Cammini minimi per grafi molto grandi

1. Definizioni

È dato un grafo orientato $G = (N, E)$. Indichiamo con n il numero di nodi N e con m il numero di archi E. Se il problema del cammino minimo è formulato su un grafo non orientato, si può sempre trasformare il grafo in uno orientato sostituendo l'arco non orientato (i, j) con la coppia antiparallela (i, j) e (j, i).

Per ogni arco $(i, j) \in E$ è definita una *lunghezza* $d_{ij} \geq 0$. Due nodi sono speciali e vengono detti *sorgente* e *destinazione*, denotati rispettivamente come s e t. La *lunghezza di un cammino* è la somma delle lunghezze degli archi del cammino. Il *cammino minimo* fra due nodi i e j è il cammino (non necessariamente unico) di lunghezza minima fra i nodi dati. La *distanza* fra il nodo i e il nodo j è la lunghezza del cammino minimo $i \to j$.

Il problema consiste nel trovare il cammino minimo dalla sorgente alla destinazione. Si indichi con δ la distanza da s a t, con δ_i la distanza da s ad i, con δ^i la distanza da i ad t e con δ^i_j la distanza fra i nodi generici i e j (nell'ordine, in generale $\delta^i_j \neq \delta^j_i$).

In base alla proprietà di cammino minimo $\delta^i_j \leq \delta^i_k + \delta^k_j$ per ogni $i, j, k \in N$ e in particolare $\delta \leq \delta_i + \delta^i$ per ogni $i \in N$. Più in dettaglio

$$\begin{aligned} \delta &= \delta_i + \delta^i \qquad \text{se } i \text{ è su un cammino minimo,} \\ \delta &< \delta_i + \delta^i \qquad \text{se } i \text{ non è su un cammino minimo} \end{aligned} \tag{1}$$

2. Complessità dell'algoritmo di Dijkstra

L'algoritmo di Dijkstra ha complessità $O(n^2)$ con un'implementazione semplice, $O((n + m) \log n) = O(m \log n)$ usando gli heap binari e $O(m + n \log n)$ usando gli heap di Fibonacci. L'implementazione con gli heap binari risulta conveniente in grafi planari dove il numero di archi è lineare nel numero di nodi (più esattamente $m \leq 3\,n - 6$) e quindi la complessità si abbassa a $O(n \log n)$. Per grafi molto grandi, con più di 100.000 nodi, il numero di operazioni da eseguire è comunque molto elevato. Ad esempio, con 100.000 nodi e 300.000 archi, il numero di operazioni è $(100.000 + 300.000) \log_2 100.000 \approx 6.643.860$, la cui esecuzione può richiedere molti secondi.

Tuttavia si tratta di valutazioni di caso peggiore, mentre l'algoritmo di Dijkstra si comporta mediamente in modo molto più efficiente. L'aspetto più interessante è che l'algoritmo si interrompe non appena visita il nodo destinazione.

Ricordiamo gli aspetti salienti dell'algoritmo di Dijkstra. Ad ogni iterazione vi è un insieme di nodi la cui distanza dalla sorgente è nota e definitiva. Indichiamo questi nodi come *visitati.* Inizialmente la sorgente è l'unico nodo visitato (nella ricerca in avanti, mentre è la destinazione nella ricerca all'indietro). I nodi adiacenti a quelli visitati ma non ancora visitati vengono detti *raggiunti* e per loro è disponibile una valutazione provvisoria della distanza. Gli altri nodi sono a distanza ignota. Ad ogni iterazione un nodo raggiunto (quello a minima distanza provvisoria) viene inserito fra i nodi visitati e nodi adiacenti a questo

nodo e non ancora visitati possono essere raggiunti per la prima volta oppure la loro distanza provvisoria può essere aggiornata. L'algoritmo termina quando viene visitata la destinazione.

Se la destinazione non è molto distante dalla sorgente, solo poche operazioni possono bastare per visitare la destinazione. Inoltre lo heap contiene soltanto i nodi finora raggiunti, che sono una frazione piccola di tutti i nodi. Si tenga allora conto anche della distanza δ fra sorgente e destinazione per valutare meglio il numero di operazioni. Sia $n(d)$ il numero di nodi a distanza minore o uguale a d. Allora la complessità si può esprimere come

$$O(n(\delta)\, \log n(\delta))$$

Tipicamente, per un grafo planare, $n(d)$ è una funzione quadratica in d. Grossolanamente si immagini l'insieme di nodi a distanza al più d come un cerchio di raggio d centrato sulla sorgente.

Una misura dell'*efficienza* dell'algoritmo, applicato ad una particolare istanza, è data dal rapporto fra il numero di nodi del cammino minimo e il numero di nodi visitati (non vengono inclusi i nodi raggiunti ma non ancora visitati). Un'efficienza uguale ad 1 vorrebbe dire che vengono visitati solo i nodi del cammino minimo, uno dopo l'altro fino a raggiungere la destinazione.

3. Ricerca bidirezionale

Una prima idea per ridurre il numero di operazioni consiste nel far partire contemporaneamente due cammini minimi, uno dalla sorgente verso la destinazione e l'altro dalla destinazione verso la sorgente. Ad un certo punto i due cammini si incontreranno mediamente a distanza $\delta/2$ e quindi il numero di nodi che ciascuna ricerca ha dovuto visitare è $1/4$ (nell'ipotesi di funzione $n(d)$ quadratica) di quelli che la ricerca solo dalla sorgente avrebbe fatto. In totale quindi solo la metà dei nodi deve essere esplorata e l'efficienza è più o meno raddoppiata.

L'algoritmo termina quando un medesimo nodo è visitato da entrambe le ricerche. Denotiamo con r questo nodo. Sia S_s l'insieme di nodi visitato da s e S^t quello visitato da t. Per definizione $S_s \cap S^t = \{r\}$. Per ogni nodo $i \in S_s$ è nota la distanza minima δ_i e per ogni nodo $i \in S^t$ è nota la distanza minima δ^i.

Dimostriamo che il cammino minimo $s \to t$ deve essere fra i cammini $s \to i \to j \to t$ dove $i \in S_s$, $j \in S^t$ ed esiste l'arco (i, j) oppure $s \to r \to t$. I cammini $s \to i \to j \to t$ hanno lunghezza $\delta_i + d_{ij} + \delta^j$ mentre il cammino $s \to r \to t$ ha lunghezza $\delta_r + \delta^r$. Quindi il migliore fra questi cammini ha lunghezza minore o uguale a $\delta_r + \delta^r$. Ogni altro cammino deve usare un nodo $k \notin S_s \cup S^t$. In base all'algoritmo di Dijkstra si ha $\delta_k \geq \delta_r$ e $\delta^k \geq \delta^r$ e quindi $\delta_k + \delta^k \geq \delta_r + \delta^r$. Quindi il cammino minimo è $s \to r \to t$ oppure uno dei cammini $s \to i \to j \to t$. In realtà il cammino $s \to r \to t$ è necessariamente già stato valutato come uno dei cammini $s \to i \to j \to t$ in un passo precedente dell'algoritmo.

Quindi l'algoritmo, quando viene rilevato un arco (i, j) fra i nodi visitati rispettivamente da s e da t, valuta la lunghezza del cammino e la confronta con quelle già calcolate.

Esempio. Si consideri un grafo a griglia ($50 \times 50 = 2500$ nodi e 4900 archi con lunghezze generate a caso uniformemente in $\{1, \ldots, 5\}$). Il nodo in rosso è la sorgente e quello in blu la destinazione. In Figura 1(a) si vede l'albero dei nodi esplorati nella ricerca del cammino minimo a partire dalla sorgente e in Figura 1(b) l'albero a partire dalla destinazione.

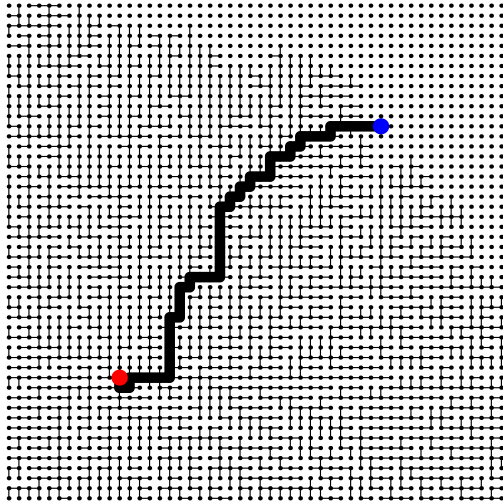


Figura 1(a)

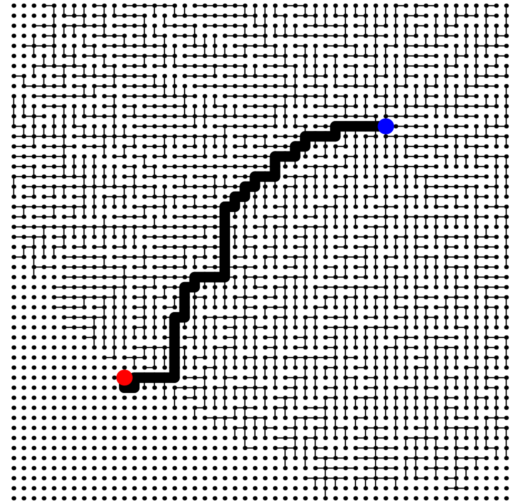


Figura 1(a)

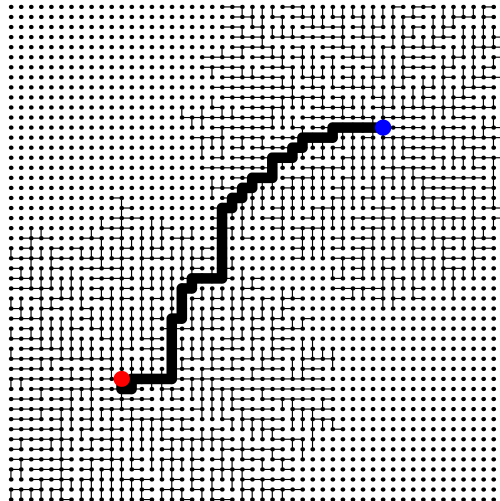


Figura 2

Il cammino minimo ha 54 nodi e vengono visitati 2107 nodi nella ricerca in avanti e 2100 nella ricerca all'indietro, per un'efficienza rispettivamente di 0.0256289 e 0.0257143. Come si vede viene esplorata in entrambi i casi la quasi totalità dei nodi (84%) e quindi la possibilità di interrompere l'algoritmo non appena si raggiunge l'altro nodo non genera un vantaggio apprezzabile. Usando la ricerca bidirezionale i due alberi alla terminazione, quando hanno in comune esattamente un nodo, sono come in Figura 2. In questo caso sono stati visitati 1638 nodi (il 65.5% del totale) e l'efficienza è 0.032967.

4. Uso di potenziali nei nodi

Guardando i nodi esplorati dall'algoritmo (Figura 2), anche con la ricerca bidirezionale, si vede che l'algoritmo 'perde tempo' ad esplorare nodi che sono troppo distanti dal nodo sorgente o destinazione per risultare utili. Naturalmente questa osservazione dipende dalla nostra conoscenza visiva del grafo mentre l'algoritmo non ha alcuna informazione sulle distanze dei nodi ancora da esplorare. Quindi l'idea per evitare queste perdite di tempo potrebbe essere quella di fornire tale informazione all'algoritmo. In particolare l'idea è di poter disporre in ogni nodo di un valore che sia una limitazione inferiore alla distanza dal nodo alla destinazione e di orientare la scelta dei nodi da esplorare in base, non solo alla distanza dalla sorgente al nodo ma anche alla limitazione inferiore dal nodo alla destinazione.

A questo scopo, operando l'algoritmo di Dijkstra in avanti (dalla sorgente s) supponiamo di modificare le lunghezze degli archi nel seguente modo

$$d'_{ij} := d_{ij} - \pi^i + \pi^j \tag{2}$$

dove π^i sono valori arbitrari assegnati ai nodi, detti *potenziali*. Si noti che i potenziali sono definiti a meno di una costante additiva e quindi il potenziale in un nodo opportuno può essere assegnato arbitrariamente. Dalla formulazione della programmazione dinamica all'indietro come programmazione lineare

$$\begin{aligned} \max\ & \pi^s - \pi^t \\ & \pi^i - \pi^j \le d_{ij} \qquad (i,j) \in E \end{aligned} \tag{3}$$

si vede che i valori d'_{ij} sono non negativi se e solo se i valori π^i sono ammissibili in (3). Inoltre valori π^i ammissibili in (3) con $\pi^t \le 0$ costituiscono limitazioni inferiori a δ^i. Infatti se si sommano le diseguaglianze $\pi^i - \pi^j \le d_{ij}$ lungo un cammino minimo $i \to t$, si ottiene $\pi^i - \pi^t \le \delta^i$, cioè $\pi^i \le \delta^i + \pi^t \le \delta^i$ (da $\pi^t \le 0$).

La lunghezza di un qualsiasi cammino $P : s \to t$ con le nuove distanze è

$$d'(P) = \sum_{(ij)\in P} d'_{ij} = \sum_{(ij)\in P} (d_{ij} - \pi^i + \pi^j) = d(P) + \pi^t - \pi^s$$

Tutte le lunghezze dei cammini $s \to t$ sono quindi alterate del fattore costante $\pi^t - \pi^s$ e il cammino minimo $s \to t$ è il medesimo per qualsiasi scelta dei potenziali (inclusi potenziali nulli, cioè le lunghezze originali), da cui

$$\delta' = \delta + \pi^t - \pi^s$$

Quindi il cammino minimo si può calcolare con una scelta arbitraria dei potenziali purché le lunghezze d' siano non negative, in modo da poter sempre adoperare l'algoritmo di Dijkstra. Analogamente la lunghezza di un cammino P_k da s ad un nodo generico k vale, con le distanze d'

$$d'(P_k) = \sum_{(ij)\in P_k} d'_{ij} = \sum_{(ij)\in P_k} d_{ij} - \pi^i + \pi^j = d(P_k) + \pi^k - \pi^s \tag{4}$$

e quindi anche per i cammini minimi da s a k si ha

$$\delta'_k = \delta_k + \pi^k - \pi^s \tag{5}$$

Allora l'albero dei cammini minimi da s è il medesimo per qualsiasi scelta dei potenziali. Ciò che cambia sono le distanze sull'albero ed è questo fatto che permette di accelerare l'algoritmo con un'opportuna scelta dei potenziali. Infatti l'algoritmo (nella ricerca in avanti) seleziona i nodi in ordine di distanza minima δ'_k secondo le lunghezze d'_{ij}, ovvero, da (5), secondo i valori δ_k del cammino minimo $s \to k$ con lunghezze originali più il valore π^k, stima per difetto di un cammino da k a t (il valore costante π^s è irrilevante in quanto indipendente da k).

Ad esempio, se fosse $\pi^k = \delta^k$, cioè proprio la distanza minima da k a t anziché semplicemente una stima per difetto, per tutti i nodi sul cammino minimo $s \to t$ si avrebbe $\delta'_k = 0$ (da $\pi^s = \delta$ e da (1)), mentre sui nodi non sul cammino minimo necessariamente $\delta'_k > 0$ (sempre da $\pi^s = \delta$ e da (1)). Questo implica che l'algoritmo sceglierà subito tutti e soli i nodi del cammino minimo.

In modo del tutto simmetrico se si opera l'algoritmo di Dijkstra a partire dalla destinazione si può pensare di ridefinire le distanze secondo

$$d''_{ij} := d_{ij} - \pi_j + \pi_i$$

per potenziali arbitrari π_i (si noti la differenza con (2)). Dalla formulazione della programmazione dinamica in avanti come programmazione lineare

$$\begin{aligned} \max\ & \pi_t - \pi_s \\ & \pi_j - \pi_i \leq d_{ij} \qquad (i,j) \in E \end{aligned} \tag{6}$$

si vede che i valori d''_{ij} sono non negativi se e solo se i valori π_i sono ammissibili in (6). Inoltre valori π_i ammissibili in (6) con $\pi_s \leq 0$ costituiscono limitazioni inferiori al cammino minimo $s \to i$. Con le opportune differenze valgono le stesse proprietà della formulazione in avanti, ovvero: la lunghezza di un qualsiasi cammino $P : s \to t$ con le nuove distanze è

$$d''(P) = \sum_{(ij)\in P} (d_{ij} - \pi_j + \pi_i) = d(P) + \pi_s - \pi_t$$

In questo caso tutte le lunghezze dei cammini $s \to t$ sono alterate del fattore costante $\pi_s - \pi_t$, da cui

$$\delta'' = \delta + \pi_s - \pi_t$$

La lunghezza di un cammino P^k da un nodo generico k a t vale, con le distanze d''

$$d''(P^k) = \sum_{(ij)\in P^k} (d_{ij} - \pi_j + \pi_i) = d(P^k) + \pi_k - \pi_t$$

e quindi

$$\delta''^k = \delta^k + \pi_k - \pi_t \tag{7}$$

5. Algoritmo bidirezionale con potenziali

Si effettui l'algoritmo di Dijkstra sia a partire dalla sorgente con lunghezze d' che a partire dalla destinazione con distanze d''. I potenziali scelti per le due ricerche sono arbitrari ed in generale diversi e devono solo essere ammissibili rispettivamente in (3) e (6) con il vincolo $\pi^t = 0$ e $\pi_s = 0$.

Come nell'algoritmo bidirezionale precedente, quando si trova un arco (i,j) con $i \in S_s$ e $j \in S^t$ si calcola la lunghezza vera del cammino $s \to i \to j \to t$ e la si confronta con quella $d(P)$ del migliore cammino P trovato finora, eventualmente aggiornandola.

L'algoritmo termina non appena si visita un nodo k tale che $\delta_k + \pi^k \geq d(P)$ nella ricerca in avanti oppure $\delta^k + \pi_k \geq d(P)$ nella ricerca all'indietro. La condizione $\delta_k + \pi^k \geq d(P)$ si può esprimere anche come $\delta'_k + \pi^s \geq d(P)$ (da (5)) dove δ'_k è direttamente disponibile dall'algoritmo di Dijkstra. Si noti ancora che l'algoritmo di Dijkstra, operando con distanze d', visita i nodi per valori δ'_k crescenti, ovvero per valori $\delta_k + \pi^k$ crescenti. Analogamente la condizione $\delta^k + \pi_k \geq d(P)$ si può esprimere come $\delta''^k + \pi_t \geq d(P)$ (da (7)).

Dimostriamo che alla terminazione dell'algoritmo il cammino minimo è stato trovato. Sia P il cammino migliore fra quelli trovati con lunghezza $d(P)$ e sia Q un cammino generico, fra quelli non trovati, con lunghezza $d(Q)$. Quindi esiste un nodo $h \in Q$ non visitato né dalla ricerca in avanti né da quella all'indietro. Sia Q_h la parte di cammino Q da s a h e sia Q^h la parte di cammino Q da h a t e quindi $d(Q) = d(Q_h)+d(Q^h)$. Siccome i potenziali π^i sono limitazioni inferiori ai cammini minimi da i a t, cioè $\pi^i \leq \delta^i$, abbiamo in particolare per il nodo h considerato

$$\delta_h + \pi^h \leq d(Q_h) + \pi^h \leq d(Q_h) + \delta^h \leq d(Q_h) + d(Q^h) = d(Q)$$

Analogamente, nell'altra direzione si ha

$$\delta^h + \pi_h \leq d(Q^h) + \pi_h \leq d(Q^h) + \delta_h \leq d(Q^h) + d(Q_h) = d(Q)$$

Sia k il nodo per il quale si verifica una delle due condizioni per la terminazione. Siccome h non è stato visitato e l'algoritmo visita i nodi in S_s per valori $\delta_i + \pi^i$ crescenti e in S^t per valori $\delta^i + \pi_i$ crescenti, si ha $\delta_k + \pi^k \leq \delta_h + \pi^h$ se $k \in S_s$ oppure $\delta^k + \pi_k \leq \delta^h + \pi_h$ se $k \in S^t$.

In entrambi i casi la condizione di terminazione implica $d(P) \leq d(Q)$.

6. Scelta dei potenziali

Sono state proposte varie scelte di potenziali. Una delle più efficaci consiste nella scelta di un sottoinsieme di nodi come di 'landmark' e di basare la scelta dei potenziali sulle distanze fra i landmark. Più esattamente sia $L \subset N$ un sottoinsieme sufficientemente più piccolo di N da poter calcolare e memorizzare tutte le distanze fra un qualsiasi nodi in N e un qualsiasi nodo di L ma non tanto piccolo da rendere poco utili i potenziali. La diseguaglianza triangolare implica

$$\delta_\ell \leq \delta_i + \delta^i_\ell, \qquad \text{cioè} \qquad \delta_\ell - \delta^i_\ell \leq \delta_i \qquad \ell \in L, i \in N$$

$$\delta^\ell_i \leq \delta^\ell_s + \delta_i, \qquad \text{cioè} \qquad \delta^\ell_i - \delta^\ell_s \leq \delta_i \qquad \ell \in L, i \in N$$

$$\delta^\ell \leq \delta^\ell_i + \delta^i, \qquad \text{cioè} \qquad \delta^\ell - \delta^\ell_i \leq \delta^i \qquad \ell \in L, i \in N$$

$$\delta^i_\ell \leq \delta^i + \delta^t_\ell, \qquad \text{cioè} \qquad \delta^i_\ell - \delta^t_\ell \leq \delta^i \qquad \ell \in L, i \in N$$

Quindi, dati i valori memorizzati δ^i_ℓ e δ^ℓ_i, una limitazione inferiore di δ_i è data da $\max\left\{\delta_\ell - \delta^i_\ell, \delta^\ell_i - \delta^\ell_s\right\}$, e una limitazione inferiore di δ^i è data da $\max\left\{\delta^\ell - \delta^\ell_i, \delta^i_\ell - \delta^t_\ell\right\}$. Una migliore limitazione inferiore si può ottenere prendendo il massimo valore, cioè

$$\pi_i = \max_{\ell \in L} \max\left\{\delta_\ell - \delta^i_\ell, \delta^\ell_i - \delta^\ell_s\right\}, \qquad \pi^i = \max_{\ell \in L} \max\left\{\delta^\ell - \delta^\ell_i, \delta^i_\ell - \delta^t_\ell\right\} \tag{8}$$

Potrebbe sorgere la preoccupazione che i valori così calcolati non soddisfino la proprietà $\pi_j - \pi_i \leq d_{ij}$ e $\pi^i - \pi^j \leq d_{ij}$. La proprietà è soddisfatta come si può vedere facilmente. Limitiamoci a dimostrare alcuni casi. Sia ad esempio

$$\pi^i = \delta^\ell - \delta^\ell_i, \qquad \pi^j = \delta^\ell - \delta^\ell_j$$

dove ℓ è lo stesso landmark per entrambi i casi. Allora

$$\pi^i - \pi^j = \delta^\ell - \delta^\ell_i - (\delta^\ell - \delta^\ell_j) = \delta^\ell_j - \delta^\ell_i \leq d_{ij}$$

dove la diseguaglianza deriva dalla proprietà di cammino minimo $\ell \to j$. Sia invece

$$\pi^i = \delta^i_\ell - \delta^t_\ell, \qquad \pi^j = \delta^j_\ell - \delta^t_\ell$$

Allora

$$\pi^i - \pi^j = \delta^i_\ell - \delta^t_\ell - (\delta^j_\ell - \delta^t_\ell) = \delta^i_\ell - \delta^j_\ell \leq d_{ij}$$

dove la diseguaglianza deriva dalla proprietà di cammino minimo $i \to \ell$. Consideriamo il caso di massimo ottenuto per landmark diversi

$$\pi^i = \delta^i_{\ell_1} - \delta^t_{\ell_1}, \qquad \pi^j = \delta^j_{\ell_2} - \delta^t_{\ell_2} > \delta^j_{\ell_1} - \delta^t_{\ell_1}$$

Allora

$$\pi^i - \pi^j = \delta^i_{\ell_1} - \delta^t_{\ell_1} - (\delta^j_{\ell_2} - \delta^t_{\ell_2}) < \delta^i_{\ell_1} - \delta^t_{\ell_1} - (\delta^j_{\ell_1} - \delta^t_{\ell_1}) = \delta^i_{\ell_1} - \delta^j_{\ell_1} \leq d_{ij}$$

Gli altri casi si dimostrano in modo analogo.

Se il nodo i si trova sul cammino minimo fra s e il landmark ℓ, allora π_i è esattamente il valore di cammino minimo $s \to i$ e, analogamente, π^i è il valore di cammino minimo $i \to t$ se i si trova sul cammino minimo fra il landmark ℓ e t.

Il calcolo dei valori δ^ℓ_i si effettua con $|L|$ esecuzioni in avanti dell'algoritmo di Dijkstra, ognuna della quali a partire da un landmark fino a visitare tutti i nodi. Analogamente il calcolo dei valori δ^i_ℓ si effettua con $|L|$ esecuzioni all'indietro dell'algoritmo di Dijkstra, a partire da un landmark fino a visitare tutti i nodi. Si tratta quindi di un calcolo che richiede un certo tempo, ma viene fatto una volta sola off-line e i suoi dati vengono poi utilizzati in tempo costante da ogni esecuzione successiva per scelte ogni volta diverse delle sorgenti e delle destinazioni.

Se il grafo è simmetrico basta una delle due esecuzioni perché l'esecuzione in avanti produce gli stessi risultati dell'esecuzione all'indietro. I grafi stradali sono 'quasi' simmetrici e le due esecuzioni vanno fatte entrambe. Tuttavia i valori δ^ℓ_i e δ^i_ℓ sono molto vicini e questo fatto viene sfruttato nella memorizzazione. Infatti la maggior parte dei bit dei due numeri sono uguali e quindi vengono memorizzati una sola volta, mentre gli ultimi bit sono diversi e vengono memorizzati separatamente. Questo permette una compressione

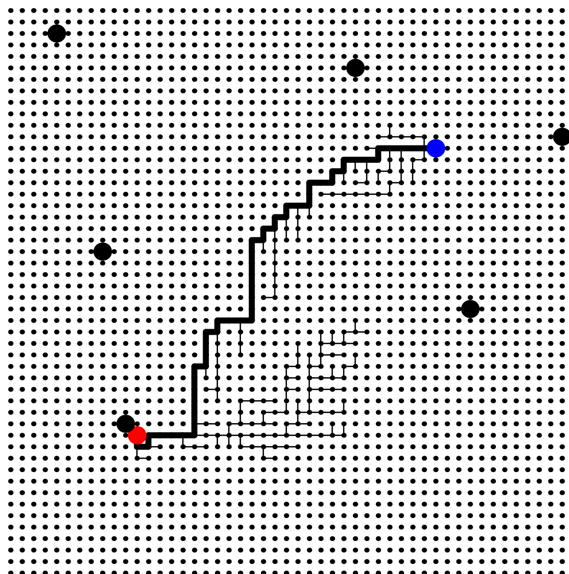


Figura 3 – Landmark casuali

dei dati di quasi il 50%. Se il grafo ha un milione di nodi e 16 landmark (il massimo con la tecnologia attuale se i nodi sono un milione) questa compressione permette di memorizzare i dati su una flash card e quindi rendere più veloce la lettura dei dati.

Il calcolo dei potenziali secondo (8), che va fatto ogni qualvolta si debba calcolare un cammino minimo fra due nuovi nodi, si può eseguire direttamente durante l'esecuzione dell'algoritmo di Dijkstra, con grande risparmio computazionale, non appena un nodo viene raggiunto.

Presentiamo due scelte di landmark. La prima è casuale. Viene deciso il numero p di landmark e poi vengono scelti con probabilità uniforme p nodi da N.

In Figura 3 si vede una scelta casuale di 6 landmark su 2500 nodi con i nodi visitati e il cammino minimo. Si noti come nelle parti di grafo dove mancano landmark l'algoritmo debba visitare molti nodi. I nodi visitati sono 206 con un'efficienza 0.262136.

Nella Figura 4 si vedono i risultati per nove scelte causali della sorgente e della destinazione, con gli stessi landmark. I tre numeri sotto ogni figura sono il numero di nodi del cammino minimo, il numero di nodi visitati e l'efficienza.

Il secondo modo di generare landmark cerca di distribuirli in modo uniforme sul grafo. Inizialmente si sceglie un nodo a caso, poi si genera il nodo più distante da questo, poi quello più distante dall'insieme generato e così di seguito fino a generare il numero desiderato di landmark. Si possono usare sia le distanze effettive che le distanze valutate come numero di archi. Se si usano le distanze effettive in questa fase si possono calcolare anche il valori δ^i_ℓ. Per trovare il nodo più distante dai landmark generati si può usare una struttura a heap che contiene tutti i nodi del grafo e le distanze dall'insieme dei landmark. La radice dello heap è il nodo più distante. Quando si preleva questo nodo (e si aggiorna lo heap) e si calcolano le distanze da questo landmark a tutti i nodi, si aggiorna anche lo heap, nodo per nodo, diminuendo eventualmente il valore della distanza dall'insieme.

In Figura 5 si vedono 6 landmark generati in modo equidistante (sono stati generati nel seguente ordine:

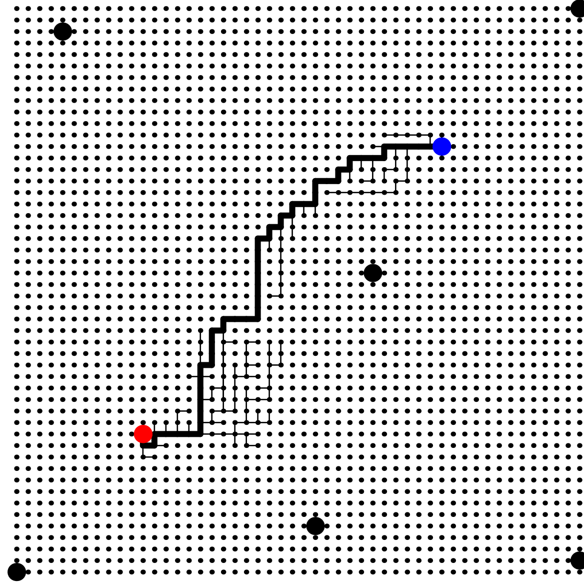
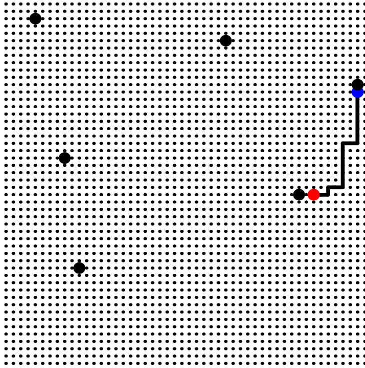


Fig. 5 – Landmark equidistanti, cammino minimo

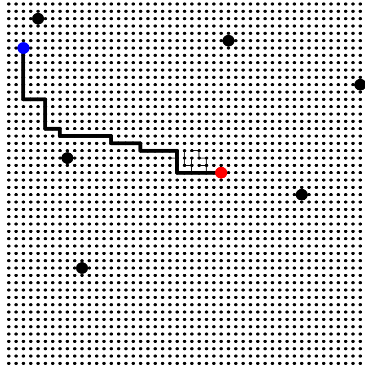
prima quello casuale in alto a sinistra, poi in basso a destra, in alto a destra, in basso a sinistra, in centro e in basso nel mezzo) e i nodi visitati per ls stessa coppia sorgente-destinazione. I nodi visitati sono 168 (contro i 54 del cammino minimo) per un'efficienza di 0.321429.

Nella Figura 6 si vedono i risultati per le stesse nove scelte causali della sorgente e della destinazione, con gli stessi landmark.

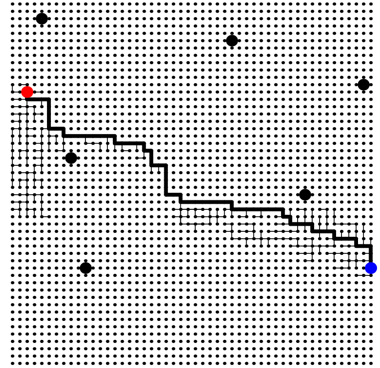
Con tecniche di questo genere (16 landmark generati in vario modo) si riescono a calcolare con tempi inferiori ai 200 ms i cammini minimi per grafi stradali di più di 6 milioni di nodi e 15 milioni di archi (corrispondenti ad una parte degli Stati Uniti).



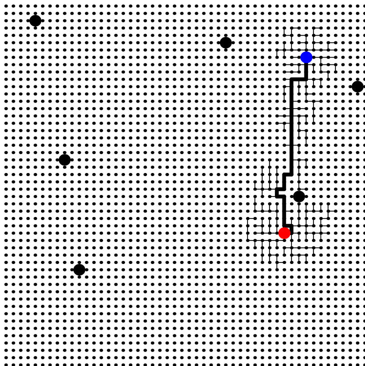
21, 22, 0.954545



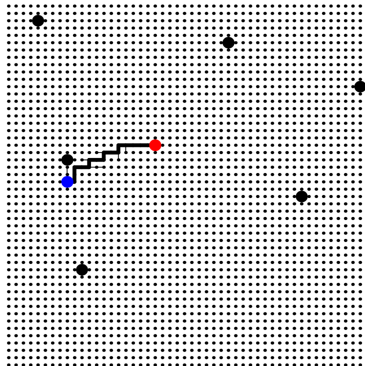
45, 58, 0.775862



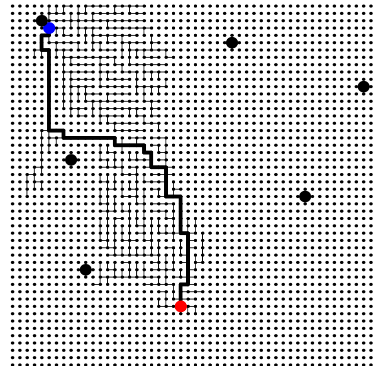
72, 318, 0.226415



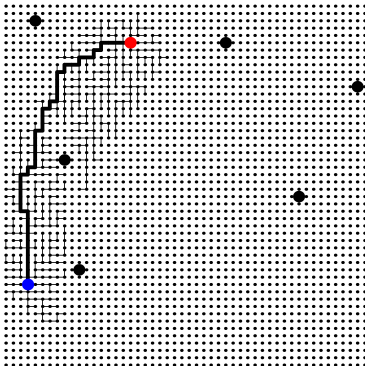
32, 236, 0.135593



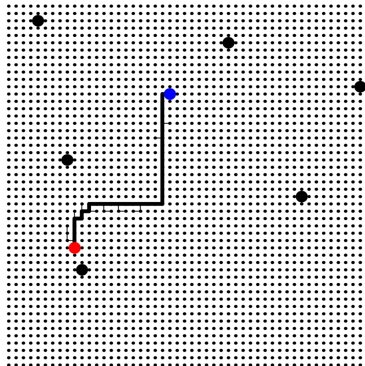
18, 32, 0.5625



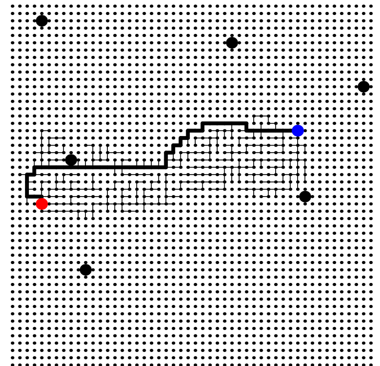
61, 582, 0.104811



50, 436, 0.114679

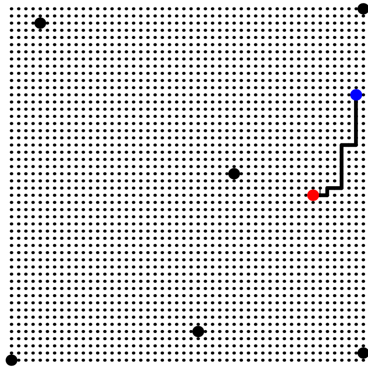


35, 50, 0.7

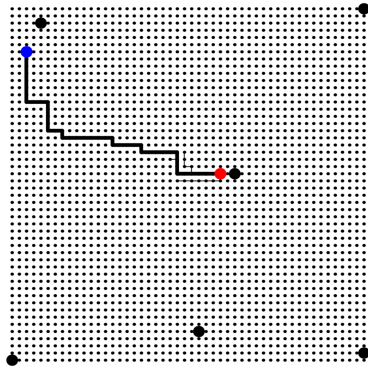


52, 360, 0.144444

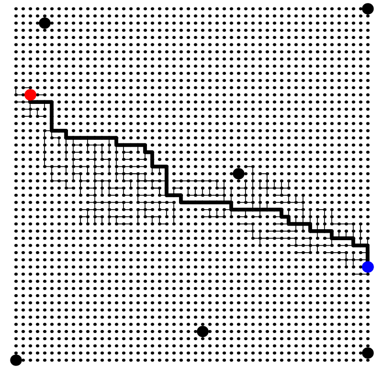
Fig. 4 – Landmark casuali



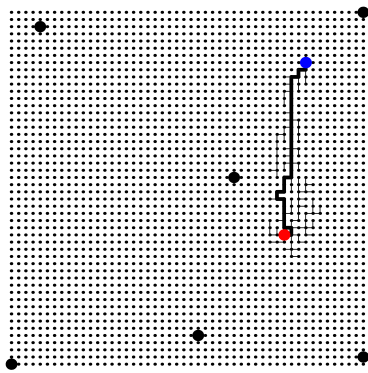
21, 24, 0.875



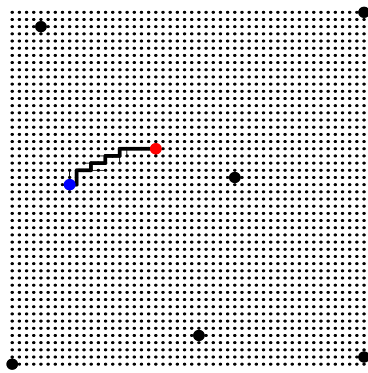
45, 58, 0.775862



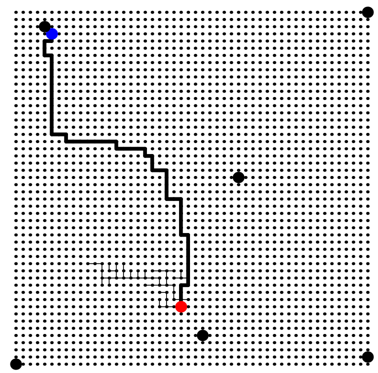
72, 424, 0.169811



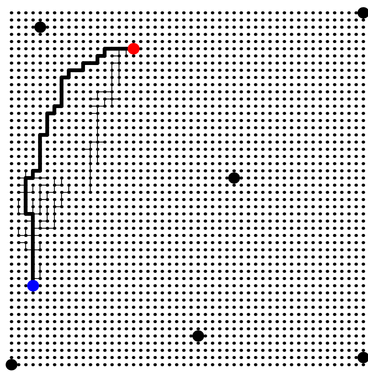
32, 120, 0.266667



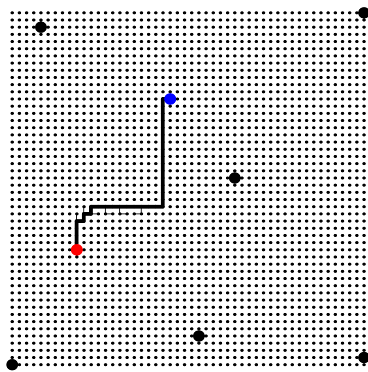
18, 32, 0.5625



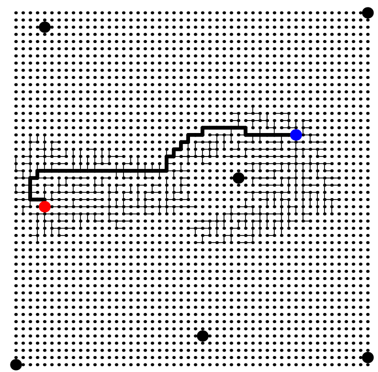
61, 108, 0.564815



50, 148, 0.337838



35, 44, 0.795455



52, 566, 0.0918728

Fig. 6 – Landmark equidistanti