Backtracking and MAC-based Job-Shop Scheduling Solver

MARIA LI

November 2024

1 Job Shop Scheduling come CSP

Un problema di soddisfacimento di vincoli (CSP) consiste nell'assegnare valori a un insieme di variabili, rispettando una serie di vincoli. Ogni variabile ha un dominio di valori, e l'obiettivo è trovare una soluzione che soddisfi tutte le restrizioni imposte. Il problema di Job-Shop Scheduling è un esempio di CSP, in cui i task devono essere assegnati a risorse, come macchine, lavoratori, rispettando vincoli di sequenza e disponibilità. L'obiettivo è ottimizzare la pianificazione per minimizzare il tempo totale di completamento, garantendo che tutte le restrizioni siano rispettate.

2 Pro e Contro dell'uso di Backtracking

Il backtracking è una tecnica di ricerca esaustiva che esplora tutte le possibili soluzioni, annullando e riprovando nel caso in cui un vincolo venga violato. Nel Job-Shop Scheduling, consente di valutare tutte le assegnazioni dei task alle risorse, calcolando la durata totale e rispettando i vincoli di precedenza tra i task. Sebbene garantisca una soluzione, se esiste, non è efficiente poiché il tempo di esecuzione cresce esponenzialmente con l'aumento di task e vincoli.

3 Uso combinato di MAC con Backtracking

Una tecnica più efficiente consiste nel combinare il backtracking con la consistenza sugli archi (MAC) che riduce i domini delle variabili eliminando combinazioni non valide e mantenendo la consistenza degli archi. In questo modo, invece di esplorare tutte le soluzioni, MAC restringe i domini, scartando in anticipo le combinazioni che violano i vincoli di precedenza.

4 Implementazione

L'implementazione del codice utilizza una combinazione di backtracking e MAC per risolvere il problema di Job-Shop Scheduling. Le principali scelte implementative includono:

- **Argomenti da linea di comando**: Utilizzati per impostare tramite la linea di comando la durata dei task e il tempo massimo di completamento;
- **Vincoli di precedenza**: Ogni task ha una lista di dipendenze che garantisce che i task successivi non possano iniziare prima del completamento di quelli precedenti;
- **Vincolo disgiuntivo**: Per i task *AxleF* e *AxleB*, è stato implementato un vincolo disgiuntivo che garantisce che non possano sovrapporsi nel tempo;
- **Domini delle variabili**: Per ogni task, è inizializzato un dominio che rappresenta gli intervalli di tempo validi per l'inizio del task. Con MAC, questi domini vengono progressivamente ristretti;
- **Funzione di backtracking con MAC**: La funzione backtrack_mac esplora ricorsivamente le assegnazioni possibili dei task, utilizzando MAC per applicare i vincoli di precedenza e ridurre lo spazio di ricerca.

Gli elementi principali sono le funzioni apply_precedence_mac e backtrack_mac.

- **apply_precedence_mac**: La funzione apply_precedence_mac è responsabile dell'applicazione dinamica dei vincoli di precedenza. In particolare, quando un task viene assegnato a una certa unità di tempo, questa funzione aggiorna i domini di partenza dei task successivi, rimuovendo tutti i valori non compatibili. Inoltre, se l'assegnazione di un task porta a un dominio vuoto per un task successivo, significa che una sequenza temporale valida non può essere più trovata, e quindi la soluzione corrente viene scartata.
- **backtrack_mac**: La funzione backtrack_mac è l'algoritmo principale che esplora ricorsivamente tutte le assegnazioni possibili di task. Utilizza la strategia di "*Minimum Remaining Values*" (MRV), selezionando ogni volta il task con il dominio più ristretto, ossia quello che ha il numero minore di possibili valori per il suo tempo di inizio. Dopo aver selezionato un task, la funzione applica i vincoli di precedenza e disgiuntivi, cercando di assegnare un tempo di inizio che rispetti le restrizioni. Se l'assegnazione porta a una soluzione valida, la soluzione viene aggiunta alla lista; in caso contrario, la funzione ripristina i domini precedenti ed esplora altre possibilità. Questa combinazione di backtracking e MAC ottimizza il processo di ricerca, limitando l'esplorazione di assegnazioni che non possono portare a soluzioni valide.

Questo approccio ha permesso di ottenere soluzioni ottimali più velocemente rispetto al backtracking puro, grazie a una strategia di pruning dei domini delle variabili. Durante la programmazione, sono stati effettuati test di esecuzione con la libreria *time* per misurare il tempo impiegato dal solver a trovare una soluzione. Sebbene la misurazione del tempo non sia inclusa nella versione finale, si è osservato che aumentando solo il limite massimo di completamento oltre 37 (senza variare la durata dei task), il solver impiega almeno un minuto per fornire una soluzione. Questo rallentamento è causato dall'aumento delle combinazioni esplorabili nel backtracking: con l'incremento delle combinazioni valide che rispettano i vincoli, cresce anche il tempo richiesto per identificare la soluzione ottimale.
Per verificare l'effettiva ottimalità delle soluzioni, sono stati effettuati confronti con MiniZinc, che riesce a risolvere un problema di dimensioni simili in circa 80-100 millisecondi. Sono stati utilizzati file .dzn creati con gli stessi dati del solver implementato, mantenendo invariati domini, variabili, vincoli di precedenza e disgiuntivi, per eseguire test comparativi diretti e verificare la correttezza delle soluzioni.

5 Applicazione e Guida alla Riproduzione dei risultati

Di seguito sono riportate soltanto le soluzioni ottimali applicate a quattro istanze distinte.

Task	**Inizio 1**	**Fine 1**	**Inizio 2**	**Fine 2**	**Inizio 3**	**Fine 3**
AxleF	11	21	12	23	11	21
AxleB	1	11	1	12	1	11
WheelRF	21	22	23	24	21	23
WheelLF	21	22	23	24	21	23
WheelRB	11	12	12	13	11	13
WheelLB	11	12	12	13	11	13
NutsRF	22	24	24	26	23	25
NutsLF	22	24	24	26	23	25
NutsRB	12	14	13	15	13	15
NutsLB	12	14	13	15	13	15
CapRF	24	25	26	27	25	26
CapLF	24	25	26	27	25	26
CapRB	14	15	15	16	15	16
CapLB	14	15	15	16	15	16
Inspect	25	28	27	30	26	30

- L'istanza 1 è sviluppata con i valori predefiniti.
- L'istanza 2 è sviluppata con una durata di Axle impostata a 11.
- L'istanza 3 è sviluppata con una durata di Wheel impostata a 2 e Inspect a 4.
- L'istanza 4 è sviluppata con tutti i valori aumentati di 1, ma non esiste alcuna soluzione per i valori impostati.

Per ottenere le soluzioni ottimali per le diverse istanze del problema, occorre eseguire il comando **python jss_solver.py** nel terminale, seguito dai parametri specificati con la sintassi:

`--nome_parametro valore`.

Ogni parametro rappresenta una durata di un task o un limite di tempo e va indicato come mostrato di seguito. Tra i parametri disponibili vi sono variabili come **duration_Axle, duration_Wheel, duration_Nuts, duration_Cap, duration_Inspect e max_time**, ciascuna seguita dal rispettivo valore desiderato.
Ad esempio, per l'istanza 3, il comando da eseguire è:

```
python jss_solver.py --duration_Wheel 2 --duration_Inspect 4
```

L'output sarà un elenco di tutte le soluzioni valide che rispettano i vincoli di precedenza e disgiuntivo. Ogni task includerà il proprio tempo di inizio e di fine, insieme al tempo totale di completamento del processo. Se il solver non trova alcuna soluzione valida, verrà mostrato il seguente messaggio nel terminale:

=====UNSATISFIABLE=====
Nessuna soluzione valida trovata.

Ogni soluzione valida viene presentata con "**Soluzione valida**" seguita da una lista di tuple, dove ogni tupla rappresenta un task, con il nome del task, il suo tempo di inizio e il tempo di fine, seguendo il formato:

[('Nome del task', Tempo di inizio del task, Tempo di fine del task),(...), ...]

Alla fine di ogni lista di task viene mostrato il tempo totale di completamento della soluzione. Se viene trovata una soluzione ottimale, essa viene riportata come "**Soluzione ottimale trovata**", seguita dalla stessa struttura di output e dal tempo totale ottimale.

Consigli Si consiglia di impostare il parametro max_time a un valore non superiore a 37, poiché un limite più alto potrebbe aumentare significativamente il tempo di risoluzione, richiedendo almeno un minuto per ottenere una soluzione.