ALGORITMI E COMPLESSITÀ

Prof. Sebastiano Vigna
6 CFU

Luca Cappelletti

Lecture Notes
Year 2017/2018

Magistrale Informatica
Università di Milano
Italy
8 ottobre 2018

Indice

1 Notazioni **2**

2 Algoritmi di Approssimazione **3**

2.1 Bilanciamento di carico (load balancing) . . . 3

2.1.1 Algoritmo di List-Scheduling . . . 3

2.1.2 Regola del longest processing time (LPT) . . . 4

2.2 Problema della selezione del numero di centri . . . 5

2.2.1 Soluzione con algoritmo greedy . . . 5

1 Notazioni

Definizione 1.0.1 (O grande). Con notazione **O grande** si intende:

$$g(n) = O(f(n)) \quad \exists c \forall n \geqslant N \quad g(n) \leqslant c \cdot f(n)$$

Definizione 1.0.2 (Problema polinomiale). Un problema $P(n)$ è detto **polinomiale** se la sua complessità temporale è $O(P(n))$.

Definizione 1.0.3 (Non Polinomiale NP). Questa classe di problemi viene definita tramite un *verificatore* con un'instanza ed un *testimone.*

$$\forall x \in L \quad \exists \omega \in 2^* \text{ t.c. il verificatore risponde in tempo P. Si.}$$
$$\forall x \notin L \quad \forall \omega \in 2^* \text{ t.c. il verificatore risponde in tempo P. No.}$$

2

Algoritmi di Approssimazione

2.1 Bilanciamento di carico (load balancing)

Con **load balancing** si intende il problema di assegnare lavori ad ogni macchina in modo da minimizzare il **makespan**, cioè il carico massimo assegnato ad una data macchina. Ogni lavoro j deve lavorare continuamente su una macchina e una macchina può processare solo un lavoro per volta. Si tratta di un problema **NP-Difficile** persino con solo due macchine.

Definizione 2.1.1 (Carico). Sia $J(i)$ il sottoinsieme di lavori assegnati alla macchina i-esima. Allora il **carico** della i-esima macchina è pari a:

$$L_i = \sum_{j \in J(i)} t_j$$

Dove t_j è il tempo necessario per processare il lavoro j.

Definizione 2.1.2 (Makespan). Con **makespan** si intende il massimo carico su qualsiasi data macchina:

$$L = \max_i L_i$$

2.1.1 Algoritmo di List-Scheduling

Prendiamo in considerazione n lavori in un ordine fissato. Assegniamo il lavoro j-esimo alla macchina il cui carico è più piccolo sino ad ora.

Complessità computazionale 2.1.3 (List Scheduling). L'implementazione del List Scheduling ha complessità computazionale $O(n \log m)$ utilizzando una coda di priorità.

Analisi dell'algoritmo greedy proposto

Lemma 2.1.4. Il **makespan** ottimo L^* risulta maggiore o uguale del tempo di lavoro massimo.

$$L^* \geqslant \max_j t_j$$

Dimostrazione. Banalmente una macchina deve processare il lavoro che consuma più tempo. □

Lemma 2.1.5. Il **makespan** ottimo L^* risulta maggiore o uguale della media dei tempi di elaborazione.

$$L^* \geqslant \frac{1}{m} \sum_j t_j$$

Dimostrazione. Il tempo di processo totale è $T = \sum_j t_j$ ed una delle macchine deve fare certamente almeno $\frac{1}{m}$ del lavoro. □

Teorema 2.1.6 (Teorema di Graham (1966)). Un algoritmo greedy è una **2-approximation**, cioè l'ottimo locale individuato dall'algoritmo è al più due volte peggiore dell'ottimo globale.

Teorema di Graham. Prendiamo in considerazione il carico di strozzatura L_i della macchina i. Sia j l'ultimo lavoro pianificato sulla macchina i, allora, quando il lavoro j viene assegnato alla macchina i, questa aveva il carico minore. Il suo carico precedentemente all'assegnamento era pari a:

$$L_i - t_j \Rightarrow L_i - t_j \leqslant L_k \quad \forall i \leqslant k \leqslant m$$

Sommando le disuguaglianze su k ed applicando il lemma 2.1.5 otteniamo:

$$\begin{aligned} L_i - t_j &\leqslant \frac{1}{m}\sum_k L_k \\ &= \frac{1}{m}\sum_k t_k \\ &\leqslant \underbrace{L^*}_{\text{Applichiamo qui il lemma 2.1.5}} \end{aligned}$$

Quindi applicando il lemma 2.1.4:

$$L_i = \underbrace{L_i - t_j}_{\leqslant L^*} + \underbrace{t_j}_{\leqslant L^*} \leqslant 2L^*$$

□

2.1.2 Regola del longest processing time (LPT)

Ordina n lavori in ordine decrescente di tempo di elaborazione e quindi esegui usando questa lista l'algoritmo di List-Scheduling.
Osservazione 2.1.7. Se vi sono al più m lavori, allora l'algoritmo di list-scheduling è ottimo: banalmente si assegna ad ogni macchina un lavoro.

Lemma 2.1.8. Se esistono più lavori che macchine, allora:

$$L^* \geqslant 2t_{m+1}$$

Dimostrazione. Prendiamo in considerazione i primi $m+1$ lavori. Il tempo di elaborazione di ognuno è in ordine decrescente, per cui tutti i lavori richiedono un tempo di elaborazione maggiore del tempo t_{m+1} dell'ultimo lavoro considerato.

Se vi sono $m+1$ lavori ed m macchine, una macchina deve necessariamente dover compiere almeno due lavori. □

Teorema 2.1.9 (Approssimazione della regola LPT). La regola LPT è un algoritmo con una $\frac{3}{2}$-approximation.

Esiste un teorema più sofisticato che dimostra che l'algoritmo sia una $\frac{4}{3}$-approximation.

Dimostrazione. La dimostrazione è analoga a quella del teorema 2.1.6, ma nella conclusione viene sfruttato il lemma 2.1.8:

$$L_i = \underbrace{L_i - t_j}_{\leqslant L^*} + \underbrace{\leqslant \frac{1}{2}L^*} \leqslant \frac{3}{2}L^*$$

□

2.2 Problema della selezione del numero di centri

Il problema consiste nel scegliere k centri C così che la distanza massima $r(C)$ da un punto al centroide più vicino è minimizzata.

2.2.1 Soluzione con algoritmo greedy

Posizioniamo il primo centro nella migliore posizione possibile e continuiamo ad aggiungere centri sino a ridurre il raggio ogni volta quanto più possibile.

Ogni nuovo centroide è posizionato in modo da essere quanto più distante possibile da quelli preesistenti.

Proprietà 2.2.1 (Proprietà dell'algoritmo greedy). Alla terminazione, tutti i centroidi in C sono uno a uno distanti almeno quanto il raggio di copertura minimo $r(C)$.

Lemma 2.2.2. Sia C^* l'insieme ottimale dei centroidi. Allora vale che:

$$r(C) \leq 2r\left(C^*\right)$$

Dimostrazione. Procediamo per assurdo e assumiamo che $r(C^*) < \frac{1}{2}r(C)$.

Per ogni centroide $c_i \in C$, consideriamone la sfera di raggio $\frac{1}{2}r(C)$ attorno. All'interno di ogni sfera esiste esattamente un $c_i^* \in C^*$, che possiamo considerare accoppiato con c_i.

Prendiamo ora in considerazione un punto s qualsiasi ed il suo centroide ottimale più vicino, c_i^*. Vale che:

$$\text{distanza}(s, C) \leq \text{distanza}(s, c_i) \underbrace{\leq}_{\text{dis. triangolare}} \underbrace{\text{distanza}(s, c_i^*) + \text{distanza}(c_i, c_i^*)}_{\leq r(C^*) \text{ dato che } c_i^* \text{ è il centroide più vicino.}} \leq 2r(C^*)$$

Ne segue che $r(C) \leq 2r(C^*)$. □

Teorema 2.2.3. Un algoritmo greedy è una 2-*approximation* per il problema della scelta dei centroidi.

Teorema 2.2.4. A meno che **P=NP**, non esiste nessuna ρ-*approximation* per la selezione dei centroidi che abbia $\rho < 2$.

Dimostrazione. Procediamo per assurdo assumendo che esista una $(2-\epsilon)$-*approximation* con cui possiamo risolvere in tempo polinomiale il set dei punti S.

Sia $G = (V, E)$, $k \in \mathbb{N}$ un'istanza di S. Procediamo a costruire un'istanza G' del problema, con V punti e distanze pari a:

$$\begin{cases} \text{distanza}(u, v) = 1 & \text{se } u, v \in E \\ \text{distanza}(u, v) = 2 & \text{se } u, v \notin E \end{cases}$$

L'istanza così costruita rispetta la disuguaglianza triangolare.

G' possiede un insieme dominante di dimensione k se e solo se esistono k centroidi C^* con $r(C^*) = 1$.

Ne segue che se G' ha un insieme dominante di dimensione k, un algoritmo con $(2-\epsilon)$-*approximation* troverebbe una soluzione C^* con $r(C^*) = 1$, dato che non può utilizzare nessun arco a distanza 2. □