Complementi ed Esercizi di Informatica Teorica II

Vincenzo Bonifaci

14 febbraio 2008

5 Problemi di ottimizzazione: lo Scheduling

Il problema dello SCHEDULING SU MACCHINE IDENTICHE è il seguente: dato un insieme di n lavori con tempi di processamento $l_1, \ldots, l_n$ ed un insieme di p macchine identiche, determinare uno *schedule* (decidere quando e su che macchina lanciare ogni lavoro) che minimizzi il tempo di completamento massimo dei lavori. Il tempo di completamento del j-esimo lavoro è dato dall'istante di tempo in cui il lavoro termina ed è quindi la somma del tempo in cui il lavoro è stato avviato e del suo tempo di processamento l_j.

5.1 List Scheduling e Largest Processing Time

È noto che lo SCHEDULING SU MACCHINE IDENTICHE è un problema NP-hard che ammette un algoritmo $(2 - 1/p)$-approssimato. Questo algoritmo, detto *List Scheduling*, considera in sequenza tutti i lavori e semplicemente assegna di volta in volta ogni lavoro alla macchina meno carica.

Vogliamo mostrare come, analogamente al caso del bin packing, considerare gli elementi in ingresso in un ordine particolare può migliorare la qualità dell'approssimazione. Applichiamo quindi l'ordinamento *Largest Processing Time* o *LPT* che ordina i lavori dal più lungo al più corto, dopodiché usiamo l'algoritmo List Scheduling come prima. Dimostriamo che l'algoritmo risultante ottiene un tempo di completamento $m(x)$ pari al più a $(4/3 - 1/(3p))$ volte l'ottimo $m^*(x)$.

Infatti, sia l_j la durata del lavoro che termina per ultimo. Abbiamo due casi: $l_j \leq m^*(x)/3$ oppure $l_j > m^*(x)/3$.

Nel primo caso, l'analisi è simile a quella per List Scheduling. Sia $W = \sum_{k=1}^{n} l_k$. Dato che il lavoro j è stato assegnato alla macchina meno carica, il tempo di completamento di ogni altra macchina è almeno $m(x) - l_j$. Questo implica che $W \geq p(m(x) - l_j) + l_j$ e quindi

$$m(x) \leq \frac{W}{p} + \frac{p-1}{p} l_j.$$

Dato che $m^*(x) \geq W/p$ e $m^*(x) \geq 3l_j$, otteniamo

$$m(x) \leq m^*(x) + \frac{p-1}{3p} m^*(x) = \left(\frac{4}{3} - \frac{1}{3p}\right) m^*(x).$$

Nel secondo caso, possiamo supporre senza perdita di generalità che il lavoro j sia anche il più corto; in caso contrario, possiamo eliminare dall'istanza tutti i lavori che seguono j nell'ordine LPT e condurre l'analisi sulla nuova istanza, in cui l'algoritmo ottiene un costo altrettanto alto rispetto all'ottimo. Nella soluzione ottima a questa istanza troncata sono stati assegnati al più due lavori ad una stessa macchina (altrimenti, essendo tutti i lavori più lunghi di l_j, avremmo $m^*(x) \geq 3l_j$). Dato che ci sono al più due lavori per macchina e dato l'ordine LPT, è facile convincersi che in questo caso l'algoritmo trova la soluzione ottima.

5.2 Uno schema di approssimazione polinomiale

Mentre il bin packing ammette uno schema di approssimazione *asintotica* polinomiale, lo scheduling ne ammette anche uno non asintotico (PTAS). Supponiamo infatti di avere a disposizione un algoritmo tempo polinomiale che, data una istanza del problema dello scheduling (SCHEDULING SU MACCHINE IDENTICHE), un numero reale ϵ e una scadenza T:

- ritorni il valore NO se non esiste una soluzione con un makespan $\leq T$;
- ritorni SI se esiste una soluzione con un makespan $\leq (1+\epsilon)T$ (in questo caso l'algoritmo ritorna anche la soluzione)

Notiamo che in alcuni casi l'algoritmo può ritornare indifferentemente sia SI che NO. Avendo a disposizione questa procedura, è possibile usarla per effettuare una ricerca binaria della soluzione al problema originale. Come intervallo di partenza è possibile usare l'intervallo $[L, 2L]$ dove L è il massimo tra la lunghezza massima dei lavori ($\max_i l_i$) e il carico medio per macchina ($\sum_i l_i/p$). Infatti, da quanto visto nell'analisi di List Scheduling, $L \leq m^*(x) \leq 2L$. Attraverso la ricerca binaria, possiamo individuare il più grande valore T tale che la procedura ritorna NO. La soluzione trovata su input $T+1$ ha costo al più $(1+\epsilon)(T+1) \leq (1+\epsilon)m^*(x)$. Il tempo necessario per la ricerca è il tempo necessario per la procedura moltiplicato $O(\log 2L) = O(\log L)$, quindi polinomiale nella lunghezza dell'input.

Naturalmente bisogna fornire una procedura con le caratteristiche enunciate sopra. Questa procedura si basa su idee simili a quelle usate nello schema di approssimazione asintotica per il bin packing:

1. rimuovi tutti i lavori corti, di durata $l_j \leq \epsilon T$;
2. raggruppa i lavori restanti in un numero di classi dipendente da ϵ, arrotondando le loro lunghezze;
3. trova in tempo polinomiale una soluzione ottima a questo problema più semplice;
4. considera l'errore dovuto agli arrotondamenti;
5. reinserisci i lavori corti considerando il tempo extra necessario.

Non dimostriamo che in effetti questa procedura è in grado di produrre in tempo polinomiale una soluzione di costo al più $(1+\epsilon)m^*(x)$ per ogni $\epsilon > 0$ fissato. Notiamo che comunque il tempo *non* è polinomiale in $(1/\epsilon)$ e non si ottiene quindi uno schema *pieno* di approssimazione (FPTAS). D'altronde l'esistenza di un tale schema per il problema dello scheduling implicherebbe P = NP.

Mostriamo però l'idea alla base del passo 3 dello schema. Consideriamo un'istanza del problema dello scheduling in cui abbiamo p macchine e n_1 lavori di lunghezza z_1, n_2 lavori di lunghezza z_2 ecc., fino a n_s lavori di lunghezza z_s. Vogliamo mostrare che se s è costante (ovvero se abbiamo un numero costante di tipologie di lavori), dato un intero T è possibile decidere in tempo polinomiale se esiste una soluzione per questa istanza con costo al più T. Supponiamo che $z_i \leq T$ per ogni i (altrimenti l'istanza non ha soluzione). Indichiamo con $M(x_1, x_2, \ldots, x_s)$ *il numero minimo di macchine necessarie per processare* x_1 *lavori di taglia* z_1*, ...,* x_s *lavori di taglia* z_s*, il tutto in tempo* $\leq T$. Osserviamo che l'istanza ha soluzione se e solo se $M(n_1, n_2, \ldots, n_s) \leq p$. Mostriamo ora come calcolare $M(n_1, \ldots, n_s)$ attraverso il tecnica della programmazione dinamica. Sia

$$V(x_1, \ldots, x_s) = \{(v_1, \ldots, v_s) : \sum_i v_i z_i \leq T, v_i \leq x_i\}$$

ovvero V è l'insieme di tutti i possibili modi in cui si possono assegnare parte dei lavori ad una macchina senza eccedere il costo T. L'insieme V può essere enumerato in tempo $O(n^s)$. Per la funzione M vale ora la seguente ricorrenza:

$$M(x_1, \ldots, x_s) \quad = \quad \begin{cases} 1, \text{ se } (x_1, \ldots, x_s) \in V(x_1, \ldots, x_s) \\ 1 + \min_{v \in V(x_1, \ldots, x_s)} M(x_1 - v_1, \ldots, x_s - v_s) \text{ altrimenti.} \end{cases}$$

Dato che una tabella che memorizza i valori di $M(x_1, \ldots, x_s)$ al variare di x ha dimensione $O(n^s)$ e dato che ogni valore può essere calcolato in tempo $O(n^s)$ in base ai valori che lo precedono nella tabella, questa procedura usa tempo polinomiale.

5.3 Scheduling con tempi di rilascio e vincoli di precedenza

Si è visto come il semplice algoritmo List Scheduling fornisca una approssimazione pari a $(2 - 1/p)$ per SCHEDULING SU MACCHINE IDENTICHE. Vogliamo qui mostrare come una semplice variante dell'algoritmo sia 2-approssimata anche per una versione più generale del problema dello scheduling in cui si hanno *tempi di rilascio* r_j dei lavori (ogni lavoro è rilasciato al tempo r_j e può quindi essere lanciato solo dopo tale istante) e *vincoli di precedenza* tra i lavori (alcuni lavori, per essere avviati, devono necessariamente attendere il completamento dei lavori da cui dipendono).

Indichiamo con $J_j \prec J_k$ il fatto che il lavoro J_k deve attendere il completamento del lavoro J_j prima di essere avviato. L'algoritmo che consideriamo è la semplice variante di List Scheduling in cui ogni volta che una macchina si

rende libera, il lavoro che viene lanciato è il primo lavoro *disponibile* sulla lista, intendendo con disponibile il fatto che il lavoro è già stato rilasciato e che i suoi predecessori sono stati tutti completati. Indichiamo con s_j il momento in cui l'algoritmo avvia il lavoro J_j.

Consideriamo lo schedule prodotto dall'algoritmo. Sia J_{j_1} il lavoro che termina per ultimo nello schedule. Partizioneremo il tempo totale dello schedule in due insiemi di intervalli tali che il tempo totale degli intervalli appartenenti a ciascuno di questi due insiemi è al più $m^*(x)$. In questo modo, il tempo totale sarà al più $2m^*(x)$.

Consideriamo l'ultimo istante t_1 prima di s_{j_1} in cui qualche macchina è libera. Dato che J_{j_1} non è stato avviato in questo slot di tempo, esso non era disponibile al tempo t_1, o perché uno dei suoi predecessori non era stato ancora completato, o perché esso stesso non era stato ancora rilasciato. Nel primo caso, consideriamo un lavoro $J_{j_2} \prec J_{j_1}$ ancora in fase di elaborazione al tempo t_1 e ripetiamo il ragionamento per J_{j_2}: consideriamo l'ultimo istante t_2 prima di s_{j_2}, ecc. Alla fine otterremo una sequenza di lavori $J_{j_k} \prec \ldots \prec J_{j_2} \prec J_{j_1}$ tale che il lavoro J_{j_k} non è stato avviato al tempo t_k perché non ancora rilasciato.

Partizioniamo ora lo schedule in due insiemi di intervalli come segue. Il primo insieme consiste degli intervalli $[0, r_k)$ e, per ogni i da 1 a k, $[s_{j_i}, s_{j_i} + l_{j_i})$; in altre parole, consiste di tutti quegli istanti in cui qualche lavoro della catena che abbiamo costruito è in fase di elaborazione, oltre all'intervallo che trascorre dall'inizio fino al rilascio del primo lavoro della catena. La somma delle lunghezze di questi intervalli è almeno pari a $m^*(x)$, in quanto ognuno dei lavori considerati è necessario al successivo.

Il secondo insieme di intervalli costituisce il tempo rimanente dello schedule. Notiamo che in questi intervalli non ci sono mai macchine ferme; questo fatto è dovuto al modo in cui sono stati scelti i lavori e in particolare alla definizione degli istanti t_i. Dato che in questi intervalli non ci sono mai macchine ferme, la somma delle loro lunghezze non può essere maggiore del carico medio delle macchine, $\sum_{i=1}^{n} l_i/p$. D'altra parte questo valore è minore o uguale al tempo di completamento dell'ottimo.

5.4 Open shop scheduling

Consideriamo ancora un'altra variante dello scheduling detta *open shop scheduling*. In questo problema, ogni lavoro J_j è composto da un insieme di operazioni $\{O_{1,j}, \ldots, O_{m,j}\}$. L'operazione $O_{i,j}$, relativa al lavoro J_j, deve essere svolta necessariamente sulla macchina M_i e richiede un certo tempo $p_{i,j}$. Le operazioni possono comunque essere compiute in un qualsiasi ordine, ma una macchina può eseguire una sola operazione alla volta. L'obiettivo considerato è sempre la minimizzazione del tempo finale di completamento (makespan).

Si tratta di un problema NP-hard, ma un semplice algoritmo goloso ottiene un fattore di approssimazione pari a 2. L'algoritmo è il seguente: ogni volta che una macchina si rende libera, non appena c'è una operazione che può essere iniziata su quella macchina, inizia l'operazione. Un'operazione può essere iniziata se il lavoro a cui appartiene non è in fase di elaborazione su un'altra macchina.

Indichiamo con P_j il tempo totale necessario per il lavoro J_j e con Π_i il tempo totale che deve essere speso sulla macchina M_i: $P_j = \sum_{i=1}^{m} p_{i,j}$, $\Pi_i = \sum_{j=1}^{n} p_{i,j}$. Notiamo che sia $\max_i \Pi_i$ che $\max_j P_j$ sono dei lower bound sul costo dell'ottimo; infatti, ogni lavoro deve essere elaborato completamente (per ogni j, $m^*(x) \geq P_j$) e ogni macchina deve elaborare tutte le operazioni destinate a quella macchina (per ogni i, $m^*(x) \geq \Pi_i$). Per mostrare la 2-approssimazione, quindi, basta dimostrare che il tempo di completamento dello schedule trovato dall'algoritmo è al più $\max_i \Pi_i + \max_j P_j$.

Consideriamo la macchina M_i che termina per ultima, e sia J_j il lavoro la cui operazione $O_{i,j}$ termina per ultima sulla macchina. Ora, ad ogni istante di tempo nello schedule, ci sono due possibilità: o il lavoro J_j è in fase di elaborazione su qualche altra macchina, oppure la macchina M_i è occupata (o entrambe le cose). Infatti, se entrambe le condizioni non fossero verificate, l'operazione $O_{i,j}$ sarebbe stata assegnata a M_i in un istante precedente. Per questo motivo, il tempo di completamento totale è pari al più a $\Pi_i + P_j \leq 2m^*(x)$.

Esercizio 5.1. Si consideri un'istanza con p macchine e $p+1$ lavori, ognuno costituito da un'operazione di lunghezza unitaria su ciascuna delle macchine. Qual è il makespan ottimo? Qual è il makespan ottenuto dall'algoritmo goloso nel caso in cui le p macchine spendano le prime p unità di tempo sui primi p lavori?