Self-Driving Rides

Paolo Bonato

Roberto Cedolin

10 luglio 2018

1 Introduzione

Il progetto che abbiamo scelto è stato definito da Google, in particolare nella sezione HashCode: `https://hashcode.withgoogle.com/`
L'obiettivo è di fare lo scheduling di un numero variabile di corse assegnandole ad uno dei veicoli disponibili. Per implementare la risoluzione di questo problema abbiamo deciso di utilizzare un linguaggio di programmazione ad alto livello come JAVA, il quale è una conoscenza comune data la formazione dei componenti del team ed è, inoltre, facilmente portabile e supportato dai maggiori IDE.

2 Classi implementate

- **Agent**: dato un problema e un algoritmo, si occupa di applicare l'algoritmo al problema calcolando il punteggio finale (score) di una soluzione.

- **Car**: individua una serie di sottoproblemi dati dall'assegnazione delle corse. Un veicolo è implementato grazie alla lista ordinata delle corse che gli sono state assegnate.
  Questa classe è inoltre in grado di calcolare il punteggio ottenuto con l'assegnazione corrente (conoscendo però il relativo bonus), e controllare che l'assegnazione fatta rispetti i vincoli imposti dal problema.

- **Ride**: assegna un identificatore ad ogni corsa, gestisce la *startPosition* e la *endPosition*, indica il tempo minimo per la partenza ed il tempo limite per l'arrivo.
- **Problem**: comprende tutta la conoscenza iniziale (dimensione della mappa, tempo limite per l'arrivo, punti bonus, lista di corse, numero di veicoli disponibili) e altri metodi utili all'ordinamento delle corse.
- **ProblemParser**: classe responsabile della creazione del problema, prende come input un file scritto utilizzando il formato specificato nel documento di presentazione del problema.
- **Position**: aggrega la posizione verticale ed orizzontale in un'unica struttura dati e implementa metodi specifici per il problema.
- **Solution**: comprende la lista di corse assegnate a ciascun veicolo e ne calcola il punteggio totale.

3 Algoritmi implementati

Gli algoritmi sono stati implementati utilizzando una intefaccia comune **Algorithm** che permette di eseguire un algoritmo generico su di un problema di tipo *Problem*. Tutti gli algoritmi implementati vedono l'assegnazione delle corse ad un veicolo come un sottoproblema indipendente.

- **Algorithm**: interfaccia comune agli algoritmi. Espone il metodo *Solution solveProblem(Problem problem)*.
- **LinearTimeTest**: algoritmo a scopo esplorativo con complessità di tempo lineare per ogni sotto-problema.
  Ad ogni corsa disponibile viene assegnato a priori un punteggio ottenuto sommando certe caratteristiche della corsa moltiplicate per un coefficiente scelto. Le caratteristiche scelte sono: tempo di partenza minimo, tempo massimo di arrivo, distanza dal centro della mappa del punto di partenza, distanza dal centro della mappa del punto di arrivo e lunghezza del tragitto.
  Questo algoritmo porta a pessimi risultati per cui si è deciso di alzarne la complessità.

- **AbstractAlgorithm**: classe astratta per un algoritmo di complessità quadratica che per ogni sotto-problema individua la corsa migliore confrontandola con tutte le rimanenti prima di assegnarla.
  L'implementazione di come confrontare le corse è lasciata all'algoritmo concreto. Questa classe lascia inoltre aperta un'altro metodo utile per effettuare migliorie alla soluzione.

- **ConcreteAlgorithm**: classe che concretizza l'algoritmo astratto e fornisce diverse strategie di risoluzione del problema.

Strategie di selezione delle corse:

- **Miglior punteggio**: algoritmo di tipo greedy che assegna ad un veicolo la corsa con il punteggio maggiore. Nel tentativo di massimizzare il punteggio guadagnato dal veicolo, è legittimo considerare come euristica il punteggio della corsa, in quanto non sovrastima mai il punteggio massimo ottenibile.

- **Minor spreco di tempo**: algoritmo di tipo greedy che assegna ad un veicolo la corsa con il minor spreco di tempo. Lo spreco di tempo è calcolato tramite il tempo trascorso per arrivare dalla posizione attuale alla posizione di partenza della corsa sommato eventualmente al tempo trascorso in attesa prima che la corsa possa partire. Nel tentativo di minimizzare lo spreco di tempo del veicolo, è legittimo considerare come euristica ammissibile lo spreco di tempo della corsa, in quanto non sovrastima mai lo spreco totale.

- **Minor spreco di tempo in rapporto al punteggio**: algoritmo di tipo greedy che assegna la corsa con il minor spreco di tempo in rapporto al punteggio. Non è una euristica ammissibile, in quanto può succedere che sovrastimi il costo.

- **Minor spreco con bonus**: algoritmo di tipo greedy che assegna la corsa che spreca meno tempo ma tiene conto del guadagno di punti tramite il bonus. Se la corsa ottiene un bonus il suo spreco di tempo viene considerato ridotto del valore del bonus moltiplicato per un coefficente. Si tratta di una euristica ammissibile perchè lo spreco di tempo totale non è mai sovrastimato.

Strategie di miglioramento della soluzione:

- **Nessuna ottimizzazione**: la soluzione corrente non viene ottimizzata.
- **Hill Climbing**: si calcola il punteggio corrente e si prova ad assegnare le corse rimanenti in diverse posizioni. Se il punteggio risulta migliorato (e la soluzione non viola i vincoli del problema) la corsa viene assegnata aumentando il punteggio.

4 Risultati e conclusioni

In questo progetto abbiamo utilizzato differenti algoritmi che hanno portato a risultati assai differenti in termini di complessità e di punteggio. Il punteggio è da intendere come punteggio totale ovvero dato dalla somma dei singoli punteggi ottenuti su ciascun file di test indicato nelle specifiche del progetto. Vediamo in tabella i risultati ottenuti.

Algoritmo utilizzato	Score $(1 * 10^3)$	Score con Hill-Climbing $(1 * 10^3)$
Test in tempo lineare	1 000	
Miglior punteggio	37 988	43 997
Minor spreco di tempo	39 912	40 039
Minor spreco di tempo in rapporto al punteggio	48 684	48 800
Minor spreco di tempo con bonus	38 131	39 204

Abbiamo ottenuto il punteggio massimo utilizzando l'algoritmo *Minor spreco di tempo in rapporto al punteggio* in cui la versione ottimizzata con *Hill-Climbing* ha ottenuto un punteggio di circa 48 800 000 punti.