Algoritmi di approssimazione

A.A. 2016-2017

Approssimazione e riduzione polinomiale
Metodo «pricing»

Approssimazione e riduzione polinomiale

Abbiamo avuto occasione di parlare dei due problemi: copertura di vertici e copertura di insiemi, entrambi NP-completi.
Sappiamo pertanto che:

$$\text{Copertura di vertici} \propto \text{Copertura di insiemi}$$

La relazione, introdotta per algoritmi di decisione, continua ad essere valida per la versione ottimizzazione (minimizzazione) dei due problemi.

La riduzione di “copertura di vertici” a “copertura di insiemi” puo` essere fatta scegliendo come universo (*U*) l’insieme degli *archi* del grafo e come sottinsiemi della famiglia *F* gli insieme degli archi incidenti in uno stesso vertice. E` facile dimostrare che il problema copertura di insiemi cosi` ottenuto ha una soluzione minima di cardinalita` k se e solo se il problema originale copertura di vertici ha una soluzione minima di cardinalita` k.

Approssimazione e riduzione polinomiale

La riduzione dice:

“se esiste un algoritmo che risolve il problema copertura di insiemi in tempo polinomiale, allora risolve anche il problema copertura di vertici in tempo polinomiale”.

Si puo` estendere questa affermazione agli algoritmi di approssimazione? , possiamo cioe` utilizzare l’algoritmo di approssimazione per la copertura di insiemi di dimensione minima per realizzarne uno per copertura di vertici di dimensione minima?

SI

Si ottiene un algoritmo con rapporto di approssimazione H(d). Accettabile per d *piccolo (*d il massimo grado di un nodo)

- Ma si tratta di un caso particolare!

Bisogna fare attenzione quando si usa la riduzione per «disegnare» un algoritmo di approssimazione: non in tutti i casi in cui esiste una riduzione polinomiale, si ha un algoritmo di approssimazione utilizzabile.

Approssimazione e riduzione polinomiale

Consideriamo un caso simile al precedente:

$$\text{Insieme indipendente} \propto \text{Copertura di vertici}$$

Possiamo utilizzare l'algoritmo di approssimazione per la copertura di insiemi di dimensione minima per realizzare un insieme indipendente di dimensione massima?

Questa volta la risposta e` NO

Il problema sorge quando si cerca di determinare il fattore di approssimazione per il problema «insieme indipendente»; la soluzione approssimata può essere molto lontana da quella ottima. Supponiamo che la copertura di vertici ottimale S* e l'insieme indipendente ottimale I* abbiano entrami dimensione |V|/2. Utilizzando l'algoritmo di approssimazione, con fattore 2 (che vedremo), per copertura di vertici, potremmo ottenere la soluzione approssimata S = V e in tal caso l'insieme indipendente approssimato I = V – S sarebbe vuoto.

Algoritmi approssimati: metodo pricing

Il metodo “pricing” e` motivato da una prospettiva economica.

L’algoritmo Greedy-Weighted-Set-Cover della copertura di insiemi definiva per ogni sottinsieme S_i della famiglia un valore $w_i / |S_i - R|$, che può essere pensato come la parte di costo di cui si caricano gli elementi di S_i che vengono coperti con la scelta di S_i stesso.

Nel metodo pricing si puo` pensare a questo costo come a un costo di condivisione.

La somma dei costi condivisi è il costo della copertura trovata dall’algoritmo:

$$|C| = \sum_{j \in U} p_j$$

L’esempio “copertura di insiemi” può pertanto essere pensato anche come esempio di pricing.

Algoritmi approssimati pricing: copertura di vertici

Consideriamo ora il problema «copertura di vertici» nella versione pesata.
Dato il grafo G = <V, E>, ad ogni vertice i del quale è associato un peso $w_i \geq 0$, trovare un insieme di vertici S che copra tutti gli archi di G con peso $W(S) = \sum_{i \in S} w_i$ minimo.

Quando si seleziona un vertice i, vengono coperti tutti gli archi incidenti in i; sarebbe «unfair» caricare questi archi di un costo totale maggiore del costo del vertice.

Definiamo **fair** il prezzo p_e se, per ogni vertice i il costo totale degli archi incidenti in i e` al massimo il costo del vertice stesso.

$$\sum_{e=(i,j)} p_e \leq w_i$$

Algoritmi approssimati pricing: copertura di vertici

Per ogni copertura di vertici S e per prezzi fair non negativi si ha:

$$\sum_{e \in E} p_e \leq \sum_{i \in S} \sum_{e=(i,j)} p_e \leq W(S)$$

Poiché S è una copertura ogni arco che contribuisce con almeno un termine p_e, eventualmente più di uno se entrambi i suoi vertici incidenti sono in S.

Chiamiamo un nodo i **tight** se $\sum_{e=(i,j)} p_e = w_i$

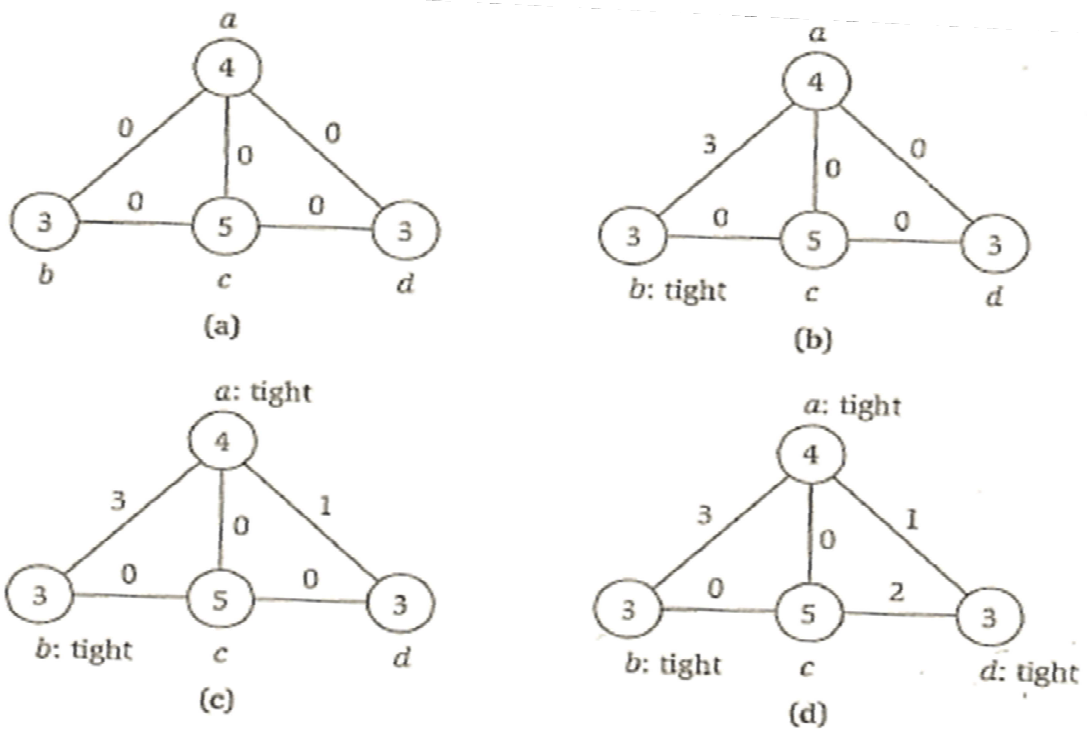
Algoritmi approssimati pricing: copertura di vertici

```
Vertex-Cover-Approx(G, w)
        Set p_e = 0 for all e ∈ E
        While there is an edge e = (i, j) such that neither i
                nor j is tight
                Select such edge e
                Increase p_e without violating fairness
        EndWhile
        Let S be the set of all tight nodes
        Return S
```

L'algoritmo Vertex-Cover-Approx trova una copertura e attribuisce i costi, che usa per guidare la selezione dei vertici. Da questo punto di vista l'algoritmo può essere anche considerato greedy.

Algoritmi approssimati pricing: copertura di vertici

Esempio



Analisi del rapporto di approssimazione

L'insieme S costruito dall'algoritmo è una copertura di vertici. Infatti se un arco 'e' non fosse coperto, nessuno dei suoi vertici incidenti sarebbe tight e ciò contraddice il fatto che il ciclo dell'algoritmo è terminato.

Sappiamo inoltre che per ogni copertura:

$$\sum_{e \in E} p_e \leq W(S)$$

quindi anche per una copertura di costo minimo S*: $\sum_{e \in E} p_e \leq W(S^*)$

Tutti i nodi in S sono tight, quindi $\sum_{e=(i,j)} p_e = w_i$ per ogni nodo $i \in S$ e ogni arco può pagare al massimo due volte:

$$W(S) = \sum_{i \in S} \sum_{e=(i,j)} p_e \leq 2 \sum_{e \in E} p_e$$

Analisi del rapporto di approssimazione

Otteniamo pertanto, se S* è una copertura di minimo costo:

$$W(S) \leq 2 \sum_{e \in E} p_e \leq 2\, W(S^*)$$

L'insieme S costruito dall'algoritmo Vertex-Cover-Approx è una copertura di vertici e il suo costo è al massimo il doppio di una copertura di peso minimo.

Algoritmi approssimati pricing: cammini disgiunti

Consideriamo un ultimo problema risolto in modo approssimato con un algoritmo *greedy* e confrontiamo questa soluzione con un algoritmo di approssimazione ottenuto con il metodo *pricing*.

Dato un grafo orientato G, k coppie di nodi (s_i,t_i) e un valore intero *c,* che chiamiamo capacità, consideriamo ciascuna coppia di nodi come una **richiesta di passaggio** dal nodo s_i al nodo t_i, appartenenti rispettivamente ad un insieme di nodi "sorgente" S e ad un insieme di nodi "destinazione" T. Vogliamo trovare i cammini che permettono di soddisfare il maggior numero possibile di richieste di passaggio, facendo in modo che ciascun arco del grafo sia presente in al più *c* dei cammini ottenuti.

Algoritmi approssimati pricing: cammini disgiunti

Il problema dei cammini disgiunti è comune nelle applicazioni in rete: *cammini in Internet* che portano dati o *cammini attraverso la rete telefonica*[1] che trasportano traffico vocale.

Percorsi che condividono archi possono interferire tra loro, e troppi cammini che condividono uno stesso arco provocano problemi nella maggior parte delle applicazioni. La massima quantità di archi condivisibili varia a seconda dell'applicazione.

Il requisito della *completa disgiunzione* è il vincolo più forte che elimina tutte le interferenze, ma nei casi in cui un po' di condivisione è permessa si possono costruire algoritmi di approssimazione migliori .

Consideriamo per primo un semplice algoritmo greedy per il caso in cui $c = 1$, cioè i cammini sono disgiunti.

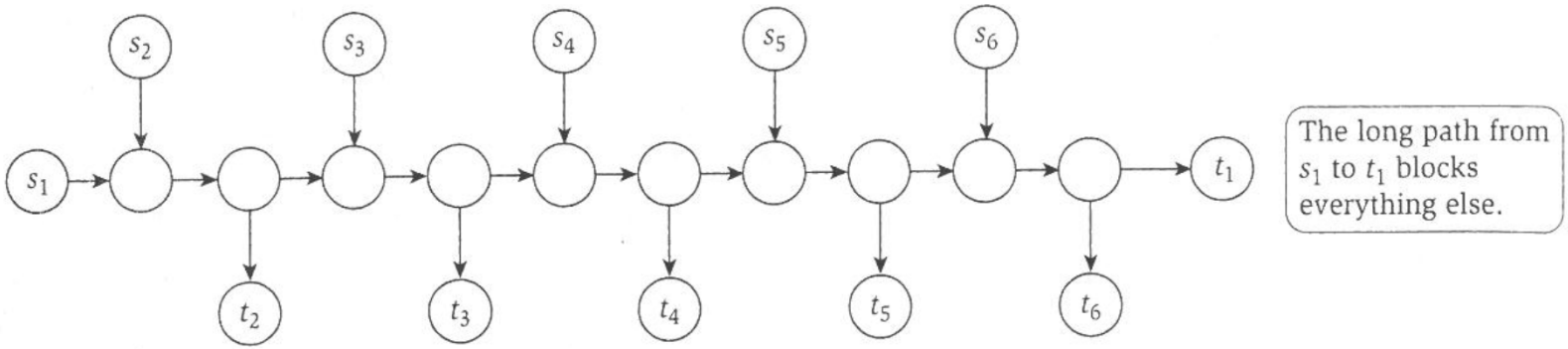
[1] A researcher from the telecommunications industry once gave the following explanation for the distinction between Maximum Disjoint Paths and network flow, and the broken reduction in the previous paragraph. On Mother's Day, traditionally the busiest day of the year for telephone calls, the phone company must solve an enormous disjoint paths problem: ensuring that each source individual s_i is connected by a path through the voice network to his or her mother t_i. Network flow algorithms, finding disjoint paths between a set S and a set T, on the other hand, will ensure only that each person gets their call through to *somebody's* mother.

Algoritmi approssimati pricing: cammini disgiunti

Il problema di trovare il massimo numero di cammini disgiunti non è solo **NP**-completo, ma anche difficile da approssimare; è stato dimostrato che, a meno che non sia **P** = **NP**, non si può trovare un algoritmo polinomiale che abbia un rapporto di approssimazione molto migliore di $O(\sqrt{m})$ per grafi orientati arbitrari con m archi, che è il rapporto realizzato dall'algoritmo greedy.

L'algoritmo deve essere disegnato in modo da evitare che il numero di cammini selezionati sia troppo minore del massimo.

Algoritmi approssimati pricing: cammini disgiunti



Se viene selezionato per primo il cammino dal nodo sorgente s_1 al nodo destinazione t_1, non si possono trovare altri cammini che non usino archi in esso contenuti. Vengono scartati tutti gli altri 5 cammini.

E` cruciale in questo esempio favorire la scelta di cammini brevi a quella di cammini lunghi.

Algoritmi approssimati pricing: cammini disgiunti

Per questo motivo, e più in generale per limitare il numero di cammini con cui un cammino selezionato può interferire, l'algoritmo greedy seleziona ad ogni iterazione un cammino “breve”.

Se il grafo G è connesso, l'algoritmo seleziona almeno un cammino.

```
Greedy-Disjoint-Paths:
Set I = ∅
Until no new path can be found
  Let P_i be the shortest path (if one exists) that is edge-disjoint
      from previously selected paths, and connects some (s_i, t_i) pair
      that is not yet connected
  Add i to I and select path P_i to connect s_i to t_i
EndUntil
```

Analisi del rapporto di approssimazione

Chiamiamo «lungo» un cammino formato da almeno $\sqrt{m}$ archi, «breve» altrimenti.

Siano I e I_s l'insieme dei cammini e dei cammini "brevi" nella soluzione fornita dall'algoritmo e I* e I_s* l'insieme dei cammini e dei cammini "brevi" in una soluzione ottima.

- I* avrà più cammini di I. Consideriamo i cammini corti in I* che *non* sono in I. Se un cammino corto P_i* in I* fosse stato libero a un certo punto l'algoritmo l'avrebbe scelto prima di scegliere un cammino lungo. Ma siccome l'algoritmo non l'ha scelto, qualche arco e in P_i* di un cammino P_j già scelto. Diciamo che e *blocca* P_i*.
- Ora i cammini scelti dall'algoritmo *crescono* in lunghezza. Il cammino P_j è stato scelto prima di considerare P_i* e quindi

$$| P_j| \leq | P_i^*| \leq \sqrt{m}$$

quindi P_j è corto.

Ora ogni arco in P_j può bloccare al massimo un cammino in I*. Quindi ogni cammino in corto in I può bloccare al più $\sqrt{m}$ cammini in I_s^*. E quindi otteniamo:

$$|I_s^* - I| \leq \sum_{j \in Is} |P_j| \leq |I_s| \sqrt{m}$$

Ora I* consiste di tre tipi diversi di cammini:

- Cammini lunghi in numero massimo di $\sqrt{m}$
- Cammini che sono anche in I
- Cammini brevi che non sono in I, in numero limitato da $|I_s| \sqrt{m}$

Ogni cammino in I_s può bloccare al più $\sqrt{m}$ cammini in I^*_s

Mettendo le cose insieme si ha:

$$|I^*| \leq \sqrt{m} + |I| + |I_s^* - I| \leq \sqrt{m} + |I| + \sqrt{m}\,|I_s| \leq (2\sqrt{m} + 1)\,|I|$$

L'algoritmo Greedy-Disjoint-Paths è un algoritmo $(2\sqrt{m} + 1)$ - approssimato

Algoritmi approssimati pricing: cammini disgiunti

L'algoritmo greedy precedente può essere interpretato come "pricing": i cammini devono pagare per usare un arco e ogni arco ha un costo unitario.

Vediamo ora una soluzione pricing per il caso in cui al massimo $c > 1$ percorsi possano condividere un arco.

Usiamo un parametro moltiplicativo β per incrementare il costo (la lunghezza) di un arco ogni volta che viene utilizzato da un percorso. Questo incoraggia l'algoritmo a "diversificare" i cammini piuttosto che concentrarli su singoli archi.

La lunghezza di un percorso P è la somma delle lunghezze degli archi che lo formano:

$$\ell(P) = \sum_{e \in P} \ell_e$$

Algoritmi approssimati pricing: cammini disgiunti

```
Greedy-Paths-with-Capacity:
Set I = ∅
Set edge length ℓ_e = 1 for all e ∈ E
Until no new path can be found
  Let P_i be the shortest path (if one exists) so that adding P_i to
      the selected set of paths does not use any edge more than c
      times, and P_i connects some (s_i, t_i) pair not yet connected
  Add i to I and select path P_i to connect s_i to t_i
  Multiply the length of all edges along P_i by β
EndUntil
```

Analisi del rapporto di approssimazione

Per trovare il rapporto di approssimazione esaminiamo il caso $c = 2$. In questo caso l'approssimazione migliore si ottiene con $\beta = m^{1/3}$.

Il punto chiave nell'analisi dell'algoritmo greedy è stato distinguere tra cammini «brevi» e «cammini «lunghi».
Nell'algoritmo greedy la funzione lunghezza era ovvia.

Per il caso che stiamo esaminando, definiamo *breve* un cammino se la sua lunghezza è minore di β^2.

- Sia I_s l'insieme dei cammini brevi selezionati dall'algoritmo
- Sia poi I^* una soluzione ottima e
- P_i^* ($i \in I^*$) l'insieme dei cammini che compongono tale soluzione.

Algoritmi approssimati pricing: cammini disgiunti

Analisi del rapporto di approssimazione

Consideriamo i cammini *corti* in I_s.

Ora però il peso dei cammini *varia* durante l'esecuzione dell'algoritmo, quindi dobbiamo decidere a che punto considerare la lunghezza dei cammini.

Scegliamo di considerare la lunghezza dei cammini nel *momento in cui sono stati scelti tutti i cammini (reali) corti*.

- Sia $\underline{\ell}()$ la funzione che dà la lunghezza a questo momento.

- Per un cammino P la lunghezza è data dunque da $\sum_{e \in P} \underline{\ell}_e$.

-Consideriamo quindi un cammino P* *corto* se $\underline{\ell}(P^*) < \beta^2$, e *lungo* altrimenti.

- Sia I_s^* l'insieme dei cammini corti in I* secondo questo criterio.

Algoritmi approssimati pricing: cammini disgiunti

Analisi del rapporto di approssimazione

Consideriamo una coppia (s_i, t_i) in I* che non è connessa dall'algoritmo di approssimazione. Allora $\underline{\ell}(P_i^*) \geq \beta^2$

Infatti se ci fosse un cammino corto (rispetto a $\underline{\ell}$) che collega s_i e t_i questo sarebbe stato scelto dall'algoritmo. Poichè al momento in cui consideriamo $\underline{\ell}$ tutti i cammini corti sono già stati scelti dall'algoritmo il peso del cammimo P_i^* deve essere $\geq \beta^2$.

Ora la lunghezza totale dei commini nel grafo è $\sum_{e \in E} \underline{\ell}_e$.

-Tale somma include senz'altro m (numero totale di archi). L'aggiunta di un cammino corto (scelto con lunghezza $\leq \beta^2$) in I_s può produrre un aumento di al più β^3 (la lunghezza del cammino viene moltiplicata per β).

L'insieme I_s dei cammini selezionati dall'algoritmo e le lunghezze $\underline{\ell}$, soddisfano $\sum_{e \in E} \underline{\ell}_e \leq \beta^3 |I_s| + m$

Algoritmi approssimati pricing: cammini disgiunti

Finalmente abbiamo

L'algoritmo Greedy-Paths-with-Capacity, usando $\beta = m^{1/3}$, è un algoritmo $(4m^{1/3} + 1)$ - approssimato nel caso $c = 2$.

Abbiamo già visto che $\underline{\ell}(P_i^*) \geq \beta^2$ per ogni $i \in (I^* - I)$. Sommando

$$\sum_{i \in |I^*-I|} \underline{\ell}(P_i^*) \geq \beta^2 \, | \, I^* - I \, |$$

Ma ogni arco appartiene al più a due cammini, quindi

$$\sum_{i \in |I^*-I|} \underline{\ell}(P_i^*) \leq 2 \sum_{e \in E} \underline{\ell}_e$$

E combinando si ottiene:

$$\beta^2 \, | \, I^* \, | \; \leq \; \beta^2 \, | \, I^* - I \, | \; + \beta^2 \, | \, I \, | \; \leq \sum_{i \in |I^*-I|} \underline{\ell}(P_i^*) \; + \beta^2 \, | \, I \, | \leq$$

$$\leq 2 \sum_{e \in E} \underline{\ell}_e + \beta^2 \, | \, I \, | \; \leq 2 \, (\, \beta^3 \, | \, I \, | \; + m) + \beta^2 \, | \, I \, |$$

Dividendo per β^2, ponendo $\beta = m^{1/3}$ e assumendo $| \, I \, | \geq 1$ otteniamo $|I^*| \leq (4m^{1/3} + 1) \cdot | \, I \, |$

Algoritmi approssimati pricing: cammini disgiunti

Notiamo che aumentando la capacità dei canali da 1 a 2 il rapporto di approssimazone cade da $O(m^{1/2})$ a $O(m^{1/3})$.

Modificando la capacità *c*, l'algoritmo risulta fornire un rapporto di approssimazione pari a $2c\, m^{1/(c+1)} + 1$.

L'algoritmo Greedy-Paths-with-Capacity, usando $\beta = m^{1/(c+1)}$, è un algoritmo $(2cm^{1/(c+1)} + 1)$ - approssimato.

Algoritmi approssimati pricing: cammini disgiunti

Analisi del rapporto di approssimazione

Consideriamo una coppia (s_i, t_i) in I* che non è connessa dall'algoritmo di approssimazione. Allora $\underline{\ell}(P_i^*) \geq \beta^2$

Infatti se ci fosse un cammino corto disponibile (rispetto a $\underline{\ell}$) che collega s_i e t_i questo sarebbe stato scelto dall'algoritmo. Quindi P_i^* o e lungo o deve includere almeno un arco già "carico" che pesa quindi β^2 per cui il costo totale di P_i^* deve essere $\geq \beta^2$.

Ora la lunghezza totale dei commini nel grafo è $\sum_{e \in E} \underline{\ell}_e$.

- Tale somma include senz'altro m (numero totale di archi). L'aggiunta di un cammino corto (scelto con lunghezza $\leq \beta^2$) a quelli di I_s può produrre un aumento di al più β^3 (la lunghezza del cammino viene moltiplicata per β, ma $\underline{\ell}$ si considera solo sui cammini corti). Quindi:

L'insieme I_s dei cammini selezionati dall'algoritmo soddisfa

$$\sum_{e \in E} \underline{\ell}_e \leq \beta^3 |I_s| + m$$