Tesina di Decision Support Systems and Analytics

Task Scheduling

Giulia Gaglione (559057)

A.A. 2024/2025

Indice

Link del GitHub del progetto:
`https://github.com/giug2/task-chains-scheduling.git`

1 Il progetto

Il progetto si propone di affrontare un problema di allocazione di sequenze di task agent-based vincolati da capacità temporali.
L'obiettivo principale è quello di pianificare l'allocazione di tali task agli slot temporali in modo da rispettare i vincoli di capacità e sequenzialità, minimizzando al tempo stesso il costo medio dell'agente che termina più tardi, come misura di equità nel sistema.
Per risolvere il problema, si divide l'approccio in due fasi:

- una prima fase di costruzione basata su un algoritmo greedy,
- una seconda fase seguente di ricerca locale che cerca di migliorare la situazione dell'agente risultato peggiore.

In un primo momento, la strategia greedy adottata è stata un algoritmo semplice, basato su un'assegnazione round-robin dei task, ma un'analisi sulle istanze rivela che tale strategia si comporta in modo ottimale, non dando alcun margine di miglioramento alla ricerca locale. Questo risultato indica che, per le istanze considerate, l'approccio greedy riesce a produrre soluzioni già ben bilanciate, rendendo inefficace l'applicazione di tecniche di ottimizzazione successive.
Di conseguenza, è stata sviluppata una nuova strategia greedy, pensata per distribuire in modo più equo i task tra gli agenti già dalla costruzione iniziale della soluzione.
In conclusione, la tesina si propone quindi di:

- valutare l'efficacia di strategie greedy;
- misurare l'impatto della local search sulla qualità della soluzione.

2 Il dataset

Il dataset utilizzato per questo progetto è composto da un insieme di istanze del problema di allocazione, organizzate all'interno di due cartelle principali: MLE e MMLE, contenenti rispettivamente 10 istanze ciascuna.
Ogni istanza rappresenta una configurazione del problema e si articola in tre file CSV:

- `I_<id>_size.csv`: contiene due valori interi che rappresentano, in ordine, il numero di agenti e il numero di time slot disponibili per l'allocazione.
- `I_<id>_capacities.csv`: specifica la capacità disponibile in ciascun time slot, ossia ogni valore indica quante risorse possono essere allocate in quello slot.
- `I_<id>_requirements.csv`: descrive i task assegnati a ciascun agente. Ogni riga rappresenta un agente e contiene una sequenza di valori, dove ciascun valore indica la quantità di risorse richieste da un task specifico in sequenza.

Tutte le istanze rispettano la struttura del problema considerato: ogni agente ha una sequenza di task che devono essere eseguiti in ordine, e ogni task deve essere assegnato a un time slot che abbia capacità sufficiente a soddisfarne la richiesta, mantenendo l'ordine e garantendo che nessun task venga assegnato prima del precedente nella catena.

3 Una prima soluzione

In una prima fase di sviluppo, si è affrontato il problema utilizzando una strategia di risoluzione semplice, ma efficace, articolata in due fasi principali: un algoritmo greedy per la costruzione iniziale della soluzione e una successiva fase di ricerca locale per ottimizzarne la qualità.
L'obiettivo era minimizzare il costo medio sostenuto dall'agente che termina mediamente più tardi, garantendo allo stesso tempo il rispetto dei vincoli di capacità e sequenzialità.

3.1 Fase 1: Algoritmo Greedy

Il primo approccio adottato per generare una soluzione iniziale si basa su una strategia greedy semplice e deterministica.

L'algoritmo implementa una strategia greedy di assegnazione dei task agli slot temporali seguendo un criterio *round-robin*. L'algoritmo procede per "livelli" di task: assegna prima tutti i task in posizione 0 degli agenti, poi quelli in posizione 1, e così via, fino ad esaurire tutti i task disponibili.
Per ogni task, viene scelto il primo time slot disponibile, successivo al task precedente dell'agente, che abbia capacità sufficiente a soddisfare la richiesta. Una volta trovato, il task viene assegnato a quello slot e la capacità disponibile dello slot viene aggiornata.
Questa strategia garantisce il rispetto dei vincoli imposti dal problema, ossia capacità, ordine dei task, ed è estremamente efficiente dal punto di vista computazionale. Tuttavia, non tiene conto di alcun criterio di ottimizzazione globale poiché ogni decisione è presa localmente.

3.2 Fase 2: Algoritmo di Ricerca Locale

Dopo aver trovato la soluzione greedy, la seconda fase prevede un algoritmo di ricerca locale che mira a ridurre il costo dell'agente più penalizzato in una soluzione iniziale di allocazione dei task.
La strategia consiste nel selezionare l'agente con costo medio più alto e tentare di riassegnare i suoi task a time slot alternativi, compatibili con i vincoli, in modo da ridurre il suo tempo di completamento.
Per un numero massimo di iterazioni, l'algoritmo tenta di migliorare la posizione dei task dell'agente critico, iterando su tutti i suoi task assegnati.
Per ciascun task, si esplorano slot alternativi rispetto a quello attuale, e per ogni nuova assegnazione possibile viene verificata l'ammissibilità della soluzione, tenendo conto delle capacità di slot e dei vincoli tra i task. Se la modifica riduce il costo dell'agente critico, allora la nuova soluzione viene accettata come miglioramento locale, e la ricerca prosegue da quella configurazione.
In assenza di miglioramenti, l'algoritmo termina anticipatamente.
Pur limitandosi a una ricerca locale sulla configurazione corrente, la procedura è in grado di ottenere miglioramenti nei casi in cui il carico dell'agente critico possa essere ridistribuito in modo più bilanciato attraverso delle modifiche.

3.3 Risultati

La strategia combinata greedy basata su round robin + ricerca locale è stata testata su 20 istanze (10 in `MLE`, 10 in `MMLE`).
È emerso però che in tutte le istanze considerate, la ricerca locale non ha

apportato alcun miglioramento rispetto alla soluzione iniziale prodotta dalla greedy.
Questo risultato ha evidenziato due aspetti fondamentali: da un lato, la strategia greedy si è rivelata efficace nel generare soluzioni vicine all'ottimo; dall'altro, la ricerca locale, operando su soluzioni già molto buone, non ha avuto margine d'azione.

#istanza	peggiore_greedy	costo_greedy	peggiore_locale	costo_locale	uguali?	guadagno
MLE_1	6	9,75	6	9,75	VERO	0,00
MLE_2	9	9,83	9	9,83	VERO	0,00
MLE_3	1	9,10	1	9,10	VERO	0,00
MLE_4	7	9,10	7	9,10	VERO	0,00
MLE_5	11	7,73	11	7,73	VERO	0,00
MLE_6	6	10,37	6	10,37	VERO	0,00
MLE_7	6	11,60	6	11,60	VERO	0,00
MLE_8	11	10,60	11	10,60	VERO	0,00
MLE_9	11	10,10	11	10,10	VERO	0,00
MLE_10	8	10,93	8	10,93	VERO	0,00
MMLE_1	30	27,57	30	27,57	VERO	0,00
MMLE_2	49	25,62	49	25,62	VERO	0,00
MMLE_3	33	26,10	33	26,10	VERO	0,00
MMLE_4	46	26,83	46	26,83	VERO	0,00
MMLE_5	47	28,14	47	28,14	VERO	0,00
MMLE_6	27	30,50	27	30,50	VERO	0,00
MMLE_7	46	30,97	46	30,97	VERO	0,00
MMLE_8	48	29,80	48	29,80	VERO	0,00
MMLE_9	36	28,07	36	28,07	VERO	0,00
MMLE_10	39	25,00	39	25,00	VERO	0,00

Tabella 1: Tabella dei risultati dell'approccio round robin.

Alla luce dei risultati ottenuti nella fase iniziale, è emersa la necessità di introdurre una variante dell'algoritmo greedy, capace di integrare già nella fase costruttiva una logica di minimizzazione del costo, al fine di favorire allocazioni più bilanciate tra gli agenti. Tale approccio è descritto nella sezione successiva.

4 Miglioramento proposto

Nonostante la strategia iniziale garantisse soluzioni valide, il fatto che la ricerca locale non apportasse alcun miglioramento ha sollevato il dubbio che l'allocazione fosse sì ammissibile, ma subottimale dal punto di vista dell'equità tra gli agenti. Per questo motivo, è stato progettato un nuovo algoritmo greedy con l'obiettivo di guidare la costruzione della soluzione verso un migliore bilanciamento del carico già nella fase iniziale.

4.1 Fase 1: Algoritmo Greedy

La nuova strategia greedy si discosta dall'approccio round-robin, introducendo una logica di scelta orientata alla minimizzazione del costo medio dell'agente.
L'algoritmo assegna iterativamente i task ancora non pianificati, scegliendo per ciascun task il primo slot disponibile che minimizza il costo medio dell'agente a cui il task appartiene. Tale costo è valutato simulando l'inserimento del task nello slot candidato e calcolando il costo complessivo dell'agente sulla base della nuova allocazione parziale.
Il processo prosegue fino a quando nessun task può più essere assegnato senza violare i vincoli. In particolare, per ogni task corrente di un agente, l'algoritmo scorre i possibili slot a partire da quello successivo al task precedente, per poterne garantire l'ordine, e verifica se lo slot ha capacità residua sufficiente.
In caso positivo, simula l'allocazione e calcola il costo risultante; alla fine seleziona lo slot con il costo minimo simulato.
Questa logica consente di distribuire i task in maniera più equilibrata nel tempo, evitando che un singolo agente accumuli ritardi eccessivi.
L'approccio rappresenta un compromesso tra la natura greedy e una valutazione euristica del costo futuro per ottenere assegnazioni più bilanciate tra gli agenti.

4.2 Fase 2: Algoritmo di Ricerca Locale

La fase di ricerca locale rimane invariata nella logica rispetto alla versione precedente: una volta ottenuta la soluzione dalla greedy migliorata, viene identificato l'agente con il costo medio più elevato e si tenta di migliorare la sua allocazione, compatibilmente con i vincoli.
Tuttavia, a differenza della fase precedente, la ricerca locale risulta ora più efficace, poiché il nuovo algoritmo greedy lascia maggiori margini di miglioramento e genera soluzioni più diversificate.

4.3 Risultati finali

L'introduzione della nuova strategia greedy ha portato a una maggiore variabilità tra la soluzione iniziale e quella ottimizzata tramite local search.
Differentemente all'approccio precedente, in tutte le istanze, la ricerca locale è riuscita a migliorare ulteriormente il costo medio dell'agente peggiore, confermando che la nuova greedy è meno deterministica e più flessibile rispetto alla soluzione precedente.
Nel complesso, la nuova strategia ha dimostrato di poter generare soluzioni più equilibrate e con maggiore margine di miglioramento, fornendo così un'alternativa efficace all'approccio greedy iniziale.

#istanza	peggiore_greedy	costo_greedy	peggiore_locale	costo_locale	uguali? 0,12	guadagno
MLE_1	1	8,12	0	8,00	FALSO	
MLE_2	6	6,81	1	6,77	FALSO	0,40
MLE_3	8	7,77	4	7,0	FALSO	0,77
MLE_4	5	7,80	8	7,64	FALSO	0,16
MLE_5	12	7,25	0	7,07	FALSO	0,18
MLE_6	5	8,68	1	8,00	FALSO	0,68
MLE_7	5	9,06	9	8,50	FALSO	0,56
MLE_8	10	7,70	4	7,53	FALSO	0,17
MLE_9	4	8,41	10	8,38	FALSO	0,40
MLE_10	5	8,65	4	8,25	FALSO	0,40
MMLE_1	48	25,14	20	25,00	FALSO	0,14
MMLE_2	39	23,81	33	22,45	FALSO	1,35
MMLE_3	48	24,05	26	23,41	FALSO	0,64
MMLE_4	25	23,91	21	23,75	FALSO	0,16
MMLE_5	33	24,90	26	24,50	FALSO	0,40
MMLE_6	37	24,65	39	23,71	FALSO	0,94
MMLE_7	45	25,90	17	25,62	FALSO	0,28
MMLE_8	25	24,57	19	24,19	FALSO	0,38
MMLE_9	19	25,26	46	24,38	FALSO	0,88
MMLE_10	43	23,70	37	23,35	FALSO	0,35

Tabella 2: Tabella dei risultati dell'approccio smart.

5 Conclusioni

Il progetto ha affrontato in modo sistematico il problema dell'allocazione equa di task sequenziali su risorse temporali limitate, proponendo una soluzione in due fasi basata su una strategia greedy seguita da una ricerca locale.
I risultati iniziali hanno mostrato come un approccio greedy semplice, pur senza considerare alcuna metrica di bilanciamento globale, fosse già in grado di produrre soluzioni competitive, al punto da rendere la ricerca locale inefficace.
Questa osservazione ha motivato lo sviluppo di una strategia greedy alternativa, più consapevole del carico degli agenti, che ha prodotto soluzioni più bilanciate e ha ampliato il margine di intervento della local search.
Nel complesso, il lavoro ha evidenziato l'importanza di una fase costruttiva solida e ha mostrato come anche piccole variazioni nell'algoritmo greedy possano incidere significativamente sulla qualità finale della soluzione.

6 Sviluppi futuri

A partire dai risultati ottenuti, sono diversi i possibili sviluppi futuri che risultano promettenti.
In primo luogo, si potrebbe estendere la strategia greedy introducendo elementi di randomizzazione controllata, con l'obiettivo di esplorare soluzioni iniziali più diversificate.
Un altro percorso interessante riguarda l'ottimizzazione multi-agente, ossia, anziché focalizzarsi esclusivamente sull'agente peggiore, la ricerca locale potrebbe essere generalizzata per migliorare iterativamente più agenti, o ridurre direttamente la varianza tra i costi medi.
Inoltre, sarebbe utile valutare l'efficacia degli algoritmi su istanze di dimensioni maggiori o con distribuzioni più irregolari di capacità e task, per testarne la scalabilità.
Infine, si potrebbero integrare tecniche meta-euristiche, quali simulated annealing o tabu search, che potrebbero portare a ulteriori miglioramenti in termini di qualità e robustezza delle soluzioni.