Progetto n.8 CSP per il problema dell'orario

Costruzione di uno risolutore per il problema dell'orario scolastico con vincoli hard e soft basato su tecniche di CSP

Definizione del problema

L'orario scolastico è una tabella che coordina principalmente questi tre elementi:

1. studenti
2. insegnanti
3. unità di tempo (in generale, dette "ore")

Gli orari scolastici solitamente hanno un ciclo settimanale, e costituiscono un problema organizzativo di capitale complessità nelle scuole.
Gli orari scolastici sono strutturalmente diversi dagli orari delle lezioni universitarie. La differenza principale è il fatto che nelle scuole superiori gli studenti devono essere tenuti occupati e controllati da un insegnante in ogni ora del giorno. Inoltre, gli insegnanti delle scuole superiori sono gravati da carichi di insegnamento molto più elevati che non nel caso delle università. Di conseguenza, la compilazione dell'orario scolastico è un problema estremamente complesso dal punto di vista computazionale, e ricade nella classe dei problemi detti Problemi di Soddisfazione di Vincoli o CSP.

Problemi tipici e vincoli

Il compito di costruire un orario scolastico comporta tipicamente i seguenti problemi, ovvero la risoluzione di un grande numero di vincoli. I vincoli che possiamo individuare in un orario scolastico settimanale di una classe.

1. *Buona formazione dell'orario*. Vi è un calendario settimanale che prevede un massimo di ore (ad esempio 30: 5 al giorno per 6 giorni), e le classi devono avere le attività assegnate in modo che gli studenti non siano mai lasciati inattivi e senza custodia. Ad esempio, se una data classe ha solo 27 ore contro le 30 delle altre, vi saranno tre giorni alla settimana in cui essa uscirà un'ora prima, e uno di questi tre giorni dovrà essere il sabato; non è assolutamente ammissibile che la classe sia lasciata senza attività in un'ora intermedia.
2. *Assegnazione bilanciata delle lezioni.* Vi è la necessità di diffondere le lezioni dello stesso insegnante in tutto il ciclo settimanale di insegnamento, in maniera che le lezioni siano quanto più possibile distanziate tra loro nel tempo. Ad esempio, non sarebbe ammissibile un orario in cui cinque ore di italiano fossero assegnate tutte al lunedì: occorre dividerle in modo che l'insegnante di italiano veda la sua classe in quattro o cinque giorni.
3. *Gestione del raggruppamento delle ore*. Se un insegnante ad esempio insegna per 6 ore alla settimana in una certa classe, teoricamente queste sei ore potrebbero essere distribuite con vari raggruppamenti (6 in un solo giorno, 5+1, 4+2, 4+1+1, ...). Nella realtà delle esigenze di insegnamento, questi modi di raggruppare le ore sarebbero soggetti a valutazioni molto diverse da parte degli insegnanti anche a seconda della materia di riferimento.
4. *Il problema delle ore cosiddette "buche"*, ossia le ore di intervallo in cui gli insegnanti non sono occupati in nessuna attività. Queste ore devono essere ridotte al minimo, perché costituiscono un grave incomodo. Ma la loro riduzione complica di molto il problema computazionale.

Analisi del problema

L'approccio con gli strumenti classici della programmazione non è in grado di risolvere il problema dell'orario scolastico, e non lo sarà mai, perché l'esplosione quantitativa dei casi da analizzare rende impossibile la soluzione di ogni caso del problema in tempi ragionevolmente utili. Il problema dell'orario scolastico appartiene alla classe dei Problemi di Soddisfazione di Vincoli (Constraint Satisfaction Problem) o CSP, riguardo alla quale esistono teorizzazioni matematiche che affrontano la questione in maniera quanto più possibile generalizzata.

Per esemplificare con semplicità la natura del problema, supponiamo di avere una scuola con 40 classi, e con 30 ore settimanali ciascuna. Un programma banale per generare l'orario scolastico attraverso una tecnica combinatoria dovrebbe esaminare ogni disposizione di 30 ore su 40 classi, agendo così:

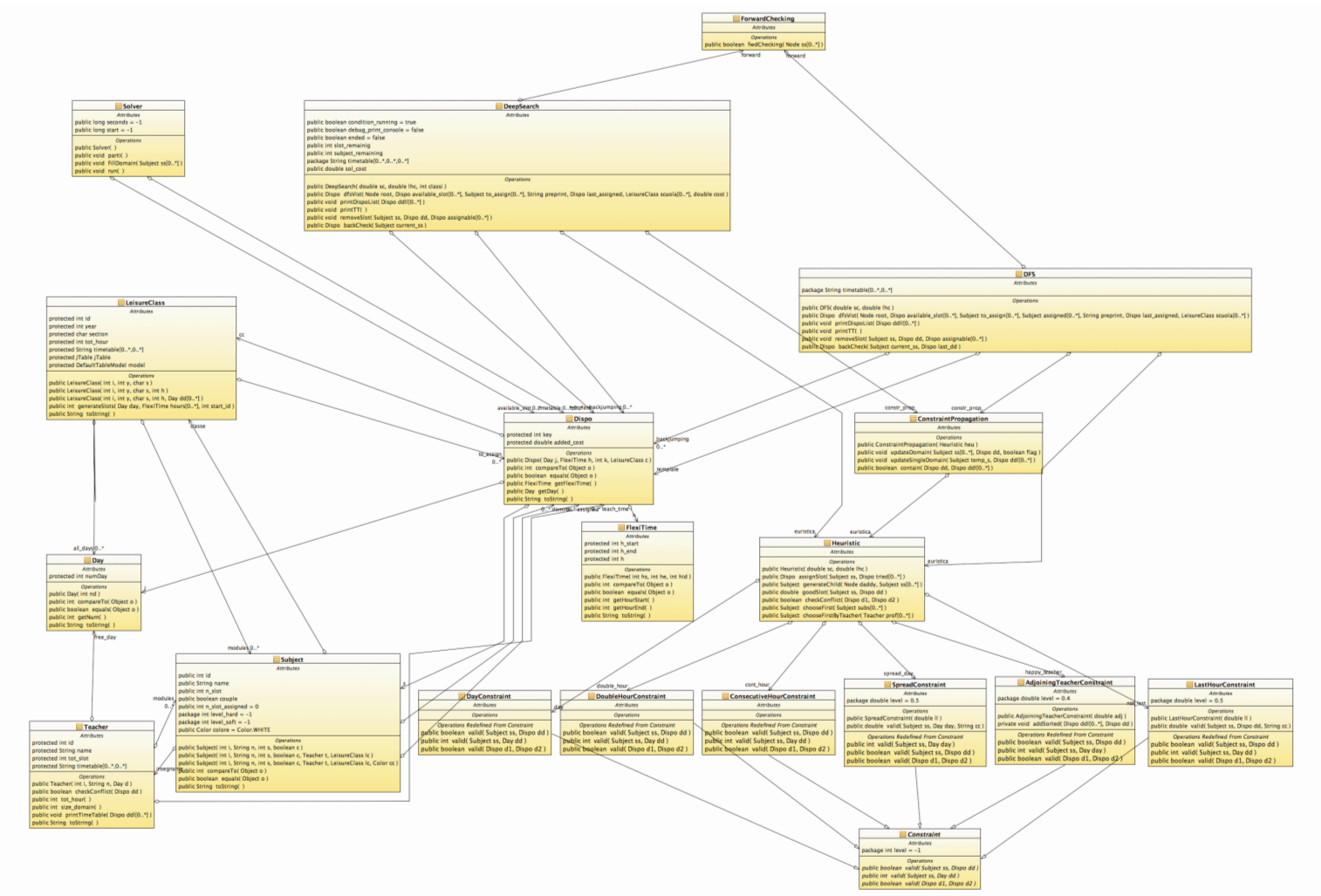
1. creare un 'cartellino' virtuale per ogni data ora della settimana (in tutto 40 x 30 = 1200 cartellini)
2. assegnare a ogni 'cartellino' un'ora di una lezione
3. esaminare se vi sono classi o insegnati assegnati due volte nella stessa ora: se sì, passare alla disposizione successiva (questa è inutilizzabile)
4. termina quando tutte le ore di tutti i corsi sono state assegnate

In questa situazione però le disposizioni da esaminare sarebbero n^k, ossia 30^40 = 1,2157 x 10^59... Ecco perché la soluzione a questa specie di problemi si trova soltanto attraverso tecniche di intelligenza artificiale, cioè attraverso metodi di calcolo che emulano la capacità di sintesi della mente umana.

Implementazione

Il progetto prevede l'utilizzo della tecnica di backtracking con backjumping per cercare di trovare la soluzione che possa soddisfare i vincoli prestabiliti.

Come e' visibile dalla figura della struttura dell'UML del progetto (che non contiene le classi che si occupano della visualizzazione grafica), questo e' molto complesso, percio' non ci addentreremo molto nel dettaglio della descrizione di tutte le classi.



Strutture dati

Per rappresentare una slot, cioe' una disposizione disponibile per un corso all'interno di un orario scolastico, utilizziamo 3 classi:

- FlexiTime: permette di rappresentare uno slot dell'orario giornaliero; ogni oggetto corrisponde ad un'ora del giorno.
- Day: rappresenta uno dei giorni della settimana.
- Dispo: le istanze di questa classe rappresentano una disposizione della settimana: e' costituita dal giorno e dall'ora del giorno. Esiste un'istanza di questa classe per ogni slot dell'orario scolastico settimanale.

Per la rappresentazione degli altri elementi abbiamo utilizzato una classe ciascuno:

-LeisureClass: serve a rappresentare una classe e principalmente contiene l'orario definitivo della classe, la lista dei corsi della classe, la lista delle disposizioni dell'orario.

-Teacher: rappresenta un insegnante, incluso il nome, giorno libero preferito e moduli insegnati.

-Subject: rappresenta una materia/corso/modulo che dir si voglia, con il nome, numero di ore, classe a cui appartiene e il puntatore all'istanza dell'insegnante che si occupa della materia.

Infine, per ultimo ma non meno importante abbiamo la classe Node la cui istanza rappresenta un nodo. Questo contiene le informazioni necessarie per la costruzione dell'albero di ricerca.

Constraint

Esiste una classe per ogni vincolo non base (o banale) del problema. I vincoli base sono hard e sono quelli che indicano che una disposizione puo' essere assegnata solo ad un corso per volta (non piu' di un corso per ora), che un insegnante non puo' trovarsi in piu' classi contemporaneamente.

Ogni classe eredita da una classe astratta Constraint che definisce i metodi base di un vincolo. Ogni classe che eredita, quindi ogni vincolo deve implementare almeno una delle funzioni che servono per la verifica di soddisfacibilita' del vincolo ed eventualmente nel caso di vincolo soft, per verificarne il costo. I 'constraint' specificati sono:

ConsecutiveHourConstraint (hard): non possono esserci ore di uno stesso corso distribuite in modo non contiguo all'interno della giornata.

DayConstraint (hard): verifica che non ci siano piu' ore di quelle ammesse per un corso nella stessa giornata.

DoubleHourConstraint (hard): verifica se il corso ammette piu' di due ore consecutive nello stesso giorno.

AdjoiningTeacherConstraint (hard): un insegnante deve avere un orario compatto, quindi non deve avere un numero di ore di buco sproporzionato rispetto alle ore che insegna.

LastHourConstraint (soft): calcola il costo dell'avere un corso sempre alla stessa ora in giorni diversi (ad esempio avere un corso sempre all'ultima ora)

SpreadConstraint (soft): calcola il costo di avere una materia distribuito equamente nei giorni della settimana.

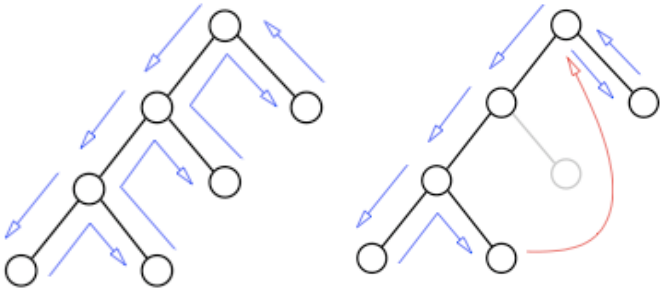
Tecnica di ricerca della soluzione

Il backtracking è una tecnica per trovare soluzioni a problemi in cui devono essere soddisfatti dei vincoli. Con questa tecnica si considerano successivamente tutte le possibili soluzioni, scartando man mano le condizioni che non soddisfano i vincoli. La tecnica adottata e' quella classica che consiste nell'esplorazione di strutture ad albero e tenere traccia di tutti i nodi e i rami visitati in precedenza, in modo da poter tornare indietro al più vicino nodo che conteneva un cammino ancora inesplorato nel caso che la ricerca nel ramo attuale non abbia successo. I nodi a profondità uguale rappresentano i possibili valori di una variabile. Il backtracking ha una complessità esponenziale, quindi è poco efficiente nell'affrontare problemi che non siano NP-completi. In generale, comunque, l'algoritmo integra euristiche che permettono di diminuirne la complessità.

L'utilizzo del backtracking standard (ossia di quello cronologico) presenta tre svantaggi principali:

– può causare thrashing ossia il ripetersi di una situazione di fallimento dovuto alla stessa causa. Il thrashing si verifica poichè il BT non identifica la vera causa del fallimento, cioè le variabili in conflitto , causando vari fallimenti durante la ricerca. Il thrashing può essere evitato usando il

backjumping (detto backtracking intelligente) ossia uno schema in cui il backtracking è eseguito sulla variabile che ha causato il fallimento.
– causa lo svolgimento di lavoro inutile (redundant work) : anche se i valori in conflitto sono identificate mediante backtracking intelligente esse non sono memorizzati per l'identificazione successiva dello stesso conflitto nei passi successivi della computazione. Per risolvere tale problema si ricorre al backchecking o al backmarking. Esiste anche un tipo di backtracking che risolve tutti e due i difetti riportati : il backtracking dependency-directed . E' importante notare che l'utilizzo di tecniche avanzate di backtracking porta con sè un certo costo computazionale che dev'essere bilanciato da una serie di vantaggi fra i quali il più importante deve essere l'aumentata efficienza .
– infine rileva i conflitti in ritardo; sarebbe necessaria l'identificazione tempestiva dei conflitti prima ancora che questi si verifichino. A questo scopo si possono applicare certe tecniche di consistenza che evitano il problema (tecniche di forward check).

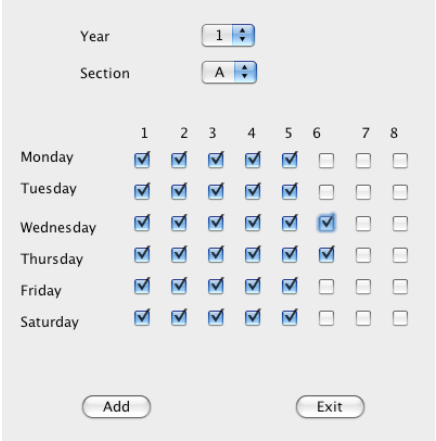


Il backjumping è una tecnica che consente di evitare il verificarsi di thrashing nel backtracking cronologico. Ha stesso comportamento del backtracking cronologico; la differenza risiede nel comportamento durante l'esecuzione del backtracking: questo cerca di identificare la variabile che causa il conflitto, usando i vincoli non soddisfatti. Se tutti i valori nel dominio sono già stati assegnati esegue backtracking rispetto alla variabile in conflitto individuata. Nel caso specifico viene mantenuta la lista delle assegnazioni del cammino dell'albero attualmente percorso, e in caso in cui non c'e' una disposizione che soddisfi l'assegnamento di un corso, allora viene controllata a ritroso la lista delle assegnazioni di disposizioni che sono in conflitto (in base ai vincoli specificati sopra, sia da un punto di vista dell'insegnate, che della classe stessa).

Oltre al backjumping il progetto include la tecnica del constraint propagation che permette di potare in anticipo i sottoalberi che non portano ad una soluzione che soddisfi i vincoli specificati. Nel caso particolare ogni corso e' dotato di un dominio di disposizioni che gli possono essere assegnate, e per ogni assegnaizione effettuata il proprio dominio viene aggiornato. Questa tecnica permette di mantenere il dominio di un corso il piu' piccolo possibile, quindi ottimizzando la complessita' spaziale, a discapito della complessita' temporale che ne risente (per ogni assegnamento per ogni corso tutto il dominio deve essere verificato).

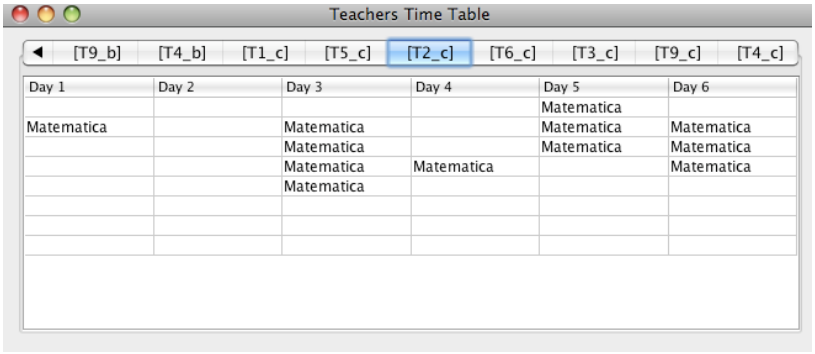
Euristiche
Le euristiche applicate sono quelle del:
"Minimo insieme di valori rimanenti" per la scelta della variabile da assegnare, scegliendo il corso con il minor numero di ore che gli possono venire assegnate.
"Valore meno vincolante" per la scelta del valore da assegnare alla variabile, andando a scegliere la disposizione da assegnare al corso in base a quanto questa limiti il dominio degli altri corsi, ma anche in base al costo della scelta della disposizione particolare (viene scelta la disposizione a costo migliore a parita' di valore meno vincolante).

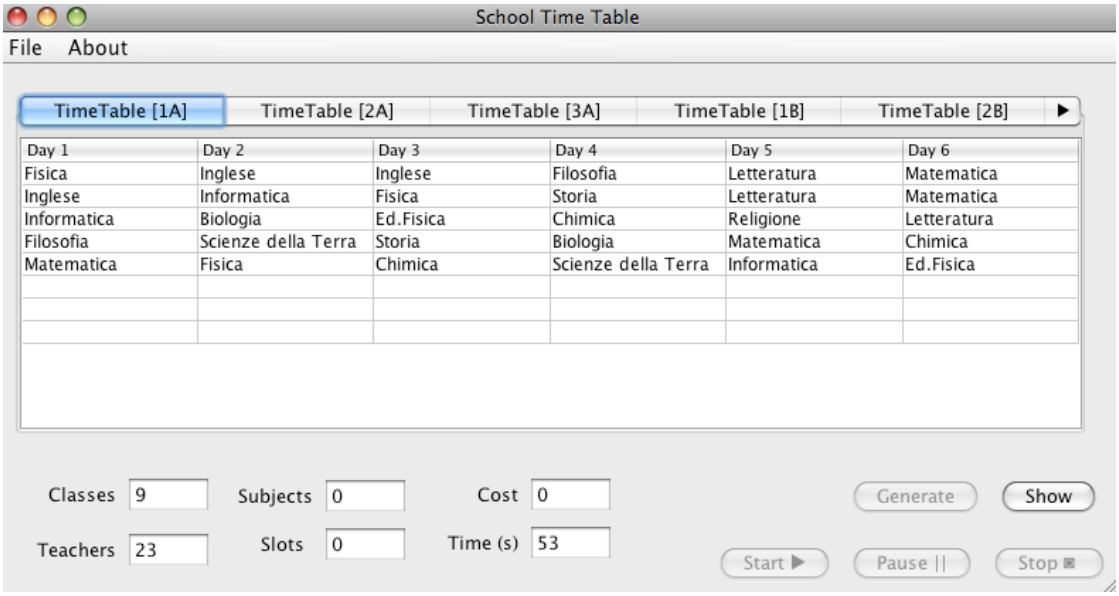
Interfaccia Grafica
L'interfaccia grafica e' essenziale e consente sia l'inserimento dei valori di input, quali gl'isegnanti, i corsi e le classi e la loro combinazione. Inoltre per ogni classe e' possibile scegliere l'orario di input che andra' a costituire l'insieme delle disposizioni da assegnare ad ogni corso. Qui e' possibile selzionare per ogni corso i giorni di lezione e per ogni giorno selezionare quali ore faranno parte dell'orario (sia in modo contiguo che non).



Per quanto riguarda la visualizzazione dell'orario calcolato vi sono una serie di tabelle ognuna rappresentante l'orario della classe.

Oltre all'orario per ogni classe e' possibile visualizzare l'orario da un punto di vista dell'insegnante.





Test
I test effettuati dimostrano che con 9 classi, ogni classe con 30 ore di lezione, 23 insegnanti e 12 corsi per classe, il tempo per trovare la soluzione e' mediamente di circa ~5 minuti.

Conclusioni
Possibili miglioramenti potrebbero prevedere un'ottimizzazine generica della complessita' temporale del progetto andando a sviluppare delle euristiche piu' specifiche e la memorizzazione di assegnazioni che hanno portato a fallimenti.
Dal punto di vista dei costi potrebbe essere migliorata la gestione degli stessi, con la possibilita' di scegliere dall'interfaccia grafica il limite desiderato (su una scala Basso-Alto).
Dal punto di vita grafico, potrebbe essere utile mostrare l'orario della scuola in un'unica tabella.
In generale il software sviluppato e' facilmente estendendbile, anche all'introduzione del concetto di classi.