

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-abbandono 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-abbandonata 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 $(n-1)k, \dots, nk-1$, 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-abbandona 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-abbandonare 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 $t = (n-1)k + 1, \dots, nk - 1$. 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. \square

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 perdita 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. \square

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 $CHOP(I_k) = \sum_n CHOP(P_n)$. Mostriamo 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 $\lambda_{j+1}, \dots, l_j$ 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 $\lambda_{j+1}, \dots, l_j$ 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 $CHOP(P_n)/ALG1_k(P_n) \leq 1 + \beta^{nk}/(\beta^{(n-1)k} - \beta^{nk}) = 1/(1-\beta^k)$. \square

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 $ALG1_k(I) = ALG1_k(I_k)$. \square

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 $1/f(k)$. \square

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. \square

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. \square

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 $ALG1_1$ ottiene 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 $4/(2 - \beta) < \min\{1/(1 - \beta), 4/(2 - \beta)\}$. \square

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-abbandonato nella fase immediatamente precedente P_{n-1} . Formalmente $ALG2_k$ funziona come segue. Se $n > 1$, sia i il job che è stato pre-abbandonato 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 $I = (a_i, v_i, l_i)_{i=1}^N$, 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 su-

periore 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. \square

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 J . 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. \square

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 . \square

Lemma 7 *Se il job i viene pre-abbandonato in $S(P_n)$ e i seguenti schedule di fase $S(P_{n+1}), \dots, S(P_{n'})$ processano solo job di valore unitario superiore a v_i , allora S^* non schedula il job i nelle fasi $P_{n+1}, \dots, P_{n'}$.*

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 $S'(P_{n+1}), \dots, S'(P_{n'})$ 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 $P_{n+1}, \dots, P_{n'}$, lo schedule S^* processa solo job che hanno valore unitario superiore a v_i . \square

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.

5.1 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 si 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.

5.2 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 invece P_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

$$ALG2_k(\sigma) = \sum_{i \in J} \beta^{t_i} (1 - \beta^{\delta_i}) / (1 - \beta) \cdot v_i.$$

Sia $CHOP(\sigma)$ il valore ottenuto da $CHOP$ nel processare i job e le porzioni di lavoro schedulate da $ALG2_k$ in σ . Più specificamente, supponiamo che in S il job $i \in J$ sia stato processato per λ_i unità di tempo prima dell'inizio di σ . Quindi $CHOP(\sigma)$ rappresenta il valore raggiunto da $CHOP$ nell'elaborazione delle unità $\lambda_{i+1}, \dots, \lambda_i + \delta_i$ del job i in S^* . Se il job i viene pre-abbandonato nel segmento σ di S , allora $CHOP(\sigma)$ rappresenta inoltre il valore ottenuto nel processare le unità $u > \lambda_i + \delta_i$ in S . Vale inoltre

$$CHOP(\sigma) \leq \sum_{i \in J} \beta^{t_{S^*}(i) + \lambda_i} (1 - \beta^{\delta_i}) / (1 - \beta) \cdot v_i + v_p(\sigma),$$

dove $v_p(\sigma)$ indica il valore aggiuntivo raggiunto da $CHOP$ per i job pre-abbandonati da $ALG2_k$ in σ . Vale inoltre $ALG2_k(I_k) = \sum_{\sigma} ALG2_k(\sigma)$ e $CHOP(I_k) = \sum_{\sigma} CHOP(\sigma)$ perché, per il Lemma 5, ogni job schedulato da $CHOP$ è anche schedulato da $ALG2_k$.

5.3 Analisi dei Segmenti

Dimostriamo tre lemmi (Lemmi 8, 9 e 10) che definiscono il limite superiore di $CHOP(\sigma)/ALG2_k(\sigma)$, per i vari segmenti. Insieme implicheranno il Teorema 3. Nelle dimostrazioni dei lemmi usiamo la seguente notazione. Per ogni fase P_n sia J_n l'insieme dei job che sono parzialmente o completamente elaborati in $S(P_n)$. Per ogni $i \in J_n$, sia $\delta_{i,n}$ il numero di unità di tempo per cui il job i viene processato in $S(P_n)$. Se il job i è schedulato solo nella fase P_n di S , allora semplicemente impostiamo $\delta_i = \delta_{i,n}$. Inoltre, sia i_n^1 il primo job schedulato in $S(P_n)$. Supponiamo che P_n sia preceduto da una fase continua. Durante la costruzione di $S(P_n)$, $ALG2_k$ potrebbe aver ritardato i tempi di inizio di alcuni job di J_n durante lo Step(2) per spostare il job i_n^1 all'inizio della fase. Sia $J'_n \subset J_n$ l'insieme di questi job posticipati. Se P_n non è preceduto da una fase continua, non esistono job posticipati e impostiamo $J'_n = \emptyset$. Osserviamo che $t_S(i) = t_{S'}(i), \forall i \in J_n \setminus J'_n \cup \{i_n^1\}$, e $t_S(i) = t_{S'}(i) + \delta_{i_n^1, n}, \forall i \in J'_n$.

Per facilità di esposizione, sia $w(t, \delta, v) = \beta^t (1 - \beta^\delta) / (1 - \beta) \cdot v$ il valore ottenuto nell'elaborazione di un job di valore unitario v per δ unità di tempo e che inizia all'istante di tempo t . Diciamo inoltre che $\delta = \infty$ quando un job di lunghezza infinita è schedulato a partire dal tempo t .

Lemma 8 *Per qualsiasi σ costituito da una sola fase, $CHOP(\sigma)/ALG2_k(\sigma) \leq \max\{1/\beta^{k-1}, 1 + \beta^{3k}/(1 - \beta^k)\}$.*

Dimostrazione Studiamo innanzitutto il caso in cui la fase P_n di σ è una fase continua o completa, vale a dire che nessun job è pre-abbandonato da $S(P_n)$. Sia $i_1 = i_n^1$ il primo job schedulato in $S(P_n)$. Vale allora

$$\begin{aligned} ALG2_k(\sigma) &= w((n-1)k, \delta_{i_1, n}, v_{i_1}) + \sum_{i \in J'_n} w(t_{S'}(i) + \delta_{i_1, n}, \delta_{i, n}, v_i) \\ &\quad + \sum_{i \in J_n \setminus (J'_n \cup \{i_1\})} w(t_{S'}(i), \delta_{i, n}, v_i) \\ &\geq \beta^{k-1} \left(w((n-1)k, \delta_{i_1, n}, v_{i_1}) + \sum_{i \in J_n \setminus \{i_1\}} w(t_{S'}(i), \delta_{i, n}, v_i) \right). \end{aligned}$$

L'ultima disuguaglianza vale in quanto qualsiasi job $i \in J'_n$ è ritardato di al massimo $k-1$ unità di tempo e quindi, per ogni $i \in J'_n$, abbiamo che $w(t_{S'}(i) + \delta_{i_1, n}, \delta_{i, n}, v_i) = \beta^{\delta_{i_1, n}} w(t_{S'}(i), \delta_{i, n}, v_i) \geq \beta^{k-1} w(t_{S'}(i), \delta_{i, n}, v_i)$. Ogni job $i \in J_n$, ad eccezione di un possibile i_1 , viene avviato in (P_n) . Se il job i_1 viene avviato in $S(P_{n'})$, con $n' < n$, allora il job viene schedulato alla fine di $S(P_{n'})$. Quindi quando $ALG2_k$ ha costruito $S(P_{n'})$, il job non è stato ritardato durante lo Step(2) dell'algoritmo per spostare un altro job all'inizio della fase. Quindi $t_S(i_1) = t_{S'}(i_1) \leq t_{S^*}(i_1)$. Supponiamo che prima di P_n il job i_1 fosse elaborato per λ_{i_1} unità di tempo in S . Poiché $t_S(i_1) \leq t_{S^*}(i_1)$, le unità di tempo $\lambda_{i_1} + 1, \dots, \lambda_{i_1} + \delta_{i_1, n}$ del job i_1 non possono essere avviate prima del inizio di P_n in S^* . Per ogni job $i \in J_n \setminus \{i_1\}$, vale $t_{S'}(i) \leq t_{S^*}(i)$. Quindi

$$CHOP(\sigma) \leq w((n-1)k, \delta_{i_1, n}, v_{i_1}) + \sum_{i \in J_n \setminus \{i_1\}} w(t_{S'}(i), \delta_{i, n}, v_i).$$

Otteniamo $CHOP(\sigma)/ALG2_k(\sigma) \leq 1/\beta^{k-1}$.

Successivamente studieremo il caso in cui P_n è una fase pre-abbandonata. La fase precedente P_{n-1} non è una fase continua mentre la fase successiva P_{n+1} è anch'essa una fase pre-abbandonata. Poiché P_n non è preceduta da una fase continua, tutti i job di J_n vengono avviati in $S(P_n)$ e $t_S(i) = t_{S'}(i), \forall i \in J_n$. Otteniamo

$$ALG2_k(\sigma) = \sum_{i \in J_n} w(t_{S'}(i), \delta_i, v_i).$$

Sia $i_p \in J_n$ il job pre-abbandonato in $S(P_n)$. Il job viene pre-abbandonato alla fine di $S(P_n)$. Inoltre, il suo valore unitario è strettamente inferiore al valore unitario di qualsiasi job schedulato in $S(P_{n+1})$, cioè lo schedule della fase successiva, poiché altrimenti $ALG2_k$ avrebbe schedulato il job i_p in $S(P_{n+1})$. Anche la fase P_{n+1} è una fase pre-abbandonata e il job pre-abbandonato in $S(P_{n+1})$ è schedulato alla fine di $S(P_{n+1})$. Pertanto, il job pre-abbandonato in $S(P_{n+1})$ ha un valore unitario strettamente inferiore rispetto a qualsiasi job programmato in $S(P_{n+2})$. Ne consegue che il job i_p ha un valore unitario strettamente inferiore rispetto a qualsiasi job schedulato in $S(P_{n+1})$ e in $S(P_{n+2})$. Il Lemma 7 assicura che $CHOP$ non scheduli i_p nelle fasi P_{n+1} e P_{n+2} . Pertanto, il valore ottenuto da $CHOP$ per la parte pre-abbandonata di i_p è limitato superiormente da $w((n+2)k, \infty, v_{i_p})$ e

$$CHOP(\sigma) \leq \sum_{i \in J_n} w(t_{S'}(i), \delta_i, v_i) + w((n+2)k, \infty, v_{i_p})$$

$$= ALG2_k(\sigma) + w((n+2)k, \infty, v_{i_p}).$$

Ricordiamo che il job i_p è quello con il valore unitario più piccolo tra i job schedulati in $S(P_n)$. Pertanto $ALG2_k(\sigma) \geq w((n-1)k, k, v_{i_p}) = \beta^{(n-1)k}(1 - \beta^k)/(1 - \beta) \cdot v_{i_p}$. Inoltre $w((n+2)k, \infty, v_{i_p}) = \beta^{(n+2)k}/(1 - \beta)v_{i_p}$. Concludiamo dicendo che $CHOP(\sigma)/ALG2_k(\sigma) \leq 1 + \beta^{3k}/(1 - \beta^k)$.

Lemma 9 *Sia σ un segmento costituito da almeno due fasi. Se σ è costituito da due fasi, supponiamo che la prima sia una fase pre-abbandonata. Se σ è costituito da tre fasi, supponiamo che il valore unitario del job pre-abbandonato nella prima fase sia almeno pari a quello del job pre-abbandonato nella terza fase. Allora vale $CHOP(\sigma)/ALG2_k(\sigma) \leq \max\{1/\beta^{k-1}, 1/(1 - \beta^{2k})\}$.*

Dimostrazione Studiamo innanzitutto il caso in cui σ consiste esattamente in due fasi P_n e P_{n+1} . Poiché P_n è una fase pre-abbandonata, per come sono costruiti i segmenti a 2 fasi, P_n non è preceduto da una fase continua. Poiché né P_n né P_{n+1} sono preceduti da una fase continua, tutti i job di J_n e J_{n+1} vengono avviati rispettivamente in $S(P_n)$ e $S(P_{n+1})$ e gli insiemi J'_n e J'_{n+1} sono vuoti. Quindi $t_S(i) = t_{S'}(i)$, $\forall i \in J_n \cup J_{n+1}$. Otteniamo

$$ALG2_k(\sigma) = \sum_{i \in J_n} w(t_{S'}(i), \delta_{i,n}, v_i) + \sum_{i \in J_{n+1}} w(t_{S'}(i), \delta_{i,n+1}, v_i). \quad (1)$$

Poiché $t_S(i) = t_{S'}(i) \leq t_{S^*}(i)$, $\forall i \in J_n \cup J_{n+1}$, $CHOP$ non può ottenere un valore più elevato per i job e le porzioni di job schedulati da $ALG2_k$ in σ . Può ottenere solo un valore aggiuntivo per la porzione di job che è stata pre-abbandonata in $S(P_n)$. Sia i_p il job pre-abbandonato. Questo job è schedulato per ultimo in $S(P_n)$. È quello con il valore unitario più piccolo tra i job schedulati in $S(P_n)$. Inoltre, ha un valore unitario strettamente inferiore rispetto a qualsiasi job schedulato in $S(P_{n+1})$ perché altrimenti $ALG2_k$ avrebbe incluso il job i_p nello schedule $S(P_{n+1})$. Per il Lemma 7 $CHOP$ non schedula il job i_p nella fase P_{n+1} . Poiché $t_{S'}(i_p) \leq t_{S^*}(i_p)$, $CHOP$ può schedulare al massimo $\delta_{i_p,n}$ unità di tempo del job i_p prima della fine di P_n . Quindi il valore addizionale di $CHOP$ ottenuto nell'elaborazione della porzione pre-abbandonata di i_p è al massimo $w((n+1)k, \infty, v_{i_p})$. Dunque

$$CHOP(\sigma) \leq ALG2_k(\sigma) + w((n+1)k, \infty, v_{i_p}). \quad (2)$$

In $S(P_n)$ e $S(P_{n+1})$ tutti i job hanno un valore unitario di almeno v_{i_p} . Quindi $ALG2_k(\sigma) \geq \beta^{(n-1)k}(1 - \beta^{2k})/(1 - \beta) \cdot v_{i_p}$. Tenendo conto che $w((n+1)k, \infty, v_{i_p}) = \beta^{(n+1)k}/(1 - \beta)v_{i_p}$ concludiamo che $CHOP(\sigma)/ALG2_k(\sigma) \leq 1 + \beta^{2k}/(1 - \beta^{2k}) = 1/(1 - \beta^{2k})$.

Consideriamo quindi il caso in cui σ è costituito da tre fasi P_n , P_{n+1} , P_{n+2} e che il job pre-abbandonato nella prima fase ha un valore per unità che è almeno pari a quello del job pre-abbandonato nella terza fase. Per definizione di un segmento trifase, P_n è una fase anticipata che non è preceduta da una fase continua e P_{n+1} è una fase continua. Questa combinazione di fasi è stata analizzata sopra. Sia σ_1 il sottosegmento costituito dalle fasi P_n e P_{n+1} . Usando (1) e (2), otteniamo

$$ALG2_k(\sigma_1) = \sum_{i \in J_n} w(t_{S'}(i), \delta_{i,n}, v_i) + \sum_{i \in J_{n+1}} w(t_{S'}(i), \delta_{i,n+1}, v_i)$$

e

$$CHOP(\sigma_1) \leq ALG2_k(\sigma_1) + w((n+1)k, \infty, v_{i_p}), \quad (3)$$

dove i_p è di nuovo il job pre-abbandonato in $S(P_n)$.

Sia σ_2 il sottosegmento costituito dalla terza fase P_{n+2} e sia $i_1 = i_{n+2}^1$ il job schedulato all'inizio di P_{n+2} . Allora

$$\begin{aligned} ALG2_k(\sigma_2) &= w((n+1)k, \delta_{i_1, n+2}, v_{i_1}) + \sum_{i \in J'_{n+2}} w(t_{S'}(i) + \delta_{i_1, n+2}, \delta_i, v_i) \\ &\quad + \sum_{i \in J_{n+2} \setminus (J'_{n+2} \cup \{i_1\})} w(t_{S'}(i), \delta_i, v_i) \\ &\geq \beta^{k-1} \left(w((n+1)k, \delta_{i_1, n+2}, v_{i_1}) + \sum_{i \in J_{n+2} \setminus \{i_1\}} w(t_{S'}(i), \delta_i, v_i) \right). \end{aligned}$$

La disuguaglianza è valida perché qualsiasi $i \in J'_{n+2}$ è ritardato di al massimo $k-1$ unità di tempo e quindi $w(t_{S'}(i) + \delta_{i_1, n+2}, \delta_i, v_i) = \beta^{\delta_{i_1, n+2}} w(t_{S'}(i), \delta_i, v_i) \geq \beta^{k-1} w(t_{S'}(i), \delta_i, v_i)$, $\forall i \in J'_{n+2}$.

Sia i'_p il job pre-abbandonato in $S(P_{n+2})$. Per assunzione questo job ha un valore per unità che è limitato superiormente dal valore unitario di i_p . Il valore aggiuntivo ottenuto da $CHOP$ nello schedulare le porzioni pre-abbandonate di i_p e i'_p è limitato superiormente dal termine $w((n+1)k, \infty, v_{i_p})$ in (3), che, di nuovo, è il valore ottenuto nello schedulare un job infinitamente lungo di valore unitario v_{i_p} a partire dal tempo $(n+1)k$. Abusando leggermente della notazione, sia

$$CHOP(\sigma_2) = w((n+1)k, \delta_{i_1, n}, v_{i_1}) + \sum_{i \in J_{n+2} \setminus \{i_1\}} w(t_{S'}(i), \delta_{i, n}, v_i)$$

il valore ottenibile da $CHOP$ nell'elaborazione i job e le porzioni di job di $S(P_{n+2})$. Valgono dunque $ALG2_k(\sigma) = ALG2_k(\sigma_1) + ALG2_k(\sigma_2)$ e $CHOP(\sigma) \leq CHOP(\sigma_1) + CHOP(\sigma_2)$. Quindi $CHOP(\sigma)/ALG2_k(\sigma) \leq \max\{CHOP(\sigma_1)/ALG2_k(\sigma_1), CHOP(\sigma_2)/ALG2_k(\sigma_2)\}$. Come mostrato sopra, il primo rapporto è limitato superiormente da $1/(1 - \beta^{2k})$. Il secondo rapporto è al massimo $1/\beta^{k-1}$. \square

Lemma 10 *Sia σ un segmento costituito da almeno due fasi. Se σ è costituito da esattamente due fasi, supponiamo che la prima sia una fase continua. Se σ è costituito da tre fasi, supponiamo che il valore unitario del job pre-abbandonato nella prima fase sia inferiore a quello del job pre-abbandonato nella terza fase. Allora vale $CHOP(\sigma)/ALG2_k(\sigma) \leq \max\{1/\beta^{k-1}, 1/(1 - \beta^{2k})\}$.*

Dimostrazione Le tecniche utilizzate per dimostrare il Lemma 10 sono simili a quelle della dimostrazione del Lemma 9. In questo caso l'analisi diventa leggermente più impegnativa perché in caso di un segmento a 2 fasi entrambe le fasi P_n e P_{n+1} possono inizialmente elaborare i job i_n^1 e i_{n+1}^1 che erano già stati schedulati nelle fasi precedenti. Mostriamo che il caso peggiore si verifica se i valori unitari di i_n^1 e i_{n+1}^1 sono approssimativamente uguali a quelli appartenenti a J'_n e J'_{n+1} , rispettivamente.

Formalmente, per prima cosa analizzeremo un segmento σ costituito da due fasi P_n e P_{n+1} . Per ipotesi, la prima è una fase continua. Siano $i_1 = i_n^1$ e $i_2 = i_{n+1}^1$ i job schedulati per primi rispettivamente in $S(P_n)$ e $S(P_{n+1})$. Allora vale

$$ALG2_k(\sigma) = w((n-1)k, \delta_{i_1, n}, v_{i_1}) + \sum_{i \in J'_n} w(t_{S'}(i) + \delta_{i_1, n}, \delta_{i, n}, v_i) \quad (4)$$

$$\begin{aligned} & + \sum_{i \in J_n \setminus (J'_n \cup \{i_1\})} w(t_{S'}(i), \delta_{i, n}, v_i) \\ & + w(nk, \delta_{i_2, n+1}, v_{i_2}) + \sum_{i \in J'_{n+1}} w(t_{S'}(i) + \delta_{i_2, n+1}, \delta_{i, n+1}, v_i) \\ & + \sum_{i \in J_{n+1} \setminus (J'_{n+1} \cup \{i_2\})} w(t_{S'}(i), \delta_{i, n+1}, v_i). \end{aligned} \quad (5)$$

Adesso analizziamo il valore che *CHOP* può ottenere per i job e le porzioni di job schedulati in $S(P_n)$ e $S(P_{n+1})$. Se in S il job i_1 viene avviato prima dell'inizio di P_n , allora i_1 viene avviato alla fine di una fase precedente $P_{n'}$, con $n' < n$, e inoltre, quando $ALG2_k$ ha costruito $S(P_{n'})$, l'algoritmo non ha ritardato il tempo di inizio del job i_1 durante lo Step(2). Poiché $t_{S'}(i_1) \leq t_{S^*}(i_1)$, *CHOP* non può schedulare le $\delta_{i_1, n}$ unità del job i_1 processate da $ALG2_k$ in $S(P_n)$ prima dell'inizio di P_n . Per qualsiasi altro job $i \in J_n, i \neq i_1$, vale $t_{S'}(i) \leq t_{S^*}(i)$. Pertanto *CHOP* non può non avviare i job e le porzioni di job schedulate in $S(P_n)$ prima dell'inizio di P_n . Il valore più alto si ottiene sequenziando i job e le porzioni di job in ordine non decrescente di valore unitario. Questa è di fatto la sequenza usata in $S'(P_n)$, che prima elabora i job di J'_n , poi le $\delta_{i_1, n}$ unità di tempo del job i_1 e infine i job di $J_n \setminus J'_n$. Sia $\Delta_1 = \sum_{i \in J'_n} \delta_{i, n}$. Il valore ottenuto da *CHOP* per i job e le porzioni di job schedulati in $S(P_n)$ è al massimo

$$\sum_{i \in J'_n} w(t_{S'}(i), \delta_{i, n}, v_i) + w((n-1)k + \Delta_1, \delta_{i_1, n}, v_{i_1}) + \sum_{i \in J_n \setminus (J'_n \cup \{i_1\})} w(t_{S'}(i), \delta_{i, n}, v_i).$$

Un'espressione analoga vale per la fase P_{n+1} . Sia i_p il job pre-abbandonato in $S(P_{n+1})$. Questo job è schedulato per ultimo in $S'(P_{n+1})$ e poiché $t_{S'}(i_p) \leq t_{S^*}(i_p)$, *CHOP* può elaborare al massimo $\delta_{i_p, n+1}$ unità di questo job prima della fine di P_{n+1} . Pertanto, il valore aggiuntivo ottenuto da *CHOP* nell'elaborazione della porzione pre-abbandonata di i_p è al massimo $w((n+1)k, \infty, v_{i_p})$. Dunque possiamo concludere affermando che

$$CHOP(\sigma) \leq \sum_{i \in J'_n} w(t_{S'}(i), \delta_{i, n}, v_i) + w((n-1)k + \Delta_1, \delta_{i_1, n}, v_{i_1}) \quad (6)$$

$$\begin{aligned} & + \sum_{i \in J_n \setminus (J'_n \cup \{i_1\})} w(t_{S'}(i), \delta_{i, n}, v_i) \\ & + \sum_{i \in J_{n+1}} w(t_{S'}(i), \delta_{i, n+1}, v_i) + w(nk + \Delta_2, \delta_{i_2, n+1}, v_{i_2}) \end{aligned} \quad (7)$$

$$+ \sum_{i \in J_{n+1} \setminus (J'_{n+1} \cup \{i_2\})} w(t_{S'}(i), \delta_{i,n+1}, v_i) + w((n+1)k, \infty, v_{i_p}) \quad (8)$$

dove Δ_2 è la lunghezza totale dei job in J'_{n+1} . Definiamo $R = CHOP(\sigma)/ALG2_k(\sigma)$. Allo scopo di limitare superiormente R consideriamo i rapporti limitati agli insiemi di job J'_n e J'_{n+1} . Se $J'_n = \emptyset$, sia

$$r_1 = \sum_{i \in J'_n} w(t_{S'}(i), \delta_{i,n}, v_i) / \sum_{i \in J'_n} w(t_{S'}(i) + \delta_{i_1,n}, \delta_{i,n}, v_i).$$

Se $J'_{n+1} \neq \emptyset$, sia

$$r_2 = \sum_{i \in J'_{n+1}} w(t_S(i), \delta_{i,n+1}, v_i) / \sum_{i \in J'_{n+1}} w(t_{S'}(i) + \delta_{i_2,n+1}, \delta_{i,n+1}, v_i).$$

Se $J'_n \neq \emptyset$, allora $r_1 = 1/\beta^{\delta_{i_1,n}} \leq 1/\beta^{k-1}$. Allo stesso modo, se $J'_{n+1} \neq \emptyset$, allora $r_2 = 1/\beta^{\delta_{i_2,n+1}} \leq 1/\beta^{k-1}$.

Se R non è limitato superiormente da $1/\beta^{k-1}$, allora R sta aumentando per valori decrescenti di v_i , con $i \in J'_n \cup J'_{n+1}$. Ricordiamo che ogni v_i , con $i \in J'_n$, è maggiore di v_{i_1} ; ogni v_i , con $i \in J'_{n+1}$, è maggiore di v_{i_2} . Quindi R è limitato superiormente da $1/\beta^{k-1}$ oppure dal rapporto ottenuto impostando $v_i = v_{i_1}$, per $i \in J'_n$ e $v_i = v_{i_2}$, per $i \in J'_{n+1}$. Con questa impostazione $\sum_{i \in J'_n} w(t_{S'}(i) + \delta_{i_1,n}, \delta_{i,n}, v_i) = w((n-1)k + \delta_{i_1,n}, \Delta_1, v_{i_1})$ e il secondo termine di (4) diventa $w((n-1)k, \delta_{i_1,n} + \Delta_1, v_{i_1})$. Allo stesso modo, l'espressione (5) diventa $w(nk, \delta_{i_2,n+1} + \Delta_2, v_{i_2})$. Analogamente, $\sum_{i \in J'_n} w(t_{S'}(i), \delta_{i,n}, v_i) = w((n-1)k, \Delta_1, v_{i_1})$ e il secondo termine di (6) diventa $w((n-1)k, \delta_{i_1,n} + \Delta_1, v_{i_1})$. L'espressione (7) diventa $w(nk, \delta_{i_2,n+1} + \Delta_2, v_{i_2})$. Con queste osservazioni, $ALG2_k(\sigma)$ si semplifica in

$$\begin{aligned} V &= w((n-1)k, \delta_{i_1,n} + \Delta_1, v_{i_1}) + \sum_{i \in J_n \setminus (J'_n \cup \{i_1\})} w(t_{S'}(i), \delta_{i,n}, v_i) \\ &+ w(nk, \delta_{i_2,n+1} + \Delta_2, v_{i_2}) + \sum_{i \in J_{n+1} \setminus (J'_{n+1} \cup i_2)} w(t_{S'}(i), \delta_{i,n+1}, v_i) \end{aligned}$$

e $CHOP(\sigma)$ si semplifica in $V + w((n+1)k, \infty, v_{i_p})$. Se R non è limitato superiormente da $1/\beta^{k-1}$, allora è limitato superiormente da $R' = (V + w((n+1)k, \infty, v_{i_p}))/V$. Ricordiamo che P_n è una fase continua. Il job schedulato per ultimo in $S(P_n)$ è quello con il valore unitario più piccolo tra i job schedulati in $S(P_n)$. Poiché il job è anche schedulato in $S(P_{n+1})$, ha un valore unitario che è almeno pari a quello del job i_p . Questo job pre-abbandonato fa parte dei job in $S(P_{n+1})$ con il valore unitario più piccolo. Concludiamo dicendo che tutti i job schedulati in $S(P_n)$ e $S(P_{n+1})$ hanno un valore unitario di almeno v_{i_p} . Ciò implica che $V \geq \beta^{(n-1)k}(1 - \beta^{2k})/(1 - \beta)v_{i_p}$. Inoltre $w((n+1)k, \infty, v_{i_p}) = \beta^{(n+1)k}/(1 - \beta)v_{i_p}$. Infine vale $R' \leq 1 + \beta^{2k}/(1 - \beta^{2k}) = 1/(1 - \beta^{2k})$.

Supponiamo adesso che σ sia costituito da tre fasi. Poiché un segmento trifase termina con una fase continua seguita da una pre-abbandonata, possiamo ridurre l'analisi a quella sviluppata in precedenza. Sia P_{n-1} la prima fase di σ . Poiché P_{n-1} non è preceduta da una fase continua, anche tutti i job schedulati in $S(P_{n-1})$ vengono avviati in questa fase.

Inoltre, quando $ALG2_k$ costruisce $S(P_{n-1})$ nessun job viene ritardato durante lo Step(2) per spostare un job all'inizio della fase. Poiché $t_S(i) = t_{S'}(i) \leq t_{S^*}(i)$, $\forall i \in J_{n-1}$, il valore ottenuto da $CHOP$ nello schedulare i job e le porzioni di job di $S(P_{n-1})$ non è superiore al valore ottenuto da $ALG2_k$. Sia i'_p il job pre-abbandonato in $S(P_{n-1})$. Poiché il job non è schedulato in $S(P_n)$ ha un valore unitario strettamente inferiore rispetto a qualsiasi altro job elaborato in $S(P_n)$. Per ipotesi ha un valore unitario inferiore rispetto al job i_p . Quindi il valore unitario del job i'_p è strettamente inferiore a quello di qualsiasi altro job sequenziato in $S(P_n)$ e in $S(P_{n+1})$. Il Lemma 7 implica che $CHOP$ non schedula il job i'_p in P_n e in P_{n+1} . Pertanto $CHOP$ schedula la parte pre-abbandonata di i'_p non prima del tempo $(n+1)k$. Ne consegue che il valore aggiuntivo totale ottenuto da $CHOP$ per le porzioni anticipate dei job i'_p e i_p è al massimo $w((n+1)k, \infty, v_{i_p})$, che è l'ultimo termine in (8). Poiché per i job e le porzioni di job schedulati in $S(P_{n-1})$ $CHOP$ non ottiene un valore superiore a quello di $ALG2_k$, ne consegue che $CHOP(\sigma)/ALG2_k(\sigma)$ è limitato superiormente dal rapporto R sopra analizzato. \square

Procediamo con la dimostrazione del Teorema 3.

Dimostrazione del Teorema 3 Il Lemma 8 analizza segmenti costituiti da una singola fase. I Lemmi 9 e 10 analizzano tutti i possibili casi di segmenti bifase e trifase. Insieme, i tre lemmi implicano che $CHOP(\sigma)/ALG2_k(\sigma) \leq \max\{1/\beta^{k-1}, 1/(1-\beta^{2k}), 1+\beta^{3k}/(1-\beta^k)\}$, per qualsiasi segmento σ . Quindi $CHOP(I_k)/ALG2_k(I_k)$ e $OPT(I_k)/ALG2_k(I_k)$ sono entrambi limitati superiormente da $\max\{1/\beta^{k-1}, 1/(1-\beta^{2k}), 1+\beta^{3k}/(1-\beta^k)\}$. Il teorema segue perché $1/\beta^{k-1}OPT(I_k) \geq OPT(I)$ e $ALG2_k(I_k) = ALG2_k(I)$. \square

Concludiamo la nostra analisi determinando il miglior valore di k .

Corollario 4 Sia $c = 1 + \phi$, dove $\phi = (1 + \sqrt{5})/2$ è la sezione aurea. Per $k = \lfloor -\frac{1}{2} \log_{\beta} c \rfloor + 1$, $ALG2_k$ ottiene un rapporto competitivo di $c \approx 2.618$.

Dimostrazione Sia $k^* = \lfloor -\frac{1}{2} \log_{\beta} c \rfloor + 1$. Valutiamo il rapporto competitivo definito dal Teorema 3. Per prima cosa, $1/\beta^{k^*-1} \leq \sqrt{c}$ e quindi $1/\beta^{2(k^*-1)} \leq c$. Dimostriamo che $1/(1-\beta^{2k^*})$ e $\beta^{3k^*}/(1-\beta^{k^*})$ sono entrambi limitati superiormente da \sqrt{c} . La prima affermazione è vera in quanto $1/(1-\beta^{2k^*}) \leq c/(c-1) = c/\sqrt{c} = \sqrt{c}$. Inoltre, $\beta^{3k^*}/(1-\beta^{k^*}) \leq 1/(c\sqrt{c}-c)$. Allo scopo di dimostrare che $1+\beta^{3k^*}/(1-\beta^{k^*}) \leq \sqrt{c}$ è sufficiente verificare che $1 \leq c - \sqrt{c}$. L'ultima disuguaglianza è soddisfatta poiché $c - \sqrt{c} = 1$. \square

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, \dots, P_n . Per ogni $t = (n-1)k, \dots, nk-1$ $ALG(m)_k$ 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 Q_n , 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-abbandonati. $ALG(m)_k$ esegue questo schedule per P_n , ignorando i jobs che potrebbero arrivare durante la fase a tempi $t = (n-1)k+1, \dots, nk-1$.

Teorema 4 $\forall k \in \mathbb{N}$ e tutte le probabilità β , $ALG(m)_k$ ottiene un rapporto competitivo di $\frac{1}{\beta^{k-1}}(1 + \frac{1}{1-\beta^k})$.

Di seguito dimostreremo il teorema 4. Sfortunatamente, non possiamo estendere l'analisi di $ALG1_k$ fornita in Sez. 4 poiché il Lemma 2 non vale per schedule generati da $CHOP$ eseguito su m -macchine. Pertanto dobbiamo utilizzare schedule "ottimali" differenti. Come al solito, dato un qualsiasi input $I = (a_i, v_i, l_i)_{i=1}^N$, consideriamo l'input k -quantizzato $I_k = (a'_i, v_i, l_i)_{i=1}^N$ con $a'_i = k \lceil a_i/k \rceil$. Vale $ALG(m)_k(I) = ALG(m)_k(I_k)$ e $1/\beta^{k-1}OPT(I_k) \geq OPT(I)$, che può essere dimostrato allo stesso modo del Lemma 1. Sia S lo schedule costruito da $ALG(m)_k$ per I_k . Inoltre sia S_{OPT} uno schedule ottimale per I_k .

L'idea principale della dimostrazione del Teorema 4 è quella di costruire un schedule S^* il cui valore sia almeno pari a quello di S_{OPT} e che ci permetterà di confrontare i valori di S e S^* . Lo schedule S^* può essere derivato da S_{OPT} . In S^* i job possono essere interrotti e migrati fra le macchine. Inoltre S^* può elaborare fino a $2m$ job a qualsiasi tempo t . Per ogni tempo t , sia J_t^* l'insieme di job schedulati in S^* al tempo t . Allo stesso modo, sia J_t l'insieme dei job elaborati in S al tempo t . L'insieme $J_t^* \setminus J_t$ Contiene i job schedulati solo in S^* e non in S in quell'istante. Definiremo un abbinamento che abbini alcuni job di J_t^* con quelli in J_t . Usando questo abbinamento possiamo limitare superiormente il valore extra ottenuto da S^* relativo al valore di S . Lo schedule S^* e l'abbinamento associato soddisfano una proprietà specifica. Per qualsiasi tempo t , ogni job $i \in J_t^* \setminus J_t$ è abbinato a un job $i' \in J_t$ il cui valore unitario è almeno pari a quello del job i , oppure il job i è schedulato in S alla fine di una fase precedente P_n , cioè in un certo istante di tempo $t' = nk - 1$ con $nk - 1 < t$. Ciascun $i' \in J_t$ è abbinato al massimo a un $i \in J_t^*$. Notiamo che l'abbinamento dipende dal tempo. Nel corso del tempo un job schedulato in S o S^* può essere abbinato a vari job. Tuttavia, per qualsiasi tempo fissato t , ciascun $i' \in J_t$ è abbinato al massimo a un $i \in J_t^*$ e viceversa. Formalmente, gli schedule S^* e S , unitamente agli insiemi J_t^* e J_t , soddisfano la seguente proprietà (P).

(P) Per qualsiasi tempo t , ogni job $i \in J_t^* \setminus J_t$ è abbinato a un $i' \in J_t$ tale che $v_{i'} \geq v_i$ oppure è schedulato in S alla fine di una fase precedente P_n , ovvero ad un certo istante $nk - 1$ con $nk - 1 < t$. Per ogni tempo t , ciascun $i' \in J_t$ è abbinato al massimo a un $i \in J_t^*$. Sia $OPT^*(I_k)$ il valore dello schedule S^* .

Lemma 11 Esiste uno schedule S^* che soddisfa la proprietà(P) e $OPT^*(I_k) \geq OPT(I_k)$.

Dimostrazione Trasformiamo S_{OPT} in uno schedule che soddisfi l'affermazione del lemma. Consideriamo innanzitutto lo schedule S . Ricordiamo che $ALG(m)_k$ schedula ogni job di I_k in una sola fase, vale a dire che nessun job viene elaborato in due o più fasi. Per ogni fase P_n , $n \geq 1$, sia J_n l'insieme di job che sono sequenziati nello schedule S in P_n ma non alla fine di questa fase al tempo $nk - 1$. Osserviamo che ogni job $i \in J_n$ è schedulato completamente, vale a dire per l_i unità di tempo in P_n : $ALG(m)_k$ schedula sempre i job con i più alti valori

unitari tra i job non terminati disponibili. Se un job è schedulato al tempo t e non al tempo $t + 1$ e t non è la fine di una fase, il job viene completamente processato poiché in I_k non arrivano job durante una fase. Sia $t_S(i)$ il tempo di inizio del job i in S . Allora il job $i \in J_n$ è schedulato negli istanti di tempo $t_S(i), \dots, t_S(i) + l_i - 1$ in S . Sia $J = \bigcup_{n \geq 1} J_n$. Ora modifichiamo S_{OPT} come segue. Per ogni job $i \in J$ e ogni tempo t con $t_S(i) \leq t \leq t_S(i) + l_i - 1$, un'unità di job i è schedulata al tempo t a condizione che S_{OPT} non abbia già schedulato questo istante di tempo t . Inoltre, le unità del job i sequenziate in S_{OPT} a tempi $t \geq t_S(i) + l_i$ vengono eliminate. Queste modifiche non riducono il valore dello schedule perché ogni unità di job eliminata al tempo t viene sostituita da un'unità corrispondente schedulata in un certo tempo $t' < t$. Sia S^* lo schedule risultante. Si noti che in S^* è possibile schedulare fino a $2m$ job in qualsiasi momento.

Per ogni tempo t , sia $\Delta_t = J_t^* \setminus J_t$ l'insieme dei job che sono schedulati solo in S^* e non in S al tempo t . Partizioniamo questo insieme in due sottoinsiemi. Sia $\Delta_t^1 \subseteq \Delta_t$ l'insieme dei job schedulati a tempi $t' < t$ in S . L'insieme $\Delta_t^2 = \Delta_t \setminus \Delta_t^1$ contiene i job rimanenti. Per prima cosa studiamo Δ_t^1 e discutiamo il fatto che ogni $i \in \Delta_t^1$ debba essere schedulato in S alla fine di una fase P_n con $nk - 1 < t$. Secondo la definizione di Δ_t^1 , il job i non è schedulato in S al tempo t ma è schedulato in S in qualche istante $t' < t$. Sia P_n la fase contenente il tempo t' . Se il job i non è schedulato in S alla fine di P_n , allora $i \in J_n$. Tuttavia, le modifiche dello schedule descritte nel paragrafo precedente hanno eliminato le unità del job i sequenziate dopo $t_S(i) + l_i - 1$ e quindi al tempo t . Concludiamo che ciascun $i \in \Delta_t^1$ soddisfa la proprietà (P). Resta da considerare i job di Δ_t^2 . Nessuno di questi job è schedulato in S a tempi t' con $t' \leq t$. Poiché i job di Δ_t^2 sono disponibili per l'elaborazione al tempo t , lo schedule S sequenzia esattamente m job al tempo t . Ricordiamo ancora che $ALG(m)_k$ elabora sempre i job non terminati disponibili con i più alti valori unitari. Quindi J_t contiene $|\Delta_t^2|$ job i cui valori unitari sono almeno pari al valore unitario di ciascun job in Δ_t^2 . Questi job di J_t possono essere abbinati in modo arbitrario ai job di Δ_t^2 . Ciò conferma che la proprietà (P) è soddisfatta per ogni $i \in \Delta_t^2$ e quindi per tutti i job di Δ_t .

Lemma 12 $\forall k \in \mathbb{N}$ e tutte le probabilità β , vale la disuguaglianza $(1 + \frac{1}{1-\beta^k})ALG(m)_k(I_k) \geq OPT^*(I_k)$.

Dimostrazione Per ogni fase P_n , $n \geq 1$, $ALG(m)_k(P_n)$ indica il valore ottenuto in S durante la fase P_n . Allo stesso modo, $OPT^*(P_n)$ è il valore ottenuto in S^* durante P_n . Valgono dunque $ALG(m)_k(I_k) = \sum_{n \geq 1} ALG(m)_k(P_n)$ e $OPT^*(I_k) = \sum_{n \geq 1} OPT^*(P_n)$. In S^* consideriamo un tempo qualsiasi t . Sia $J_{t,1}^* \subseteq J_t^*$, l'insieme di job che sono schedulati anche in S al tempo t o che sono abbinati a un job in J_t . L'insieme $J_{t,2}^* = J_t^* \setminus J_{t,1}^*$ è costituito dai job rimanenti di J_t^* . Secondo la definizione di $J_{t,1}^*$ e la proprietà (P), ogni job di $J_{t,2}^*$ è schedulato in S alla fine di una fase precedente, ovvero ad un certo tempo $t' = nk - 1$ con $t' < t$. Sia $OPT_1^*(t) = \sum_{i \in J_{t,1}^*} \beta^t v_i$ e $OPT_2^*(t) = \sum_{i \in J_{t,2}^*} \beta^t v_i$ i valori ottenuti nello schedulare job di $J_{t,1}^*$ e $J_{t,2}^*$ rispettivamente, al tempo t . Sommando queste ultime quantità per ogni tempo t di una fase P_n , definiamo $OPT_1^*(P_n) = \sum_{t=(n-1)k}^{nk-1} OPT_1^*(t)$ e $OPT_2^*(P_n) = \sum_{t=(n-1)k}^{nk-1} OPT_2^*(t)$. Vale dunque $OPT^*(P_n) = OPT_1^*(P_n) + OPT_2^*(P_n)$.

Sia $ALG(m)_k(t) = \sum_{i \in J_t} \beta^t v_i$ il valore ottenuto nello schedule S al tempo t . Ancora una volta $ALG(m)_k(P_n) = \sum_{t=(n-1)k}^{nk-1} ALG(m)_k(t)$ è il valore totale per la fase P_n . Per

la definizione di $J_{t,1}^*$, ogni job di questo insieme è contenuto in J_t o è abbinato ad un job $i' \in J_t$ tale che $v_i \leq v_{i'}$. Quindi $OPT_1^*(t) \leq 2ALG(m)_k(t)$ e, sommando ciascun membro di questa disuguaglianza per tutti gli istanti di tempo della fase P_n , otteniamo $OPT_1^*(P_n) \leq 2ALG(m)_k(P_n)$. Segue che

$$\begin{aligned} OPT^*(I_k) &= \sum_{n \geq 1} OPT_1^*(P_n) + \sum_{n \geq 1} OPT_2^*(P_n) \leq \sum_{n \geq 1} 2ALG(m)_k(P_n) + \sum_{n \geq 1} OPT_2^*(P_n) \\ &= 2ALG(m)_k(I_k) + \sum_{n \geq 1} OPT_2^*(P_n). \end{aligned}$$

Adesso troviamo un limite superiore per $\sum_{n \geq 1} OPT_2^*(P_n)$. Per ogni $n \geq 1$, sia L_n l'insieme di job schedulati in S alla fine di P_n , ovvero al tempo $nk - 1$. Per ogni tempo t , ogni job $i \in J_{t,2}^*$ è contenuto in qualche insieme L_n con $nk - 1 < t$. Dunque $\sum_{n \geq 1} OPT_2^*(P_n)$ è limitato superiormente dal valore totale ottenuto nello schedulare i job di L_n in S^* a tempi $t > nk - 1$, $\forall n \geq 1$. Il valore massimo possibile ottenibile nello schedulare qualsiasi job $i \in L_n$ a tempi $t > nk - 1$ è al massimo $\beta^{nk}/(1 - \beta) \cdot v_i$, che è ottenuto se il job è schedulato per una durata infinita a partire dal tempo nk . Concludiamo affermando che

$$OPT^*(I_k) \leq 2ALG(m)_k(I_k) + \sum_{n \geq 1} \frac{\beta^{nk}}{1 - \beta} \sum_{i \in L_n} v_i$$

e

$$\frac{OPT^*(I_k)}{ALG(m)_k(I_k)} \leq 2 + \left(\sum_{n \geq 1} \frac{\beta^{nk}}{1 - \beta} \sum_{i \in L_n} v_i \right) / \left(\sum_{n \geq 1} ALG(m)_k(P_n) \right)$$

. Mostriamo infine che, per ogni $n \geq 1$, vale $(\frac{\beta^{nk}}{1 - \beta} \sum_{i \in L_n} v_i) / ALG(m)_k(P_n) \leq \beta^k / (1 - \beta^k)$. Ciò implica che $OPT^*(I_k) / ALG(m)_k(I_k) \leq 2 + \beta^k / (1 - \beta^k) = 1 + 1/(1 - \beta^k)$ e il lemma segue.

Consideriamo di nuovo una qualsiasi fase P_n . Ogni $i \in L_n$ è schedulato in S alla fine di P_n , cioè al tempo $nk - 1$. Nel descrivere $ALG(m)_k$ abbiamo sostenuto che in ogni fase e su ogni macchina i job sono sequenziati in ordine non crescente di valore unitario. Quindi su ogni macchina in cui un job $i \in L_n$ è schedulato in S , i job con un valore unitario di almeno v_i sono sequenziati per tutta la fase P_n . Poiché P_n è costituita da istanti di tempo $(n - 1)k, \dots, nk - 1$, otteniamo $ALG(m)_k(P_n) \geq (\beta^{(n-1)k} - \beta^{nk}) / (1 - \beta) \cdot \sum_{i \in L_n} v_i$ e, come desiderato, $(\frac{\beta^{nk}}{1 - \beta} \sum_{i \in L_n} v_i) / ALG(m)_k(P_n) \leq \beta^{nk} / (\beta^{(n-1)k} - \beta^{nk}) = \beta^k / (1 - \beta^k)$. \square

Siamo pronti a dimostrare il Teorema 4.

Dimostrazione del Teorema 4 I Lemmi 11 e 12 implicano $(1 + 1/(1 - \beta^k))ALG(m)_k(I_k) \geq OPT^*(I_k) \geq OPT(I_k)$, per ogni input I . Poiché $ALG(m)_k(I) = ALG(m)_k(I_k)$ e $1/\beta^{k-1}OPT(I_k) \geq OPT(I)$, il teorema è dimostrato. \square

Determiniamo infine la scelta migliore di k . Per ogni β , il rapporto competitivo del Corollario 5 è limitato superiormente da $(1 + 1/(\sqrt{2} - 1))/(2 - \sqrt{2}) = 1/(3 - 2\sqrt{2}) \approx 5.828$.

Corollario 5 Per $k = \lceil \log_\beta(2 - \sqrt{2}) \rceil$, l'algoritmo risultante $ALG(m)_k$ ottiene un rapporto competitivo di $(1 + 1/(1 - \beta(2 - \sqrt{2}))) / (2 - \sqrt{2})$.

Dimostrazione La funzione $f(x) = \frac{1}{\beta^{x-1}}(1 + \frac{1}{1-\beta^x})$ è minimizzata per $x^* = \log_\beta(2 - \sqrt{2})$. Inoltre cresce per $x \geq x^*$. Pertanto, per $k = \lceil \log_\beta(2 - \sqrt{2}) \rceil$, il rapporto competitivo di $ALG(m)_k$ è limitato superiormente da $f(x^* + 1) = (1 + 1/(1 - \beta(2 - \sqrt{2}))) / (2 - \sqrt{2})$. \square

7 Implementazioni

In questa sezione verranno mostrate e discusse delle possibili implementazioni dei tre algoritmi precedentemente introdotti. Tali implementazioni sono state progettate al fine di poter ottenere il valore atteso di ciascun algoritmo, in modo tale da poter verificare empiricamente i risultati ottenuti dalle analisi descritte sopra. Il linguaggio di programmazione utilizzato è Java. Inoltre sono state fatte le seguenti assunzioni: per ogni job i , $a_i \in \mathbb{N}$ mentre v_i e l_i sono interi positivi diversi da 0. Le fasi sono numerate a partire da 0. Per quanto riguarda $ALG(m)_k$, è stato assunto che m sia un intero positivo maggiore o uguale a 1. Inoltre ogni job è stato implementato attraverso la classe Job, contenente i seguenti campi: int index, che rappresenta l'indice del job i ; int arrivalTime, che rappresenta il suo tempo di arrivo a_i ; int value, che rappresenta il suo valore unitario v_i ; int length, che rappresenta la sua lunghezza l_i .

7.1 Implementazione di $ALG1_k$

L'algoritmo $ALG1_k$ è stato implementato attraverso la classe ALG1k. I suoi campi sono:

- double beta, che rappresenta la probabilità β ;
- int k, che rappresenta la lunghezza di ciascuna fase;
- LinkedList<Job> totalJobs, che rappresenta l'insieme totale dei job I
- LinkedList<Job> activeJobs, che rappresenta l'insieme dei job disponibili in una certa fase Q_n

Il suo unico costruttore ha come parametri k, beta e totalJobs. Il metodo “start” fa partire l'esecuzione dell'algoritmo e restituisce il valore atteso su input totalJobs, k e beta. In pratica, finché ci sono job da processare e per ogni fase n :

- Rimuove dalla lista totalJobs i job che sono arrivati prima della fase n e li inserisce nella lista activeJobs (metodo “scheduleActiveJobs”).
- Ordina la lista activeJobs in ordine non-decrescente di valore unitario. I job che hanno lo stesso valore unitario vengono ordinati in ordine crescente di tempi di arrivo (metodo “sortActiveJobs”). L'algoritmo di ordinamento utilizzato è il merge-sort.
- Se activeJobs contiene almeno un elemento, estrae e rimuove il suo Job in testa j e, per ogni t compreso tra nk e $k(n + 1) - 1$, somma $\beta^t * v_j$ al valore atteso calcolato fino al quel momento, decrementa l_j di 1 e, se $l_j = 0$ e $t \neq k(n + 1) - 1$, imposta j come il job in testa ad activeJobs (se esiste), dopo averlo estratto e rimosso (metodo “processPhase”).

7.2 Implementazione di $ALG2_k$

L'algoritmo $ALG2_k$ è stato implementato attraverso la classe $ALG2k$. I suoi campi sono:

- double beta, che rappresenta la probabilità β ;
- int k, che rappresenta la lunghezza di ciascuna fase;
- LinkedList<Job> totalJobs, che rappresenta l'insieme totale dei job I
- LinkedList<Job> activeJobs, che rappresenta l'insieme dei job disponibili in una certa fase Q_n
- Job interruptedJob, che rappresenta l'eventuale job da dover ripristinare in una determinata fase.

Il suo unico costruttore ha come parametri k, beta e totalJobs. Il metodo “start” fa partire l'esecuzione dell'algoritmo e restituisce il valore atteso su input totalJobs, k e beta. In pratica, finché ci sono job da processare e per ogni fase n :

- Rimuove dalla lista totalJobs i job che sono arrivati prima della fase n e li inserisce nella lista activeJobs (metodo “scheduleActiveJobs”).
- Ordina la lista activeJobs in ordine non-decrescente di valore unitario. I job che hanno lo stesso valore unitario vengono ordinati in ordine crescente di tempi di arrivo (metodo “sortActiveJobs”). L'algoritmo di ordinamento utilizzato è il merge-sort.
- Se activeJobs contiene almeno un elemento, estrae e rimuove a uno a uno i job in testa ad activeJobs e li inserisce in una lista S finché non ci sono più job da estrarre oppure se la somma delle lunghezze dei job estratti è maggiore o uguale a k. A questo punto, se tra i job di S è presente interruptedJob, viene rimosso e reinserito in testa ad S. Se la somma delle lunghezze dei job in S è superiore a k e se interruptedJob è l'ultimo in S e non è l'unico in S, allora la sua lunghezza viene ridotta in modo tale che la somma delle lunghezze di tutti i job presenti in S sia uguale a k. Se invece interruptedJob non era presente in S ma era tra i job appartenenti ad activeJobs, allora viene rimosso da quest'ultima lista (metodo “createSchedule”).
- Se S contiene almeno un elemento, finché S non è vuoto, estrae e rimuove il suo Job in testa j e, per ogni t a partire da kn , somma $\beta^t * v_j$ al valore atteso calcolato fino al quel momento e decrementa l_j di 1 finché $t \geq k(n+1) - 1$ o $l_j = 0$. Se si verifica una delle due condizioni allora controlla se $t \geq k(n+1) - 1$ e se $l_j \neq 0$. In tal caso impostata interruptedJob = j lo aggiunge ad activeJobs. (metodo “processPhase”).

7.3 Implementazione di $ALG(m)_k$

L'algoritmo $ALG(m)_k$ è stato implementato attraverso la classe $ALGmk$. I suoi campi sono:

- double beta, che rappresenta la probabilità β ;

- int m , che rappresenta il numero di macchine m ;
- int k , che rappresenta la lunghezza di ciascuna fase;
- LinkedList<Job> totalJobs, che rappresenta l'insieme totale dei job I
- LinkedList<Job> activeJobs, che rappresenta l'insieme dei job disponibili in una certa fase Q_n

Il suo unico costruttore ha come parametri k , m , β e totalJobs. Il metodo “start” fa partire l'esecuzione dell'algoritmo e restituisce il valore atteso su input totalJobs, k , m e β . In pratica, finché ci sono job da processare e per ogni fase n :

- Rimuove dalla lista totalJobs i job che sono arrivati prima della fase n e li inserisce nella lista activeJobs (metodo “scheduleActiveJobs”).
- Ordina la lista activeJobs in ordine non-decrescente di valore unitario. I job che hanno lo stesso valore unitario vengono ordinati in ordine crescente di tempi di arrivo (metodo “sortActiveJobs”). L'algoritmo di ordinamento utilizzato è il merge-sort.
- Crea un array di m elementi chiamato machines e in ciascuna posizione a partire dalla prima inserisce i primi job di activeJobs, dopo averli estratti e rimossi, finché possibile. A questo punto, per ogni t compreso tra nk e $k(n+1) - 1$, accede ad ogni elemento di machines a partire dal primo fino all'ultimo disponibile e, per ciascun job j (se presente), somma $\beta^t * v_j$ al valore atteso calcolato fino al quel momento e decrementa l_j di 1. Se a questo punto $l_j = 0$, se t non è l'ultimo tempo della fase n e activeJobs contiene almeno un elemento, allora estrae e rimuove l'elemento in testa di activeJobs e lo sostituisce alla posizione di machines in cui si trovava j , altrimenti rimuove j da tale posizione (metodo “processPhase”).

8 Simulazioni

In questa sezione verranno mostrati i risultati ottenuti da alcune simulazioni effettuate sugli algoritmi precedentemente descritti. Lo scopo di tali simulazioni è quello di verificare empiricamente la validità dei teoremi dimostrati precedentemente. Ciascun algoritmo online è stato messo a confronto con un determinato algoritmo offline in base a determinate caratteristiche che verranno mostrate successivamente. La necessità di utilizzare tali algoritmi offline per il confronto nasce dall'impossibilità di determinare con certezza l'implementazione di un algoritmo offline ottimale per la risoluzione del problema dello storyboard scheduling.

8.1 Test effettuati su $ALG1_k$

Prima di mostrare i risultati ottenuti, occorre definire l'algoritmo offline utilizzato per il confronto. Tale algoritmo è *CHOP*, che è stato precedentemente descritto in Sez. 4 come l'algoritmo che permette di riprendere l'esecuzione dei job interrotti in un momento successivo. 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 . Vale dunque

$CHOP(I_k) \geq OPT(I_k)$. Inoltre dimostriamo il seguente corollario del Teorema 1.

Corollario del Teorema 1 *Per ogni input I e ogni rispettivo input k -quantizzato I_k , $\forall k \in \mathbb{N}$ e per tutte le probabilità β vale $c \cdot ALG1_k(I) \geq OPT(I) \iff c \cdot ALG1_k(I) \geq 1/\beta^{k-1}CHOP(I_k)$ con $c = \frac{1}{\beta^{k-1}(1-\beta^k)}$.*

Dimostrazione Per il Lemma 1, $\forall k \in \mathbb{N}$ e per tutte le probabilità β , vale

$$\frac{1}{\beta^{k-1}}OPT(I_k) \geq OPT(I).$$

Inoltre, durante la dimostrazione del Teorema 3, è stato dimostrato che

$$\frac{1}{1-\beta^k}ALG1_k(I) \geq CHOP(I_k).$$

Combinando queste due disequazioni otteniamo

$$OPT(I) \leq \frac{1}{\beta^{k-1}}OPT(I_k) \leq \frac{1}{\beta^{k-1}}CHOP(I_k) \leq \frac{1}{\beta^{k-1}(1-\beta^k)}ALG1_k(I) = c \cdot ALG1_k(I)$$

e dunque possiamo affermare che

$$c \cdot ALG1_k(I) \geq OPT(I) \iff c \cdot ALG1_k(I) \geq \frac{1}{\beta^{k-1}}CHOP(I_k) \quad \square$$

Grazie al risultato di questo corollario è possibile confrontare $ALG1_k$ su input I con $CHOP$ su input I_k per poter verificare che il rapporto competitivo di $ALG1_k$ sia effettivamente quello dimostrato, cioè $\frac{1}{\beta^{k-1}(1-\beta^k)}$. Adesso verranno descritti i test effettuati.

8.2 Test effettuati su $ALG2_k$

Anche per $ALG2_k$, l'algoritmo offline utilizzato per il confronto è $CHOP$. Vale infatti il seguente corollario del Teorema 3.

Corollario del Teorema 3 *Per ogni input I e ogni rispettivo input k -quantizzato I_k , $\forall k \in \mathbb{N}$ e per tutte le probabilità β vale $c \cdot ALG2_k(I) \geq OPT(I) \iff c \cdot ALG2_k(I) \geq \frac{1}{\beta^{k-1}}CHOP(I_k)$ con $c = 1/\beta^{k-1} \cdot \max\{1/\beta^{k-1}, 1/(1-\beta^{2k}), 1+\beta^{3k}/(1-\beta^k)\}$.*

Dimostrazione Per il Lemma 1, $\forall k \in \mathbb{N}$ e per tutte le probabilità β , vale

$$\frac{1}{\beta^{k-1}}OPT(I_k) \geq OPT(I).$$

Inoltre, durante la dimostrazione del Lemma 3, è stato dimostrato che

$$\max\{1/\beta^{k-1}, 1/(1-\beta^{2k}), 1+\beta^{3k}/(1-\beta^k)\} \cdot ALG2_k(I) \geq CHOP(I_k).$$

Combinando queste due disequazioni otteniamo

$$OPT(I) \leq \frac{1}{\beta^{k-1}} OPT(I_k) \leq \frac{1}{\beta^{k-1}} CHOP(I_k) \leq \frac{1}{\beta^{k-1}(1 - \beta^k)} ALG2_k(I) = c \cdot ALG2_k(I)$$

e dunque possiamo affermare che

$$c \cdot ALG2_k(I) \geq OPT(I) \iff c \cdot ALG2_k(I) \geq \frac{1}{\beta^{k-1}} CHOP(I_k) \quad \square$$

9 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)