativo per reti da 200 nodi
250
200
150
100
50
0
n° reti
1,000
1,002
1,004
1,006
1,008
1,010
1,012
1,014
C.C. (solution/optimal)
SPH
SPH-MST
SPHZ
SPHZ-MST
SPHV-MST
DNH
KSPH
KSPH-MST
ADH
ADH-MST

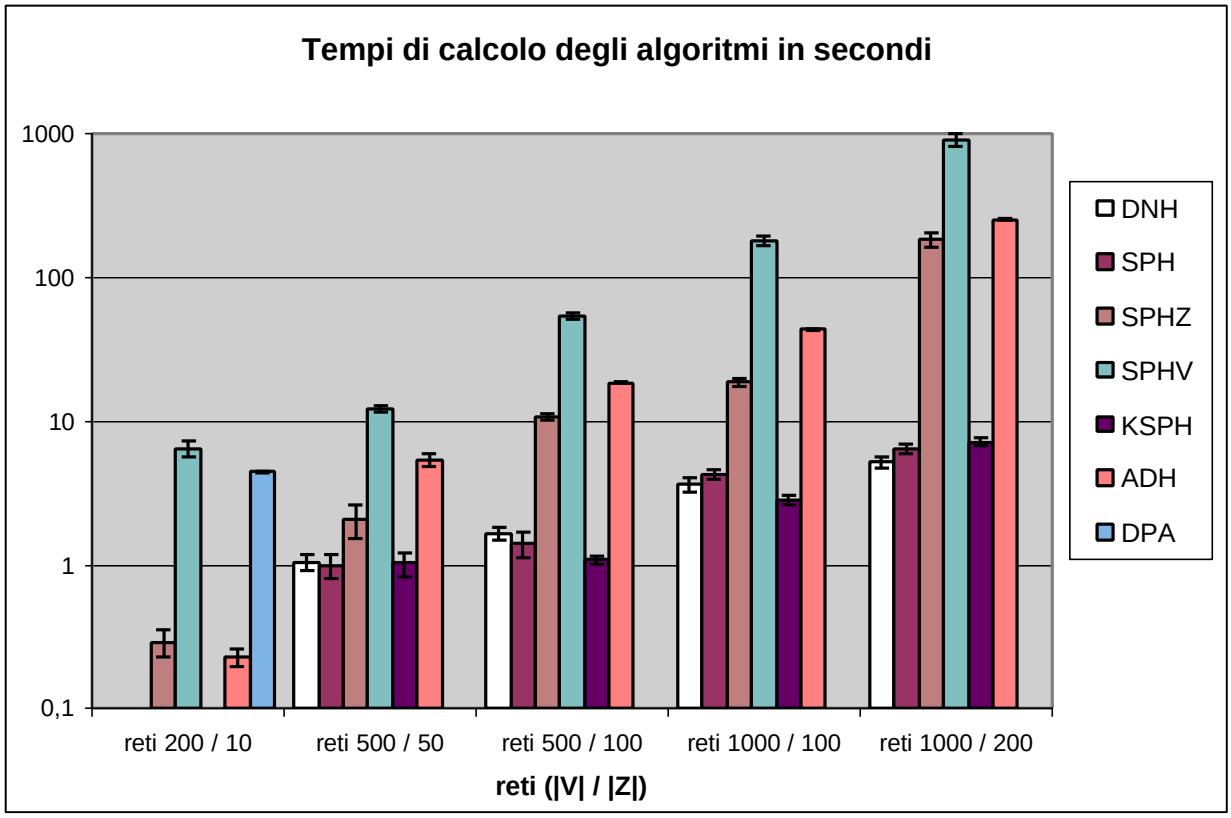

Percentuale di soluzioni ottime per reti da 200 nodi
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
36%
78%
36%
46%
52%
51%
87%
92%
58%
65%
SPH
SPHZ
DNH
KSPH
ADH
SPH-MST
SPHZ-MST
SPHV-MST
KSPH-MST
ADH-MST


Cost Competitiveness Cumulativo per reti da 500 nodi
n° reti
C.C. (solution/best)
0
50
100
150
200
250
300
1,000
1,002
1,004
1,006
1,008
1,010
1,012
1,014
SPH
SPH-MST
SPHZ
SPHZ-MST
SPHV-MST
DNH
KSPH
KSPH-MST
ADH
ADH-MST


Percentuale di soluzioni subottime per reti da 500 nodi
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
9%
32%
6%
8%
10%
29%
83%
93%
25%
31%
SPH
SPHZ
DNH
KSPH
ADH
SPH-MST
SPHZ-MST
SPHV-MST
KSPH-MST
ADH-MST
|Z| = 25
|Z| = 50
|Z| = 100
media


Cost Competitiveness Cumulativo per reti da 1000 nodi
n° reti
C.C. (solution/best)
0
50
100
150
200
250
300
1,000
1,002
1,004
1,006
1,008
1,010
1,012
1,014
SPH
SPH-MST
SPHZ
SPHZ-MST
SPHV-MST
DNH
KSPH
KSPH-MST
ADH
ADH-MST


Percentuale di soluzioni subottime per reti da 1000 nodi
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
1%
11%
1%
4%
16%
3%
35%
48%
11%
43%
SPH
SPHZ
DNH
KSPH
ADH
SPH-MST
SPHZ-MST
SPHV-MST
KSPH-MST
ADH-MST
|Z| = 50
|Z| = 100
|Z| = 200
media



Da tutti i grafici presentati è possibile trarre alcune conclusioni sul comportamento delle euristiche al crescere delle dimensioni del problema. In generale, tutte le euristiche si comportano effettivamente meglio rispetto ai limiti teorici di competitività, considerato che nella maggior parte dei casi questa risulta minore di 1,015, nonostante il basso grado di connettività delle reti, che comporta un compito per le euristiche più arduo. Il tempo di esecuzione è in linea con il costo computazionale teorico, anche se con deviazioni standard – indicate nel grafico – fino al 15% ÷ 20% dalla media in diversi casi, soprattutto per le reti più piccole, dove le euristiche più veloci impiegano qualche decina di millisecondi e non sono rappresentate; per confronto, il tempo di calcolo medio per DPA sulle reti da 200 nodi è risultato di 4,5 secondi con una deviazione standard solo dell'1,2%, poiché DPA esegue la sua procedura indipendentemente da caratteristiche del problema specifico, mentre le euristiche se ne possono avvantaggiare per condurre il calcolo più o meno velocemente. Si noti anche che con questa scelta di n e p la SPH-V ha impiegato mediamente più tempo del DPA e senza trovare il 100% delle soluzioni ottime; d'altronde il DPA eseguito già con $n = 200$ e $p = 15$ impiega circa 10 minuti e con $p = 17$ supera le due ore, tempi di gran lunga superiori a quelli delle euristiche.

Più in dettaglio, l'andamento della competitività cumulata mostra come ogni euristica abbia un suo comportamento specifico: la veloce DNH è in effetti anche la peggiore, ottenendo solo il 36% di soluzioni ottime nelle reti da 200 nodi e addirittura solo 2 soluzioni subottime nelle reti da 1000 nodi; in confronto SPH, che pure ha basse percentuali di successo, mostra una

curva di competitività migliore in tutti i casi e suscettibile di ulteriore miglioramento grazie al post-processing; questo di fatto è stato in grado di migliorare uniformemente e con un basso costo aggiuntivo la competitività di tutte le euristiche, specie quelle ripetitive: SPH-V + MST ottiene più del 90% di soluzioni ottime nelle reti da 200 nodi o subottime in quelle da 500 nodi. Questa euristica si è dunque confermata come soluzione di riferimento, almeno per reti da 200 e 500 nodi, dato che le curve di competitività cumulata sono quasi orizzontali, con pochissimo margine dalla soluzione ottima.

Il confronto è invece diverso e più interessante con le reti da 1000 nodi: qui infatti nessuna euristica sembra prevalere nettamente sulle altre, e tutte hanno curve di competitività notevolmente più basse, con ampi margini di miglioramento rispetto alla soluzione ottima. In particolare, le uniche euristiche significative risultano ancora quelle ripetitive, SPH-Z + MST e SPH-V + MST, e l'ADH + MST, in quanto le altre hanno curve più basse e percentuali di successo inferiori al 20%. Ma come si vede nel grafico delle percentuali di successo, le prestazioni di SPH-V + MST diminuiscono sensibilmente all'aumentare di *p*, mentre salgono quelle di ADH + MST, che ottiene, per $p = 200$, il 57% di soluzioni subottime contro il 28% di SPH-V + MST; questo comportamento indica, come già accennato, la debolezza del metodo SPH, pur ripetuto, rispetto al più sofisticato metodo ADH, soprattutto nei casi più complessi. Questo trend verrà ulteriormente approfondito in futuro con prove su reti da 2000 nodi e 200 o 300 nodi di multicast.

Infine, è opportuno sottolineare che i migliori risultati sono stati ottenuti in ogni caso con le euristiche più costose in termini di risorse di calcolo (fino a 4 minuti per ADH e fino a 15 minuti per SPH-V, contro un massimo di 5 secondi per DNH), pertanto da questo confronto non emerge nessuna euristica da preferire *tout-court*, ma – come è tipico dei problemi ingegneristici – si tratta di scegliere il miglior compromesso tra prestazioni e costi computazionali in funzione dei requisiti specifici del problema in esame. In questo scenario può risultare interessante lo sviluppo di tecniche di post-processing iterative, che si possono comportare in modalità *just in time*, soddisfacendo così una più ampia gamma di requisiti.

5. Altre applicazioni del problema di Steiner

Il problema di Steiner, come molti problemi NP-completi, ricorre spesso in diversi settori delle scienze e dell'ingegneria. In questo capitolo vengono esaminate, a titolo esemplificativo, due istanze del problema relative all'ingegneria civile e alla filogenesi.

Si consideri la posa in opera, in un centro urbano o in un'area più vasta, delle varie reti di servizi, quali la rete idrica, la rete del gas o le reti di telecomunicazioni: è naturale cercare di minimizzare la lunghezza delle canalizzazioni da costruire, mantenendo comunque collegate tutte le utenze richieste, essendo il costo del lavoro proporzionale in prima approssimazione a tale lunghezza. Ancora, si consideri la realizzazione di un circuito in tecnologia VLSI, dove tra gli altri obiettivi vi è quello di minimizzare la lunghezza delle piste per diminuire il ritardo di propagazione dei segnali. Ciascuno di questi problemi può essere espresso in termini di problemi di Steiner euclidei o rettilinei, dato che tutte le lunghezze considerate sono distanze euclidee, o Manhattan nel caso VLSI, caso quest'ultimo particolarmente studiato grazie agli enormi investimenti in questa tecnologia.

D'altra parte, può essere utile garantire una certa ridondanza alla rete da costruire, in modo che siano tollerabili fino ad un certo numero k di collegamenti interrotti per una condizione di guasto o manutenzione senza che l'intera rete perda la sua funzionalità. In questo caso, si può riformulare il problema di Steiner in una versione generalizzata [Win87] che prevede una *k-ridondanza* del grafo soluzione G_S opportunamente definita.

Infine, nell'ambito dell'analisi filogenetica, che si occupa di stabilire le relazioni evoluzionistiche presenti tra le diverse specie viventi o estinte, assume un ruolo centrale la costruzione dell'albero filogenetico, le cui foglie rappresentano le singole specie viventi, e i cui punti di ramificazione corrispondono a specie – tipicamente estinte – dalle quali si è avuta una differenziazione di almeno un carattere. Per la costruzione di un tale albero, si assume che a ogni specie si possa associare un nodo dotato di diversi attributi (i suoi caratteri, il patrimonio genetico, ecc.), e che la distanza tra i nodi sia proporzionale alle differenze tra i caratteri delle specie corrispondenti; si ottiene così una rete di specie, del tutto analoga alle reti fin qui analizzate. Inoltre, si assume un criterio di *parsimonia* nella identificazione dei legami tra le diverse specie, ipotizzando che il percorso evolutivo più breve che connette un antenato ad uno dei suoi discendenti sia anche il più probabile. Ecco allora che, con le ipotesi fatte, l'albero filogenetico coincide esattamente con l'albero di Steiner calcolato sulla rete suddetta.

6. Conclusioni

Questo lavoro ha illustrato ed analizzato il problema di Steiner nelle reti, sia dal punto di vista teorico, sia tramite prove sperimentali. È stato analizzato in particolare un algoritmo ottimo di programmazione dinamica e una serie di euristiche diverse, che sono state poi confrontate sul piano delle prestazioni effettive con prove estensive su un grande numero di reti simulate. Sono stati affrontati quindi tutti i problemi specifici del caso per una corretta implementazione di tali prove, ottenendo al termine risultati pienamente soddisfacenti.

Come già detto, è previsto un ulteriore sviluppo della ricerca sia per quanto riguarda l'analisi di nuovi metodi euristici, sia per l'estensione delle prove sperimentali a casi più complessi, nell'auspicio che chiariscano ancor meglio alcuni interrogativi rimasti aperti.

Per concludere, vorrei riportare un contributo[7] per la soluzione *ottima* del problema fin qui esaminato nella sua versione sul piano euclideo:

«Un metodo interessante per la ricerca dell'albero minimo di Steiner coinvolge soltanto dell'acqua, del sapone, due lastre di vetro ed alcuni dischetti. Usando le due lastre di vetro come un sandwich, posizioniamo i dischi tra i due piani in modo che corrispondano alla posizione dei nodi da connettere [i nodi *Z* di multicast]. Faremo in modo che le lastre e i dischetti siano ben fissati assieme in modo che non cadano; lascerò questo all'immaginazione del lettore. Poi immergeremo tutto quanto in una soluzione di acqua e sapone in modo che si formi uno strato di sapone tra le due lastre. Poiché il sapone "vuole" minimizzare la sua area, il film andrà a disporsi in una forma, che rappresenta l'albero minimo di Steiner. Sebbene questa tecnica è poco pratica per problemi complessi, dimostra come la natura e le leggi della fisica possano cooperare insieme per risolvere problemi interessanti».

[7] C. ISENBERG, *The science of soap films and soap bubbles*, Tiero, Avon (U.K.), in: G. W. FLAKE, *The Computational Beauty of Nature*, MIT Press, 1999, p. 322.

Appendice. Codici sorgenti

- OMISSIS -

Bibliografia

- [DW72] S. E. DREYFUS, R. A. WAGNER, *The Steiner problem in graphs*, Networks, **1**, 1972, pp. 195-207.
- [TM80] H. TAKAHASHI, A MATSUYAMA, *An approximate solution for the Steiner problem in graphs*, Math. Japan, **24**, 1980, pp. 573-577.
- [Ray83] V. J. RAYWARD-SMITH, *The computation of nearly minimal Steiner trees in graphs*, Int. Math. Ed. Sci. Tech., **14**, 1983, pp. 15-23.
- [Win87] P. WINTER, *Steiner problem in networks: a survey*, Networks, **17**, 1987, pp. 129-167.
- [WI88] B. M. WAXMAN, M. IMASE, *Worst-case performance of Rayward-Smith's Steiner tree heuristics*, Inform. Proc. Lett., **29**, 1988, pp. 283-287.
- [Win92] P. WINTER, J. MACGREGOR SMITH, *Path-distance heuristics for the Steiner problem in undirected networks*, Algorithmica, **7**, 1992, pp. 309-327.
- [Voss92] S. VOSS, *Steiner's problem in graphs: heuristic methods*, Discrete Appl. Math., **40**, 1992, pp. 45-72.
- [KPP93a] V. P. KOMPELLA, J. C. PASQUALE, G. C. POLYZOS, *Multicast routing for multimedia communication*, IEEE/ACM Trans. on Netw., **1**, 3, Giugno 1993, pp. 286-292.
- [KPP93b] V. P. KOMPELLA, J. C. PASQUALE, G. C. POLYZOS, *The multimedia multicasting problem*, Tech. Rep. CS93-313, Settembre 1993.
- [ZPG95] Q. ZHU, M. PARSA, J. J. GARCIA-LUNA-ACEVES, *A source-based algorithm for delay constrained minimum-cost multicasting*, IEEE Proc. Infocom., 1995, pp. 377-385.
- [RN95] S. J. RUSSELL, P. NORVIG, *Artificial Intelligence: a modern approach*, Prentice Hall, 1995.
- [Doar96] M. B. DOAR, *A better model for generating test networks*, ftp.nexen.com/pub/papers, 1996.
- [Bau96] F. BAUER, *Multicast routing in point-to-point networks under constraints*, Ph.D. Dissertation, Univ. of California Santa Cruz, 1996.
- [Bau97] F. BAUER, A. VARMA, *ARIES: a rerrangeable inexpensive edge-based on-line Steiner algorithm*, IEEE J. on Sel. Areas in Comm., **15**, 3, Aprile 1997, pp. 382-397.
- [CLR97] T. H. CORMEN, C. H. LEISERSON, R. L. RIVEST, *Introduction to Algorithms*, MIT Press, Boston, 1997.
- [RMM99] S. RAGHAVAN, G. MANIRAMAN, C. S. R. MURTHY, *A rearrangeable algorithm for the construction of delay-constrained dynamic multicast trees*, IEEE/ACM Trans. on Netw., **7**, 4, Agosto 1999, pp. 514-528.

*****