New Online Algorithms for Story Scheduling in Web Advertising

1 Introduzione

L'ambito di studio è lo storyboarding in cui gli advertisers desiderano presentare sequenze di ads (storie) senza interruzioni su una posizione di rilievo di una pagina web. Queste storie/jobs arrivano "online" e vengono attivate in base alla cronologia di navigazione dell'utente che, in ogni momento, continua a navigare con probabilità β. Nello storyboarding un singolo advertiser controlla una posizione pubblicitaria di rilievo per un certo continuativo periodo di tempo. L'advertiser crea una storia pubblicizzando i propri prodotti e la mostra utilizzando gli slot di tempo a lui concessi. Tipicamente molti advertiser competono per tale posizione. L'obiettivo di un ad server è quello di costruire uno schedule che massimizzi il risultato aspettato.

Formalmente lo storyboarding è stato formulato come un problema di scheduling di jobs online da Dasgupta, Ghosh, Nazerzadeh e Raghavan in una precedente pubblicazione [1]. Consideriamo un utente che al tempo $t = 0$ inizia una sessione web. Il tempo è "slotted". In qualsiasi momento l'utente continua a navigare con probabilità β e smette di navigare con probabilità $1 - \beta$ con $0 < \beta \leq 1$. Quindi il tempo di navigazione è una variabile casuale geometrica (distribuzione geometrica). Nel tempo le storie (jobs) arrivano online. Ogni job i è specificato da un tempo di arrivo a_i, una lunghezza l_i, che rappresenta la lunghezza della storia, e un valore per unità v_i, che rappresenta il premio ottenuto dal server mostrando un unità del job i. Considerando tutti i jobs in arrivo si ottiene un'istanza del problema $I = (a_i, v_i, l_i)_{i=1}^{N}$ con $N \in \mathbb{N} \cup \{\infty\}$

Uno schedule S per I specifica quali job processare ad ogni tempo $t \geq 0$. S può anche non contenere tutti i job di I, è infatti permesso tralasciare dei job non desiderati. S è ben definito se ogni job schedulato è processato a tempi $t \geq a_i$ per al massimo l_i unità di tempo. Inoltre il pre-rilascio dei jobs è permesso, cioè un job i può essere processato per meno di l_i unità di tempo. In questo caso non si può ottenere valore per la porzione pre-rilasciata non schedulata di un job. Dato uno scheduler S, il suo valore è definito come il valore atteso $\sum_{t=0}^{\infty} \beta^t \cdot v(t)$ dove $v(t)$ è il valore per unità del job schedulato al tempo t. Sia ALG un algoritmo online che, dato qualsiasi input I, costruisca uno schedule di valore $ALG(I)$. Sia $OPT(I)$ il valore di uno schedule ottimale offline per I. Si dice che ALG è c-competitivo se esiste una costante α tale che $c \cdot ALG(I) + \alpha \geq OPT(I)$, $\forall I$ [2].

2 Lavori precedenti

Secondo questa pubblicazione, lo storyboarding è stato studiato, da un punto di vista algoritmico, solamente da Dasgupta et al. [1]. Una prima osservazione è che se $\beta = 1$ allora il problema è semplice da risolvere in quanto il valore dello schedule non dipende dal tempo in cui vengono eseguiti i vari job. Quindi ci si concentra nel caso in cui $0 < \beta < 1$. Dasgupta et al. [1] hanno mostrato che nessun algoritmo deterministico online può avere un rateo competitivo di $\beta+\beta^2$ che quindi è limitato superiormente da 2. Come risultato principale Dasgupta et al. hanno sviluppato un algoritmo greedy che è 7-competitivo. Inoltre hanno affrontato un'estensione del problema in cui i jobs hanno un valore per unità crescente anziché costante e in cui il valore è ottenuto solo quando il job è completamente finito. Infine hanno studiato un'altra estensione del problema in cui un job dev'essere schedulato immediatamente all'arrivo, altrimenti è perso.

3 Contributi di questo articolo

Gli autori di questa pubblicazione hanno presentato tre nuovi algoritmi online per lo storyboarding. Tutte le strategie seguono il paradigma del processare una data sequenza di job I in fasi, in cui una fase consiste in k intervalli di tempi consecutivi per qualche $k \in \mathbb{N}$. Sia $k \geq 1$ un intero, una k-fase consiste in k intervalli di tempo consecutivi. Nello specifico l'n-esima k-fase P_n è la sottosequenza di intervalli di tempo $(n-1)k, \ldots, nk-1$, per ogni $n \geq 1$. All'inizio di ogni fase un algoritmo crea uno schedule per la fase, ignorando i job che potrebbero arrivare durante la fase. In questo modo le decisioni di scheduling vengono prese una volta ogni tanto.

Il primo algoritmo presentato crea uno schedule ottimale per ogni fase e pre-rilascia i jobs che non hanno finito la loro esecuzione alla fine della rispettiva fase. Viene dimostrato che il rateo competitivo di questa strategia è esattamente $1/\beta^{k-1} \cdot (1-\beta^k), \forall\beta$ e $\forall k \in \mathbb{N}$. Inoltre con la migliore scelta di k si ottiene una competitività di $4/(2-\beta)$ che è limitata superiormente da 4, $\forall\beta$ e $\forall k \in \mathbb{N}$. Infine se scegliamo $k = 1$, l'algoritmo risultante è $1/(1-\beta)$ competitivo e tale scelta permette di ottenere migliori limitazioni della competitività per β piccoli, cioè per $\beta < 2/3$.

Il secondo algoritmo presentato è una raffinazione del primo algoritmo che preferisce non pre-rilasciare i jobs che non hanno terminato alla fine della rispettiva fase ma invece prova a rieseguirli nella fase successiva. Il rateo competitivo di questa strategia è limitato superiormente da $1/\beta^{k-1} \cdot max\{1/\beta^{k-1}, 1/(1-\beta^{2k}), 1+\beta^{3k}/(1-\beta^k)\}$. Inoltre con la migliore scelta di k, otteniamo un fattore competitivo di $c = 1+\phi$, dove $\phi = (1+\sqrt{5})/2$ è la sezione aurea. Si ottiene quindi $c \approx 2.618$ che è molto vicino al limite teorico di 2 presentato da Dasgupta et al. [1] per un β generico.

Il terzo algoritmo presentato è una generalizzazione del primo in cui si presuppone che ci siano m macchine parallele (ad positions) disponibili. In questo caso il valore dello schedule creato è dato da $\sum_{t=0}^{\infty}\sum_{j=1}^{m}\beta^t \cdot v(t,j)$, dove $v(t,j)$ è il valore unitario del job schedulato sulla macchina j al tempo t. Viene dimostrato che, per la migliore scelta di k, il rateo competitivo di questa strategia è $(1+1/(1-\beta(2-\sqrt{2}))/(2-\sqrt{2})$. Il rateo è limitato inferiormente da $2/(2-\sqrt{2}) \approx 3.414$ e limitato superiormente da $1/(3-2\sqrt{2}) \approx 5.828$.

4 $ALG1_k$

Il primo algoritmo, chiamato $ALG1_k$, crea uno schedule ottimale per ogni k-fase P_n. Formalmente $ALG1_k$ funziona come segue. Diciamo che un job i è disponibile al tempo t se il job è arrivato entro t, cioè $a_i \leq t$ e non è stato mai schedulato ad un tempo qualsiasi $t' < t$. Sia Q_n l'insieme dei jobs disponibili all'inizio di P_n, cioè tutti quei job i tali che $a_i \leq (n-1)k$. $ALG1_k$ ordina i job di Q_n in ordine non crescente di valore unitario. I job che hanno lo stesso valore unitario vengono ordinati in ordine crescente di tempo di arrivo. A questo punto $ALG1_k$ assegna ad uno ad uno i job ordinati a P_n, finché tutti i k intervalli di tempo vengono schedulati oppure la sequenza di jobs finisce. Se l'ultimo job schedulato non termina al termine di P_n, viene pre-abbandonato. $ALG1_k$ esegue questo schedule per P_n, ignorando i jobs che potrebbero arrivare durante la fase a tempi $t = (n-1)k+1, \ldots, nk-1$. Per prima cosa valutiamo le prestazioni di $ALG1_k$, per un k generico. Poi determineremo la migliore scelta di k.

Teorema 1 *$\forall k \in \mathbb{N}$ e $\forall \beta$, $ALG1_k$ è $1/(\beta^{k-1}(1-\beta^k))$-competitivo.*

Di seguito costruiremo le basi per arrivare a dimostrare il precedente teorema. Sia $I = (a_i, v_i, l_i)_{i=1}^N$ un input arbitrario. Nell'elaborare I, $ALG1_k$ rimanda i job che arrivano dopo l'inizio di una fase fino all'inizio della fase successiva. Si consideri un input k-quantizzato I_k in cui il tempo di arrivo di qualsiasi job è impostato al successivo multiplo intero di k, cioè $I_k = (a'_i, v_i, l_i)_{i=1}^N$, dove $a'_i = k\lceil a_i/k \rceil$. Se a_i è un multiplo di k e quindi coincide con l'inizio di una k-fase allora il job non viene ritardato. Altrimenti il job viene ritardato fino all'inizio della fase successiva. Lo schedule generato da $ALG1_k$ per I_k è identico a quello calcolato da $ALG1_k$ per I. Quindi $ALG1_k(I_k) = ALG1_k(I)$. Al fine di dimostrare il Teorema 1 sarà conveniente confrontare $ALG1_k(I_k)$ con $OPT(I_k)$. Il lemma successivo assicura che $OPT(I_k)$ e il vero valore ottimale $OPT(I)$ differiscono al massimo di un fattore $1/\beta^{k-1}$.

Lemma 1 *$\forall k \in \mathbb{N}$ e $\forall \beta$, vale la disuguaglianza $1/\beta^{k-1} \cdot OPT(I_k) \geq OPT(I)$.*

Dimostrazione Consideriamo uno schedule ottimale per I e spostiamo l'intero schedule di $k-1$ unità di tempo a destra, cioè il tempo di inizio di ogni lavoro è ritardato esattamente di $k-1$ unità di tempo. Lo schedule così modificato è corretto per I_k perché, per qualsiasi job i, il suo tempo di arrivo a'_i in I_k è ritardato di al massimo $k-1$ unità di tempo rispetto al corrispettivo a_i in I. Lo schedule modificato ha un valore di $\beta^k \cdot OPT(I)$, e uno schedule ottimale per I_k ottiene un valore tanto alto almeno quanto il precedente. □

Per stimare OPT(I_k) consideriamo un algoritmo offline ottimale più forte che è stato proposto anche da Dasgupta et al. [1]. Questo algoritmo permette di riprendere l'esecuzione dei job interrotti in un momento successivo. Chiamiamo questa strategia offline $CHOP$. Per qualsiasi input, in qualsiasi istante t, $CHOP$ schedula un job che ha il più alto valore per unità tra i job non completati che sono arrivati entro il tempo t. Ovviamente, $CHOP(I_k) \geq OPT(I_k)$. Sia S lo schedule calcolato da $ALG1_k$ per I_k e sia S^* lo schedule generato da $CHOP$ per I_k . Assumiamo senza perdità di generalità che in S^* tutti i job che hanno un certo valore unitario v siano elaborati nello stesso ordine in cui sono processati

in S. Più precisamente, tutti i job che hanno valore unitario v sono elaborati in ordine crescente di tempi di arrivo. I job di valore unitario v che arrivano nello stesso momento vengono elaborati nello stesso ordine in cui vengono processati in S. Lo schedule S^* può essere facilmente modificato in modo tale da soddisfare questa proprietà. Per qualsiasi job i, diciamo che $t_S(i)$ indica il suo tempo di inizio in S e che $t_{S^*}(i)$ indica il suo tempo di inizio in S^*. Se il job i non viene mai processato in S (o S^*), allora impostiamo $t_S(i) = \infty$ (o $t_{S^*}(i) = \infty$). Il seguente lemma afferma che $ALG1_k$ avvia ogni job prima o allo stesso momento rispetto a $CHOP$.

Lemma 2 *Per qualsiasi job i, $t_S(i) \leq t_{S^*}(i)$.*

Dimostrazione La disuguaglianza vale ovviamente per i job che non sono mai stati schedulati da $CHOP$. Supponiamo che il lemma non valga per ogni job e sia i quello che si trova per primo in S^* con $t_{S^*}(i) < t_S(i)$. Sia $t^* = t_{S^*}(i)$ e sia P_n la fase contenente t^*. Inoltre, sia il job j quello schedulato da $ALG1_k$ al tempo t^*. In I_k i job arrivano solo all'inizio di una fase, ovvero quando $ALG1_k$ prende le decisioni di scheduling. Quindi all'inizio di P_n il job i è arrivato e può essere schedulato da $ALG1_k$. Dal momento che $ALG1_k$ ordina i job disponibili in ordine di valore unitario non crescente e non esegue il job i prima o allo stesso tempo di t^*, allora vale $v_j \geq v_i$.

Dimostriamo adesso che al tempo t^*, $CHOP$ ha già terminato il job j. Questo vale chiaramente se $v_j > v_i$ perché $CHOP$ schedula sempre un job non terminato con il valore unitario più alto. Se $v_j = v_i$, allora di nuovo $CHOP$ deve aver già completato il job j perché in S^* i job di valore unitario $v = v_j = v_i$ si verificano nello stesso ordine come in S e il job j precede il job i in S.

Poiché $CHOP$ al tempo t^* ha già terminato il job j, allora il tempo di inizio di j è minore o uguale a $t - l_j$. D'altra parte $ALG1_k$ non ha avviato il job j prima del tempo $t - l_j + 1$ perché lo sta ancora elaborando. Concludiamo che $t_{S^*}(j) < t_S(j) \leq t_{S^*}(i)$, il che contraddice l'assunzione che il job i sia il primo in S^* violando la disuguaglianza desiderata. □

Il lemma successivo mette in relazione il valore di $ALG1_k$ con quello di OPT, ottenuti con input I_k .

Lemma 3 *$\forall k \in \mathbb{N}$ e $\forall \beta$, vale la disuguaglianza $1/(1-\beta^k) \cdot ALG1_k(I_k) \geq OPT(I_k)$.*

Dimostrazione Per qualsiasi $n \geq 1$, sia I_n l'insieme dei job schedulati da $ALG1_k$ durante la fase P_n, cioè, formalmente

$$I_n = \{i \mid (n-1)k \leq t_S(i) \leq nk - 1\}.$$

Sia $ALG1_k(P_n)$ il valore ottenuto da $ALG1_k$ nello scheduling dei job di I_n e sia $CHOP(P_n)$ il valore ottenuto da $CHOP$ nell'elaborazione di questi job. Vale quindi $ALG1_k(I_k) = \sum_n ALG1_k(P_n)$. Una conseguenza del Lemma 2 è che tutti i job che sono stati schedulati da $CHOP$ sono presenti anche nello schedule di $ALG1_k$. Quindi $CHOP(I_k) = \sum_n CHOP(P_n)$. Mostreremo che, $\forall n \in \mathbb{N}$, vale la disuguaglianza $CHOP(P_n)/ALG1_k(P_n) \leq 1/(1-\beta^k)$. Questo implica che $CHOP(I_k)/ALG1_k(I_k) \leq 1/(1-\beta^k)$ e il lemma poi segue in quanto $CHOP(I_k) \geq OPT(I_k)$.

Consideriamo una qualsiasi k-fase P_n. Nello schedule S sia j l'ultimo job iniziato in P_n e sia λ_j il numero di unità di tempo per le quali j è schedulato in P_n e quindi nell'intero schedule S. Dal Lemma 2, per qualsiasi job i, vale $t_S(i) \leq t_{S^*}(i)$. Quindi il valore totale ottenuto da $CHOP$ nello schedulare i job $i \in I_n$ con $i \neq j$, così come le prime λ_j unità di tempo del job j non può essere superiore ad $ALG1_k(P_n)$.

Se il job j viene pre-abbandonato in S alla fine di P_n, allora $CHOP$ può ottenere un ulteriore valore aggiuntivo nello schedulare le rimanenti unità di tempo $\lambda_{j+1}, \ldots, l_j$ del job j in S. Anche in questo caso, poiché $t_S(j) \leq t_{S^*}(j)$, queste unità non possono essere sequenziate prima dell'inizio della fase P_{n+1}, cioè al tempo nk. Quindi il valore addizionale ottenibile per le unità $\lambda_{j+1}, \ldots, l_j$ è limitato superiormente da $\sum_{t=nk}^{\infty} \beta^t v_j = \beta^{nk}/(1-\beta) \cdot v_j$, che è ottenuto schedulando un job di valore unitario v_j di infinita lunghezza a partire dal tempo nk.

Quindi $CHOP(P_n) \leq ALG1_k(P_n) + \beta^{nk}/(1-\beta) \cdot v_j$. In ciascuna fase $ALG1_k$ ordina i job in ordine di valore unitario non crescente. Di conseguenza ogni job di I_n ha un valore unitario di almeno v_j. Concludiamo che valgono $ALG1_k(P_n) \geq \sum_{t=(n-1)k}^{nk-1} \beta^t v_j = (\beta^{(n-1)k} - \beta^{nk})/(1-\beta) \cdot v_j$ e $CHOP(Pn)/ALG1_k(P_n) \leq 1 + \beta^{nk}/(\beta^{(n-1)k} - \beta^{nk}) = 1/(1-\beta^k)$. □

Adesso possiamo dimostrare il Teorema 1.

Dimostrazione del Teorema 1 Combinando i lemmi 1 e 3 si ottiene che, $\forall k \in \mathbb{N}$ e $\forall \beta$, vale la disuguaglianza $1/(\beta^{k-1}(1-\beta^k))ALG1_k(I_k) \geq OPT(I)$ per ogni input I. Il teorema segue poiché vale $ALG1_k(I) = ALG1_k(I_k)$. □

Determiniamo il miglior valore di k.

Corollario 1 *Per $k = \lceil -log_\beta 2 \rceil$, l'algoritmo risultante $ALG1_k$ è $4/(2-\beta)$-competitivo.*

Dimostrazione La funzione $f(x) = \beta^{x-1}(1-\beta^x)$ è massimizzata per $x^* := -log_\beta 2$. Scegliendo $k = \lceil -log_\beta 2 \rceil$ si ottiene che $x^* \leq k \leq x^* + 1$ e $f(x^*) \geq f(k) \geq (2-\beta)/4 = f(x^*+1)$ in quanto $f(x)$ è strettamente decrescente per $x > x^*$. Il corollario è dimostrato, in quanto la competitività di $ALG1_k$ è $1/f(k)$. □

Il teorema che segue dimostra che il rapporto competitivo di $ALG1_k$ mostrato sopra non può essere più piccolo.

Teorema 2 *$\forall k \in \mathbb{N}$ e $\forall \beta$, il rapporto competitivo di $ALG1_k$ non è inferiore a $1/(\beta^{k-1}(1-\beta^k))$.*

Dimostrazione Supponiamo che $ALG1_k$ abbia ottenuto un rapporto competitivo $c < 1/(\beta^{k-1}(1-\beta^k))$. Esiste quindi una costante α tale che vale $c \cdot ALG1_k(I) + \alpha \geq OPT(I)$ per ogni input I. Consideriamo un input specifico I costituito da un singolo job che arriva al tempo 1, che ha un valore di $v = \alpha/(\beta(1-c\beta^{k-1}(1-\beta^k))$ e lunghezza infinita. $ALG1_k$ avvia questo job al tempo k e lo elabora per k unità di tempo in modo che valga $ALG1_k(I) = \beta^k(1-\beta^k)/(1-\beta) \cdot v$. D'altra parte $OPT(I) = \beta/(1-\beta) \cdot v$. Quindi

$$c \cdot ALG1_k(I) + \alpha = c \cdot ALG1_k(I) + (\alpha/v)v = c \cdot ALG1_k(I) + \beta(1 - c\beta^{k-1}(1-\beta^k))v$$

$$< c \cdot ALG1_k(I) + \frac{\beta}{1-\beta}(1 - c\beta^{k-1}(1-\beta^k))v = \frac{\beta}{1-\beta} \cdot v = OPT(I),$$

dove la disuguaglianza vale in quanto $1 - \beta < 1$. Abbiamo ottenuto una contraddizione. □

Infine in questa sezione consideriamo l'algoritmo $ALG1_1$ in cui la lunghezza della fase k è uguale a 1. Questo algoritmo schedula in qualsiasi momento un job con il valore per unità più elevato tra i job disponibili. Questo job viene elaborato per una sola unità di tempo.

Corollario 2 *$\forall\beta$, il rapporto competitivo di $ALG1_1$ è esattamente $1/(1-\beta)$.*

Dimostrazione Il teorema 1 implica che $ALG1_1$ è $1/(1-\beta)$-competitivo. Inoltre per il Teorema 2 tale rapporto competitivo non può essere più piccolo. □

Combiniamo infine i Corollari 1 e 2 per ottenere il seguente risultato.

Corollario 3 *Impostando $k = 1$ se $\beta \leq 2/3$ e $k = \lceil -log_\beta 2 \rceil$ altrimenti, otteniamo un algoritmo $ALG1_k$ che ottiene un rapporto competitivo di $min\{1/(1-\beta), 4/(2-\beta)\}$.*

Dimostrazione Osserviamo che $1/(1-\beta) \leq 4/(2-\beta)$ è vero se e solo se $\beta \leq 2/3$. Supponiamo che $\beta \leq 2/3$. In questo caso l'algoritmo risultante $ALG1_1$ ottiente un rapporto competitivo di $1/(1-\beta) \leq min\{1/(1-\beta), 4/(2-\beta)\}$, cfr. Corollario 2. Adesso supponiamo che $\beta > 2/3$. Per il Corollario 1, l'algoritmo $ALG1_k$ con $k = \lceil -log_\beta 2 \rceil$ raggiunge una competitività di $4/(2-\beta) < min\{1/(1-\beta), 4/(2-\beta)\}$. □

5 $ALG2_k$

Il secondo algoritmo, chiamato $ALG2_k$, è una raffinazione di $ALG1_k$; il principio di funzionamento è praticamente lo stesso. L'unica differenza sta nel fatto che tenta di ripristinare l'esecuzione di un eventuale job in una fase P_n se è stato pre-rilasciato nella fase immediatamente precedente P_{n-1}. Formalmente $ALG2_k$ funziona come segue. Se $n > 1$, sia i il job che è stato pre-rilasciato nella fase P_{n-1}. A questo punto definiamo il job residuo $r = (a_i, v_i, l_r)$; l_r è la parte rimanente del job i che dev'essere processata. Aggiungiamo r a Q_n. $ALG2_k$ ordina e schedula i job di Q_n nello stesso modo di $ALG1_k$. Sia $S'(P_n)$ lo schedule ottenuto in questo modo. Se $S'(P_n)$ non contiene r allora $S'(P_n)$ è lo schedule finale per la fase. Altrimenti, se $S'(P_n)$ contiene r e tale job è schedulato per s unità di tempo al tempo t, allora $ALG2_k$ modifica $S'(P_n)$ in modo da spostare il job r all'inizio della fase. Nello specifico il tempo di inizio di ogni job appartenente a $S'(P_n)$ e diverso da r, viene aumentato di s unità di tempo, mentre il tempo di inizio di r diventa $t = (n-1)k$. Questo è lo schedule finale di $ALG2_k$ per P_n. Teorema 3 $\forall k \in \mathbb{N}$ e $\forall\beta$, l'algoritmo $ALG2_k$ raggiunge un rapporto competitivo di $1/\beta^{k-1} \cdot max\{1/\beta^{k-1}, 1/(1-\beta^{2k}), 1+\beta^{3k}/(1-\beta^k)\}$. Costruiamo le basi per poter dimostrare il Teorema 3. Rispetto alla dimostrazione del Teorema 1, l'analisi è più complessa perché dobbiamo occuparci dei ritardi subiti da ALG2k nello Step(2) quando si schedula una porzione di job i_n all'inizio della fase P_n e posticipando così l'inizio dei job con valori unitari più alti. Inoltre, al fine di raggiungere un piccolo rapporto competitivo, dobbiamo addebitare

la perdita di un job pre-abbandonato in una fase a una o più fasi adiacenti. Ancora una volta, per ogni input $I = (a_i, vi, l_i)_{i=1}^N$, consideriamo l'input k-quantizzato $I_k = (a_i, v_i, l_i)_{i=1}^N$, dove il tempo di arrivo di qualsiasi job i è $a_i = k\lceil a_i/k \rceil$. Vale $ALG2_k(I_k) = ALG2_k(I)$ e, come mostrato nel Lemma 1, $1/\beta^{k-1}OPT(I_k) \geq OPT(I)$.Confronteremo $ALG2_k(I_k)$ con $CHOP(I_k)$, dove $CHOP$ è l'algoritmo offline ottimale descritto in Sez. 3. Di nuovo, sia S lo schedule calcolato da $ALG2_k$ per I_k e sia S^* lo schedule di CHOP per I_k. Analogamente a quanto detto nella Sez. 3 supponiamo, senza perdita di generalità, che in S^* tutti i job con un determinato valore unitario v siano elaborati nello stesso ordine in cui sono processati in S. Al fine di valutare $ALG2_k(I_k)$, definiamo uno schedule S' che ci consente di provare un'affermazione analoga al Lemma 2 e, inoltre, di confrontare i valori unitari dei job schedulati in S' e S^*. Per qualsiasi fase P_n, consideriamo lo schedule $S'(P_n)$ calcolato durante lo Step(1) di $ALG2_k$. Se $n > 1$ e il job residuo i_n^r è schedulato per s_n^r unità di tempo a partire dal tempo t_n^r in P_n, allora modifichiamo $S'(P_n)$ schedulando il job originale i_n per s_n^r unità di tempo a partire dal tempo t_n^r. Da adesso in poi chiameremo questo schedule modificato come $S'(P_n)$. Lo schedule S è la concatenazione di $S(P_n)$, per ogni $n \geq 1$. In $S'(P_n)$ i job sono sequenziati in ordine di valore unitario non crescente. Tra i job di valore unitario $v = v_{i_n}$, il job i_n viene elaborato per primo. Lo schedule $S'(P_n)$ differisce da $S(P_n)$ solamente per il job i_n, in quanto in $S'(P_n)$ viene sequenziato dopo quei job che hanno un valore unitario strettamente superiore a v_{i_n}. Ciascuno di questi job inizia e termina in P_n. Lo spostamento della porzione di job di i_n non influisce sul relativo ordine dei job che hanno lo stesso valore per unità. Quindi in S' e S, e quindi in S' e S^*, i job con un determinato valore unitario v occorrono nello stesso ordine. Notiamo che lo schedule S' non è corretto in quanto un job i_n può essere interrotto alla fine della fase P_{n-1} e ripreso più tardi in P_n. Per qualsiasi job i, sia $t_{S'}(i)$ il suo tempo di inizio in S'. Come al solito $t_S(i)$ e $t_{S^*}(i)$ denotano il tempo di inizio del job i rispettivamente in S e S^*. I job che non compaiono mai in uno schedule hanno un tempo di inizio infinito. In seguito, con il Lemma 5, proveremo un'affermazione corrispondente a quella di Lemma 2: per ogni job i, esiste $t_{S'}(i) \leq t_{S^*}(i)$. Per provare questo lemma abbiamo bisogno del seguente lemma ausiliario che implica, in particolare, che ogni job venga interrotto al massimo una volta in S'. Il lemma sarà anche essenziale nella dimostrazione di Lemma 6. Lemma 4 Se un job viene interrotto in S', allora questa interruzione si verifica alla fine di una fase P_{n-1} e il job è uguale a i_n, cioè quello elaborato per ultimo in $S(P_{n-1})$ e $S'(P_{n-1})$. Il job è schedulato di nuovo solo in P_n e non si verificano ulteriori interruzioni in S'. Dimostrazione Consideriamo lo schedule S e successivamente sostituiamo $S(P_n)$ con $S'(P_n)$, per numeri di fase n crescenti. Identifichiamo le interruzioni introdotte da queste sostituzioni. Come accennato in precedenza, $S'(P_n)$ è uguale a $S(P_n)$ tranne che per il job i_n, se esso è presente in $S(P_n)$; viene infatti sequenziato dopo quei job che hanno un valore unitario superiore a v_{i_n}. Tutti questi job iniziano e finiscono in $S(P_n)$ e quindi non vengono interrotti in $S'(P_n)$. Pertanto, quando si sostituisce $S(P_n)$ con $S'(P_n)$, solo il job i_n può essere interrotto e in questo caso il job i_n è schedulato per ultimo in $S(P_{n-1})$ e in $S'(P_{n-1})$. Se il job i_n viene effettivamente interrotto alla fine di $S'(P_{n-1})$ e successivamente continuato in $S(P_n)$, allora non viene elaborato per tutta la fase P_n. Quindi il job i_n in $S(P_n)$ non viene elaborato fino alla fine di P_n e non può essere elaborato ulteriormente in alcuna fase successiva. Poiché S' e S mettono in sequenza lo stesso insieme di job all'interno di ciascuna fase, segue il lemma. Lemma 5 Per ogni job i, $t_{S'}(i) \leq t_{S^*}(i)$. Dimostrazione La dimostrazione del lemma è simile a quella del Lemma 2. Tuttavia, qui dobbiamo distinguere

i casi a seconda che il job i sia interrotto o meno in S'. Per i job che non sono mai schedulati da $CHOP$ non c'è nulla da mostrare. Supponiamo che la disuguaglianza desiderata non valga per tutti i job e sia il job i quello che si presenta per primo in S^* con $t_{S'}(i) > t_{S^*}(i)$. Sia $t^* = t_{S^*}(i)$ e siache P_n la fase contenente il tempo t^*. Sia j il job elaborato in S' al tempo t^*. Vale $v_j \geq v_i$ in quanto il job i è arrivato entro $(n-1)t$ (cioè l'inizio di P_n), e in S' i job sono schedulati in ordine di valore unitario non crescente all'interno di ciascuna fase. Adesso dimostriamo che $CHOP$ termina il job j prima del tempo t^*. Ciò vale ovviamente se $v_j > v_i$ perché $CHOP$ processa sempre un job incompiuto con il valore unitario più alto. Nel caso in cui $v_j = v_i$, allora $CHOP$ deve aver terminato il job j perché in S' il job j si trova prima del job i e i job con lo stesso valore unitario vengano elaborati nello stesso ordine in S' e in S^*. Adesso distinguiamo due casi. Se il job j non è stato interrotto in S' prima del tempo t^*, abbiamo che $t_{S'}(j) \geq t^* - l_j + 1$. D'altra parte, $t_{S^*}(j) \leq t^* - l_j$ perché $CHOP$ ha terminato il job j prima tempo t^*. Otteniamo $t_{S^*}(j) < t_{S'}(j) \leq t^*$, che è una contraddizione dell'ipotesi che afferma che il job i sia il primo in S^* violando la disuguaglianza desiderata.

Se il job j è stato interrotto in S' prima del tempo t^*, allora per il Lemma 4 è uguale al job processato alla fine di P_{n-1}. Sia J l'insieme di job diversi da j e schedulati tra l'inizio di P_n e t^* in S'. Tutti questi job iniziano e finiscono in P_n e hanno un valore per unità strettamente superiore al job j. Sia l la lunghezza totale dei job in Z. Per il Lemma 4, il job j è stato interrotto una sola volta e quindi $t_{S'}(j) \geq t^* - l_j - l + 1$. Poiché tutti i job di J hanno un valore unitario superiore a v_j e $v_j \geq v_i$, allora $CHOP$ deve averli terminati tutti prima del tempo t^*. Pertanto $t_{S^*}(j) \leq t^* - l_j - l$ poiché, per ipotesi, $t_{S'}(i') \leq t_{S^*}(i')$ vale $\forall i' \in J$. Concludiamo di nuovo che $t_{S^*}(j) < t_{S'}(j) \leq t^*$ e otteniamo una contraddizione con la nostra ipotesi iniziale.

Uno degli obiettivi principali dell'analisi che seguirà è quello di definire un limite della perdita subita da $ALG2_k$ nel pre-abbandonare i job. Il Lemma 7 sarà fondamentale in quanto specifica il primo istante in cui un job pre-abbandonato in S può ripetersi in S^*. La dimostrazione si basa sul Lemma 6 che confronta i valori unitari dei job schedulati in S e S'. A qualsiasi tempo t, sia $v_{S^*}(t)$ il valore unitario del job schedulato in S^* e sia $v_{S'}(t)$ sia il valore unitario del job schedulato in S' . Se all'istante t nessun job è schedulato in S' o in S^*, il valore corrispondente $v_{S'}(t)$ o $v_{S^*}(t)$ è zero.

Lemma 6 Per ogni tempo t, $v_{S^*}(t) \geq v_{S'}(t)$. Dimostrazione Consideriamo un qualsiasi tempo t. Se nessun job è schedulato al tempo t in S', non c'è nulla da mostrare. Altrimenti sia i il job schedulato in S' al tempo t. Dal Lemma 5 vale $t_{S'}(i) \leq t_{S^*}(i)$. Se il job i non viene interrotto in S', allora $t_{S'}(i) \geq t - l_i + 1$ e quindi $t_{S^*}(i) \geq t - l_i + 1$. Pertanto $CHOP$ non termina il job i prima del tempo t ed processa un job con valore unitario di almeno v_i al tempo t in S^*. Se il job i viene interrotto in S', allora sia P_n sia la fase contenente t. Per il Lemma 4, il job i è uguale al job i_n processato per ultimo in $S'(P_{n-1})$. Sia J l'insieme di job $j \neq i$ processati tra l'inizio di P_n e il tempo t in S'. Tutti questi job iniziano in $S'(P_n)$. Sia l la lunghezza totale dei job in J. Poiché il job i viene interrotto una sola volta (vedere di nuovo Lemma 4) $t_{S'}(i) \geq t - l_i - l + 1$. Utilizzando il Lemma 5 otteniamo che $t_{S^*}(i) \geq t - l_i - l + 1$ e $t_{S'}(j) \leq t_{S^*}(j)$, $\forall j \in J$. Ciò implica che $CHOP$ non può finire tutti i job di $J \cup \{i\}$ prima del tempo t . Per la definizione di S', i job vengono elaborati in ordine non crescente di valore unitario in ciascuna fase. Tra i job di valore v_{i_n}, il job i_n viene processato per primo. Pertanto tutti i job di J hanno un valore unitario superiore a v_i. Concludiamo dicendo che al tempo t $CHOP$ processa un job con valore unitario di almeno

v_i.

Lemma 7 Se il job i viene pre-abbandonato in $S(P_n)$ e i seguenti schedule di fase $S(P_{n+1}), ..., S(P_{n'})$ processano solo job di valore unitario superiore a v_i, allora S^* non schedula il job i nelle fasi $P_{n+1}, ..., P_{n'}$. Dimostrazione Per qualsiasi fase P_l, gli schedule $S(P_l)$ e $S'(P_l)$ elaborano lo stesso insieme di job; ovviamente alcuni job potrebbero essere elaborati solo parzialmente in tali schedule. Quindi $S'(P_{n+1}), ..., S'(P_{n'})$ elaborano solo job con un valore unitario superiore a v_i. Per il Lemma 6, in qualsiasi istante lo schedule S^* elabora un job il cui valore unitario è almeno pari a quello del job schedulato in S'. Quindi nelle fasi $P_{n+1}, ..., P_{n'}$, lo schedule S^* processa solo job che hanno valore unitario superiore a v_i.

Nel resto dell'analisi prima classificheremo le fasi e poi costruiremo dei segmenti di al massimo tre fasi consecutive. Per questi segmenti limiteremo superiormente la perdita subita da $ALG2_k$ nel pre-abbandonare i job.

3.1 Classificazione delle fasi Classifichiamo le fasi considerando lo schedule originale S. Una fase P_n si dice pre-abbandonata se un job è pre-abbandonato in $S(P_n)$. La fase P_n di dice continua se il job schedulato per ultimo in $S(P_n)$ è schedulato anche all'inizio di $S(P_{n+1})$. La fase P_n si dice completa se tutti i job schedulati in $S(P_n)$ hanno terminato la loro esecuzione entro la fine di P_n.

Citiamo e verifichiamo alcune proprietà di queste fasi nello schedule S. (a) In ogni fase P_n al massimo un job è pre-abbandonato in $S(P_n)$. (b) Se P_n è una fase continua o completa, allora nessun job viene pre-abbandonato in $S(P_n)$. (c) Se P_n è una fase pre-abbandonata, il job pre-abbandonato è quello con il valore unitario più piccolo tra i lavori schedulati in $S(P_n)$. Queste proprietà possono essere verificate come segue. Sia P_n una fase qualsiasi. Quando $ALG2_k$ costruisce uno schedule per P_n, in primo luogo ordina i lavori di Q_n in ordine di valore unitario non crescente. In questa sequenza ordinata solo l'ultimo job, ad esempio il job i, assegnato a P_n potrebbe non essere schedulato completamente nella fase e quindi è un candidato per il pre-abbandono. Il job i ha il valore unitario più piccolo tra i job schedulati nella fase. Questo dimostra le proprietà (a) e (c). Se il job i non viene spostato all'inizio della fase durante lo Step(2) di $ALG2_k$ e proseguito all'inizio della fase successiva, allora P_n è una fase continua e nessun job viene pre-abbandonato in $S(P_n)$. Per definizione, nessun job è pre-abbandonato in una fase completa. Questo dimostra la proprietà (b). Osserviamo che nello schedule S ogni fase o è pre-abbandonata o continuata o completa.

3.2 Segmenti di Schedule Per ulteriori analisi, partizioniamo lo schedule S in segmenti in cui un segmento è composto da un massimo di tre fasi consecutive. Lo scopo di questi segmenti è quello di combinare fasi pre-abbandonate "costose" con altre fasi in modo da ammortizzare la perdita di pre-abbandono dei job. Innanzitutto costruiamo segmenti costituiti da tre fasi. Le fasi P_n, P_{n+1}, P_{n+2} formano un segmento se P_n è una fase pre-abbandonata che non è preceduta da una fase continua, P_{n+1} è una fase continua e P_{n+2} è una fase pre-abbandonata. Tra le fasi rimanenti costruiamo segmenti costituiti da due fasi. Le fasi P_n, P_{n+1} formano un segmento se (a) P_n è una fase pre-abbandonata che non è preceduta da una fase continua e P_{n+1} è una fase continua o completa o (b) P_n è una fase continua seguita da una fase pre-abbandonata P_{n+1}. Ogni fase rimanente forma un segmento separato.

Dimostriamo adesso che la segmentazione sopra descritta è ben definita. Consideriamo innanzitutto i segmenti trifase costruiti inizialmente. Nessuno di questi segmenti, ad esempio σ e σ', possono sovrapporsi: La prima fase P'_n di σ' è una fase pre-abbandonata che non è preceduta da una fase continua. La fase P'_n non può essere uguale alla seconda fase di

σ perché quest'ultima è fase continua. Inoltre, P'_n non può essere uguale alla terza fase di σ poiché quella è una fase anticipata che è preceduta da una fase continua. Adesso prendiamo in considerazione qualsiasi coppia di segmenti a 2 fasi σ e σ'. Ancora una volta non può verificarsi sovrapposizione. Supponiamo che la prima fase P'_n di σ' sia una fase pre-abbandonata che non è preceduta da una fase continua. P'_n non può essere identica alla seconda fase di σ perché quest'ultima è una fase continua, una fase completa o una fase pre-abbandonata che è preceduta da una fase continua. Adesso supponiamo che P'_n sia una fase continua seguita da una fase pre-abbandonata. Se P'_n e la seconda fase di σ fossero coincidenti, allora le tre fasi di σ e σ' formerebbero un segmento trifase poiché la prima fase di σ è una fase pre-abbandonata che non è preceduta da una fase continua.

Adesso dichiariamo due proprietà di una fase P_n pre-abbandonata che forma un segmento monofase separato. In primo luogo, una tale fase non può essere preceduta da una fase continua: Se P_n è stata preceduta da una fase continua P_{n-1}, allora P_{n-1} potrebbe essere la seconda fase di un segmento a 2 fasi o un segmento monofase separato. Nel primo caso il segmento a 2 fasi e P_n formerebbero inizialmente un segmento a 3 fasi. Nell'ultimo caso invece P_{n-1} e P_n formerebbero un segmento a 2 fasi come descritto nella proprietà (b) sopra. La seconda proprietà è la seguente: P_n è seguito da una fase pre-abbandonata. Se fosse seguito da una fase continua P_{n+1}, allora P_{n+1} potrebbe essere l'inizio di un segmento a 2 fasi descritto nella proprietà (a) sopra o un segmento monofase separato. Nel primo caso P_n e il segmento a 2 fasi si combinerebbero in un segmento a 3 fasi perché P_n non è preceduto da una fase continua. Nell'ultimo caso inveceP_n e P_{n+1} formerebbero un segmento a 2 fasi come descritto nella proprietà (a). In sintesi, una fase pre-abbandonata che forma un segmento monofase separato non è preceduta da una fase continua ed è seguita da una fase pre-abbandonata.

Per un segmento σ, sia $ALG2_k(\sigma)$ il valore ottenuto da $ALG2_k$ su σ. In particolare, sia J l'insieme dei job schedulati da $ALG2_k$ nelle fasi di σ. L'insieme J include anche quei job che sono solo parzialmente elaborati in σ e che potrebbero anche essere schedulati in fasi prima o dopo σ. Supponiamo che il job $i \in J$ sia processato per δ_i unità di tempo a partire dall'istante t_i in σ. Allora

6 $ALG(m)_k$

Il terzo algoritmo, chiamato $ALG(m)_k$, crea uno schedule ottimale per ogni k-fase P_n considerando m macchine parallele. Formalmente $ALG(m)_k$ funziona come segue. Di nuovo, sia Q_n l'insieme dei jobs i che sono arrivati entro l'inizio di P_n, cioè $a_i \leq (n-1)k$, e che non sono stati schedulati nelle fasi precedenti $P_1, \ldots, P_n$. Per ogni $t = (n-1)k, \ldots, nk-1$ $ALG(m)_k$ determina gli m jobs che hanno il più alto valore unitario tra i jobs di Q_n che non hanno terminato al tempo t. Ognuno di questi è schedulato per una sola unità di tempo. Se un job era già stato schedulato al tempo $t-1$, allora viene assegnato alla stessa macchina al tempo t. Se, tra i jobs che non hanno terminato di Qn, l'm-esimo valore unitario più grande è v ed esistono diversi jobs che hanno questo valore, allora viene data la precedenza a quei job che erano già stati schedulati precedentemente. I jobs non ancora iniziati invece vengono considerati in ordine crescente di tempo di arrivo. Se alla fine di P_n ci sono jobs schedulati

che non hanno terminato, vengono pre-rilasciati. $ALG(m)_k$ esegue questo schedule per P_n, ignorando i jobs che potrebbero arrivare durante la fase a tempi $t = (n-1)k+1, \ldots, nk-1$.

7 Bibliografia

1. Dasgupta, A., Ghosh, A., Nazerzadeh, H., Raghavan, P.: Online story scheduling in web advertising. In: Proceedings of the 20th annual ACM-SIAM symposium on discrete algorithms, pp. 1275–1284 (2009)

2. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM 28, 202–208 (1985)