IL PROBLEMA DELLA GENERAZIONE AUTOMATICA DELL'ORARIO DELLE LEZIONI: TEORIA E PRATICA

THE COURSE TIMETABLING PROBLEM: THEORY AND PRACTICE

Luca Di Gaspero e Andrea Schaerf

Sommario

In questo lavoro si descrive un'applicazione di Intelligenza Artificiale per la soluzione del problema della generazione automatica dell'orario delle lezioni di una facoltà universitaria italiana. In particolare, vengono descritti sia gli aspetti teorici riguardanti la formulazione del problema e le tecniche di soluzione impiegate, sia aspetti pratici legati all'utilizzo concreto delle soluzioni proposte e alla loro validazione sul campo.

In this paper we describe an Artificial Intelligence application for the solution of the course timetabling problem in an Italian university. In particular, we describe both the theoretical aspects regarding the formulation of the problem and the solution techniques, and the aspects related to how the proposed solutions have been implemented and validated in practice.

Parole chiave: Generazione automatica di orari, Ricerca Locale, Ricerca Tabù

1 Introduzione

Il problema della generazione dell'orario delle lezioni (CTTP: *Course TimeTabling Problem*) è un problema combinatorio che consiste nell'assegnazione settimanale delle lezioni di un insieme di corsi ai periodi e alle aule disponibili.

La soluzione manuale del CTTP tipicamente richiede diversi giorni/uomo di lavoro, ed inoltre le soluzioni trovate sono spesso insoddisfacenti; ad esempio, gli studenti sono spesso costretti a rinunciare a seguire un corso a causa di sovrapposizioni di orario con altri corsi che intendono seguire.

A causa di questa difficoltà, la possibilità di automatizzare la soluzione è stata molto studiata nelle letteratura scientifica fin dai primi anni '60 [4]. Il problema è stato considerato sia dal punto di vista teorico, con dimostrazioni di complessità computazionale (ad es. [11]), sia dal punto di vista della soluzione di casi pratici e dello sviluppo di applicazioni a livello industriale (ad es. [2]).

Le tecniche utilizzate per la soluzione variano da metodi propri della Ricerca Operativa, quali la programmazione intera [25], fino a metodi legati all'Intelligenza Artificiale quali il soddisfacimento di vincoli [15] e la ricerca locale [16]. La fonte più completa sullo stato dell'arte in questo campo sono gli atti della serie di conferenze PATAT [21], dedicata a questo argomento, oltre ad alcuni articoli di rassegna (ad es. [3, 23]).

In questo lavoro affrontiamo il problema specifico di una facoltà universitaria italiana. Nel nostro paese la recente transizione al nuovo ordinamento didattico (il cosiddetto "3+2") ha creato sia una frammentazione dei corsi, e quindi una maggiore difficoltà di gestione, ma anche una rivoluzione dell'offerta didattica, con la conseguente impossibilità di generare un orario semplicemente per differenza da quello dell'anno precedente. In questo contesto, si è reso quindi ancora più cruciale l'utilizzo di un risolutore automatico.

Il risolutore che abbiamo sviluppato è basato sul paradigma della ricerca locale. Questo paradigma si è dimostrato particolarmente adeguato per la soluzione di un vasto numero di problemi di generazione di orari come dimostrano le numerose pubblicazioni in tal senso (cfr. [21]). Anche l'esperienza specifica del nostro gruppo di ricerca ha portato a risultati interessanti utilizzando queste tecniche. In particolare, abbiamo sviluppato dei risolutori per altri problemi di generazione di orari [5, 7, 9, 22] che hanno dato buoni risultati anche in comparazione (ove è stato possibile) con altri risolutori presentati in letteratura.

Il risolutore descritto in questo lavoro viene impiegato dalla Facoltà di Ingegneria di Udine a partire dall'A.A. 2000-01, e si è dimostrato uno strumento indispensabile per questa difficile fase di transizione dal vecchio al nuovo ordinamento didattico. Nel corso di questi anni abbiamo avuto modo di valutare sul campo le soluzioni proposte da questo strumento che si sono sempre rivelate essere di buona qualità e soddisfacenti per i soggetti interessati.

2 Definizione del problema

Esistono numerose formulazioni del problema CTTP che differiscono tra loro principalmente per il tipo dei vincoli presi in considerazione. In questo lavoro descriviamo la formulazione che viene utilizzata nella nostra facoltà, rimandando a [23] per formulazioni alternative.

I dati fondamentali del problema sono:

I corsi: Ogni corso è erogato in un numero di lezioni settimanali da assegnare ad aule e periodi distinti, ed è seguito da un numero (stimato) di studenti.

I periodi: L'orizzonte di programmazione (nel nostro caso una settimana) è diviso in giorni e ciascun giorno è diviso in periodi di lezione. I periodi hanno tutti la stessa lunghezza, fissata a priori (nel nostro caso 2 ore).

Le aule: Le aule ospitano le lezioni ed hanno una propria capienza espressa in termini di posti a sedere.

I curricula: Un curriculum è un insieme di corsi seguiti dal medesimo gruppo di studenti. Quindi, qualsiasi coppia di corsi dello stesso curriculum ha degli studenti in comune. Un corso può appartenere anche a più curricula distinti. Un esempio di curriculum è l'insieme dei corsi del terzo anno di Ing. Civile orientamento "edilizia".

L'assegnazione cercata deve rispettate un dato insieme di vincoli *hard* e *soft*. I vincoli *hard* sono quelli che devono assolutamente essere soddisfatti dalla soluzione, mentre quelli *soft* possono essere violati al costo di peggiorare la qualità della soluzione.

I tipi di vincoli *hard* e *soft* presenti nel nostro problema sono elencati di seguito. Si noti che alcuni di essi sono considerati in entrambe le forme, nel senso che ogni singolo vincolo di quel tipo, a seconda delle esigenze, può essere dichiarato sia *hard* che *soft*.

1. **Lezioni (hard):** Il numero di lezioni di ciascun corso deve essere esattamente quello fissato.
2. **Occupazione aule (hard):** Due lezioni non possono tenersi nella stessa aula nello stesso periodo.
3. **Conflitti (hard/soft):** Le lezioni di corsi appartenenti allo stesso curriculum, oppure tenuti dallo stesso docente, devono essere erogate in periodi diversi (conflitto *hard*). Due corsi sono invece in conflitto *soft* se la loro sovrapposizione è sgradita ma non inaccettabile; ad esempio, i corsi di docenti che vorrebbero avere la possibilità di sotituirsi a vicenda vengono dichiarati in conflitto *soft*.
4. **Indisponibilità periodi (hard/soft):** I docenti possono dichiararsi non disponibili per un numero massimo fissato di periodi, e le lezioni dei loro corsi non devono tenersi in tali periodi (indisponibilità *hard*). Inoltre i docenti possono dichiarare il loro sgradimento per altri periodi, senza però la certezza di non avere lezioni assegnate a tali periodi (indisponibilità *soft*). Le indisponibilità *soft* sono pesate in modo inversamente proporzionale al numero di indisponibilità espresse dal docente.
5. **Indisponibilità aule (hard/soft):** A causa dell'assenza di particolari attrezzature (ad es. videoproiettore) alcune aule possono essere dichiarate come inutilizzabili o sgradite (*hard/soft*) per certi corsi. Inoltre, le aule possono essere non utilizzabili o preferibilmente non utilizzate (*hard/soft*) in certi periodi specifici.
6. **Preassegnamenti (hard):** È possibile che, per ragioni organizzative, una lezione debba necessariamente tenersi in un dato periodo e/o una data aula. In questo caso, il suo assegnamento viene prefissato e non può essere modificato.
7. **Capienza aule (soft):** Il numero di studenti che segue un corso deve essere minore o uguale al numero di posti a sedere delle aule in cui le sue lezione vengono erogate.
8. **Numero minimo di giorni di lavoro (soft):** Le lezioni di un corso devono tenersi in un numero minimo di giorni specificato per ogni corso. Ad esempio, un corso che ha 5 lezioni (da due ore) settimanali, normalmente deve tenersi su almeno 4 giorni.
9. **Lezioni giornaliere per curriculum (soft):** Il numero di lezioni giornaliere per gli studenti di ciascun curriculum deve essere possibilmente compreso all'interno di un dato intervallo (tipicamente 2 come minimo e 3 come massimo, per lezioni da 2 ore).
10. **Compattezza del curriculum (soft):** Il carico di lezioni giornaliero per gli studenti che seguono un certo curriculum deve essere il più possibile compatto, evitando "buchi" tra le lezioni (fatto salvo l'intervallo per il pranzo).
11. **Lezioni consecutive del curriculum (soft):** Possibilmente ciascuno studente non deve avere più di un dato numero di lezioni consecutive.
12. **Stabilità aula (soft):** Per quanto possibile le lezioni di un corso devono tenersi tutte nella stessa aula.

La funzione obiettivo complessiva (da minimizzare) è la somma pesata delle violazioni dei vincoli *soft*. I pesi da assegnare ai vari tipi di vincoli sono configurabili dall'utente, ma hanno un coefficiente fisso legato al numero di studenti coinvolti nella violazione (ad esempio, gli studenti che non hanno posto a sedere in aula, oppure quelli che devono seguire 6 ore di lezione consecutive).

È facile verificare che il problema dell'esistenza di una soluzione ammissibile per il CTTP è una generalizzazione

del problema della colorabilità di un grafo [12, Prob. GT 4, p. 291] ed è quindi NP-completo. Di conseguenza, è estremamente improbabile che esistano degli algoritmi efficienti che garantiscano di trovare una soluzione per tutte le istanze.

3 La ricerca locale

La ricerca locale è un paradigma di ricerca e ottimizzazione che è stata introdotto circa 40 anni fa [18]. Lo studio della ricerca locale nella comunità di IA ha avuto un impulso a seguito dei lavori fondamentali di Minton *et al.* [20] e Selman *et al.* [24] nei primi anni '90. Inoltre, molte delle caratteristiche attuali delle tecniche di ricerca locale nascono direttamente da metodi propri dell'IA, quali l'utilizzo di memoria nella ricerca e l'impiego di metodi di apprendimento adattivo.

Le tecniche di ricerca locale sono intrinsecamente *non esaustive*, nel senso che non garantiscono di trovare una soluzione ammissibile (o ottima), ma esplorano lo spazio di ricerca in modo non sistematico fino a quando viene soddisfatto un criterio di arresto.

Nel seguito descriviamo gli elementi di base della ricerca locale ed alcune delle tecniche utilizzate in questo lavoro.

3.1 Elementi di base della ricerca locale

Data una istanza p di un problema di ricerca o di ottimizzazione P, si associa ad esso uno *spazio di ricerca* S. Ogni elemento $s \in S$ corrisponde ad una soluzione potenziale di p, ed è chiamato uno *stato* di p. La ricerca locale si basa su una funzione N che assegna ad ogni stato $s \in S$ il suo *vicinato* (o *intorno*) $N(s) \subseteq S$. Ogni stato $s' \in N(s)$ è detto un *vicino* di s.

Un algoritmo di ricerca locale parte da uno stato iniziale s_0, che può essere generato casualmente o ottenuto con un'altra tecnica, ed entra in un ciclo che *naviga* lo spazio di ricerca, passando da uno stato s_i ad uno dei suoi vicini s_{i+1}, fino a che uno specifico criterio di arresto non è soddisfatto.

Il vicinato di uno stato s è solitamente descritto in modo intensionale, in termini delle modifiche (chiamate *mosse*) che possono essere applicate per trasformare s nei membri di $N(s)$. Una mossa è tipicamente composta da un insieme limitato di attributi che descrivono il cambio di stato.

Dato uno stato s ed una mossa m, denotiamo con $s \circ m$ lo stato ottenuto da s applicando la mossa m. Quindi un vicinato può essere visto come un insieme di mosse potenziali, sebbene non tutte le mosse possano, in generale, essere applicate in ogni stato, perché potrebbero essere *inammissibili*, cioè portare ad uno stato fuori dallo spazio di ricerca.

Un'ulteriore componente della ricerca locale, che ha lo scopo di guidare la ricerca verso le regioni più promettenti dello spazio di ricerca, è la cosiddetta *funzione costo* f che stima la qualità dello stato. Per i problemi di ottimizzazione, f generalmente tiene conto del numero di vincoli violati e della funzione obiettivo del problema.

Una delle limitazioni principali della ricerca locale è la possibilità di rimanere intrappolati in una zona confinata dello spazio di ricerca costituita dal bacino di attrazione di un cosiddetto *minimo locale*.

Per ovviare a questo problema, che è inerente al meccanismo stesso di questo tipo di ricerca, sono state concepite diverse strategie per guidare la ricerca ad un livello di astrazione più elevato di quello della descrizione del problema. Tali strategie sono note con il nome di *meta-euristiche*, anche se il termine meta-euristica ha un significato più generale della sola ricerca locale comprendendo anche tecniche basate su altri paradigmi (quali ad esempio gli Algoritmi Genetici).

Nell'ambito della ricerca locale, le meta-euristiche definiscono principalmente la strategia da utilizzare per selezionare la mossa in ciascuno stato e per terminare la ricerca.

Due delle strategie più comunemente utilizzate sono i metodi di discesa (HC: *hill climbing*) e la ricerca tabù (TS: *tabu search*). Esse verranno descritte brevemente nei seguenti paragrafi, limitandoci alla formulazione utilizzata dal nostro risolutore. Per una trattazione più esaustiva rimandiamo ad altri lavori sull'argomento (ad es. [1, 14]).

3.2 Metodi di discesa (HC)

HC in realtà non è una singola tecnica, bensì una famiglia di tecniche basate sull'idea di eseguire solo mosse che migliorano o lasciano inalterato il valore della funzione costo f.

Tra le varie tecniche di HC proposte in letteratura noi utilizziamo la strategia cosidetta *randomizzata non-ascendente* (HC-RNA) che opera nel seguente modo: ad ogni iterazione i si seleziona una mossa casuale m_i, e se $f(s_i \circ m_i) \leq f(s_i)$ allora si assegna $s_{i+1} := s_i \circ m_i$, altrimenti si lascia lo stato invariato, cioè $s_{i+1} := s_i$.

Si noti che HC-RNA non si ferma quando raggiunge un minimo locale. Infatti, la ricerca potrebbe continuare indefinitamente muovendosi tra due o più stati di costo uguale. Per gestire questa situazione, il criterio di arresto è basato sul numero di iterazioni trascorse dall'ultimo miglioramento. In dettaglio, viene fissato un valore n tale che l'algoritmo si ferma dopo n iterazioni che non hanno migliorato la funzione costo, cioè si ferma all'iterazione j tale che $f(s_j) = f(s_{j-1}) = \cdots = f(s_{j-n})$.

3.3 Ricerca tabù (TS)

In ogni stato s_i, TS esplora esaustivamente il vicinato corrente $N(s_i)$. Tra gli elementi di $N(s_i)$, quello che mostra il valore minimo della funzione costo f diventa il nuovo stato corrente s_{i+1}, indipendentemente dal fatto che $f(s_{i+1})$ sia maggiore o minore di $f(s_i)$.

Questa scelta permette all'algoritmo di *uscire* dai minimi locali, ma crea il rischio di entrare in ciclo tra un insieme di stati vicini. Per evitare cicli infiniti viene utilizzata la cosiddetta *lista tabù*, che determina le mosse proibite. Queta lista memorizza le mosse accettate più recente e proibisce l'esecuzione delle loro mosse *inverse*.

La modalità più semplice di funzionamento della lista tabù è quello di una coda di lunghezza fissata k: quando una mossa nuova viene aggiunta alla lista, la mossa più vecchia viene eliminata. Tuttavia, in questo lavoro utilizziamo un meccanismo più generale (e più efficace) che assegna ad ogni mossa che entra nella lista un tempo di permanenza casuale scelto nell'intervallo tra due valori fissati k_{min} e k_{max}. Ciascuna mossa viene eliminata dalla lista quando il proprio tempo di permanenza è terminato. In questo modo la lunghezza della lista non è fissa ma varia dinamicamente tra k_{min} e k_{max} incrementando la robustezza dell'algoritmo.

Un'altra componente importante della TS è il cosiddetto criterio di *aspirazione* che può cancellare lo stato di proibizione di una mossa: se una mossa m porta ad uno stato il cui costo è migliore dell'ottimo corrente, allora lo stato risultante è comunque accettato come nuovo stato corrente.

Infine, la versione di TS da noi adottata include anche una forma di modifica adattiva dei pesi, nota in letteratura come *shifting penalty* [13]. Ogni componente della funzione costo viene moltiplicata per un coefficente aggiuntivo (un valore reale compreso tra 0 e 1) che varia adattivamente durante la ricerca. In particolare, quando non ci sono violazioni di quel tipo per un dato numero di iterazioni, il peso viene abbassato, mentre quando le violazioni sono presenti il peso torna ad alzarsi. In questo modo, la forma della funzione costo viene alterata continuamente permettendo alla ricerca locale di penetrare in regioni dello spazio di ricerca che altrimenti difficilmente sarebbero state raggiunte.

Come per HC, anche in questo caso la ricerca termina quando non ci sono miglioramenti della funzione costo per un numero fissato n di iterazioni consecutive.

Come commento finale sul TS menzioniamo il fatto che questa è una versione semplificata della tecnica e che versioni più sofisticate includono anche forme più complesse di proibizione e meccanismi di memoria a lungo termine. Resta il fatto che questa forma (eventualmente con qualche variante) è quella più usata nella letteratura sul TS.

3.4 Ricerca locale composta

Una delle proprietà più importanti del paradigma della ricerca locale è che tecniche diverse possono essere combinate ed alternate in modo semplice per generare algoritmi complessi. In particolare, un meccanismo molto semplice per combinare diverse tecniche è quello che noi chiamiamo strategia *token-ring*.

Dato uno stato iniziale s_0 ed un insieme di tecniche di base $t_1, \ldots, t_q$, che chiameremo *runner*, la ricerca *token-ring* esegue circolamente ciascun *runner* t_i, partendo sempre dallo stato migliore generato dall'esecuzione del *runner* precedente t_{i-1} (o t_q se $i = 1$). L'intero processo si ferma quando ha eseguito un giro (*round*) completo degli algoritmi senza ottenere nessun miglioramento da alcun *runner*, mentre i *runner* componenti si arrestano in base ai loro criteri specifici.

Si noti che la ricerca *token-ring* permette di combinare non solo tecniche diverse (ad es. HC e TS), ma anche vicinati diversi. Infatti è anche utile utilizzare la stessa tecnica, ma con vicinati diversi per i *runner*. Questa possibilità è ampliamente sfruttata in questo lavoro in cui vengono utilizzati nell'algoritmo complessivo quattro *runner* diversi che utilizzano altrettanti vicinati distinti.

L'efficacia della ricerca *token-ring*, soprattutto per due soli *runner*, è stata messa in luce da diversi autori (cfr. [14]). In particolare, quando uno dei due *runner*, diciamo t_2, non è usato allo scopo di migliorare la soluzione ma per diversificare la regione di ricerca, quest'idea prende il nome di ricerca locale *iterata* (cfr. ad es. [19]). In questo caso, l'esecuzione di t_2 è chiamata operatore di mutazione oppure mossa *kick*.

4 Applicazione

Per risolvere il CTTP tramite ricerca locale abbiamo bisogno innanzitutto di definire lo spazio di ricerca, la funzione costo, la relazione di vicinato e la soluzione iniziale.

Il nostro spazio è composto da tutte le assegnazioni in cui i vincoli *hard* dei tipi 1, 4, 5 e 6 sono soddisfatti. Sono invece ammissibili gli stati in cui i vincoli *hard* di tipo 2 e 3 sono violati, anche se sono penalizzati dalla funzione costo. La funzione costo è quindi una somma pesata delle violazioni del vincoli *hard* di tipo 2 e 3 e delle violazioni dei vincoli *soft*. In tale somma i vincoli *hard* hanno un peso molto maggiore di quelli *soft* in modo che sia sempre preferita una mossa che risolve una violazione *hard* rispetto ad una che risolve delle violazioni *soft*. I pesi sono comunque soggetti al meccanismo dello *shifting penalty* (se utilizzato) per cui durante la ricerca un vincolo *hard* può trovarsi a pesare meno di uno *soft*.

La soluzione iniziale è generata in modo casuale, in modo però da garantire il soddisfacimento dei vincoli *hard* di tipo 1, 4, 5 e 6.

4.1 Vicinati

Nel CTTP, abbiamo a che fare con l'assegnazione a ciascuna lezione di due tipi di risorse: il periodo e l'aula. Risulta quindi intuitiva la definizione di due struttre di vicinato che gestiscono separatamente ciascuna risorsa.

Il primo vicinato che utilizziamo, chiamato P, è quindi definito semplicemente dal cambio del periodo assegnato ad una lezione di un corso lasciando invariata l'aula. Il secondo, chiamato A, è definito dal cambio dell'aula ad una lezione in un dato periodo. Una mossa è considerata ammissibile se rispetta i vincoli di tipo 1, 4, 5 e 6.

Tabella 1: I 4 *runner* del risolutore per il CTTP

tecnica	N(·)	iterazioni	k_{min}–k_{max}	SP
HC	P⊕A	1'000'000	—	NO
TS	P	2'000	30–50	SI
TS	A	1'000	10–20	NO
HC	P⊗A	1'000'000	—	NO

Tabella 2: Risultati e tempi medi dei 4 *runner*

runner		guadagno	tempi
tecnica	N(·)	medio	(s)
HC	P⊕A	84.2	11.6
TS	P	232.1	62.7
TS	A	12.5	1.6
HC	P⊗A	61.7	12.9

Dati questi due vicinati di base definiamo il vicinato unione P⊕A definito dall'insieme delle mosse di un tipo o dell'altro, ed il vicinato composizione P⊗A le cui mosse invece sono date dalla sequenza di una mossa P seguita da una mossa A applicata alla stessa lezione.

4.2 Algoritmo risolutivo

Per questo problema abbiamo implementato e valutato numerosi algoritmi sia semplici sia composti. L'algoritmo che sperimentalmente ha dato i risultati migliori è un token-ring composto dai quattro *runner* mostrati nella Tabella 1, dove le colonne rappresentano rispettivamente la tecnica utilizzata, il vicinato, il numero massimo di iterazioni dall'ultimo miglioramento, la lunghezza della lista tabù (solo TS) e l'utilizzo o meno del meccanismo dello *shifting penalty*.

Intuitivamente, il primo *runner* (HC) ha il compito di trovare rapidamente i facili miglioramenti nella fase iniziale. Il secondo e il terzo sono dei TS che invece intensificano la ricerca nelle due direzioni date dai due vicinati di base. Infine, sia l'HC finale sia l'HC iniziale (nei giri successivi al primo) permettono di spostare la ricerca in nuove regioni dello spazio, fornendo la necessaria diversificazione.

La figura 1 mostra l'andamento della media del valore funzione costo (su 100 prove) degli stati finali di ciascun *runner*. La tabella 2 mostra invece i miglioramenti totali prodotti da ciascuna *runner* e i tempi medi di ciascuna esecuzione (ad esclusione del primo giro). Si noti come il TS che usa il vicinato A dia un contributo modesto, a fronte però anche di un dispendio di risorse ancora più modesto. Questo è dovuto allo scarso impatto della disposizione delle lezioni nelle aule rispetto ai vari vincoli *soft*.

5 Implementazione e risultati

Il risolutore è realizzato integralmente in C++ standard utilizzando il compilatore `gcc` (v 3.3) sia in ambiente Linux

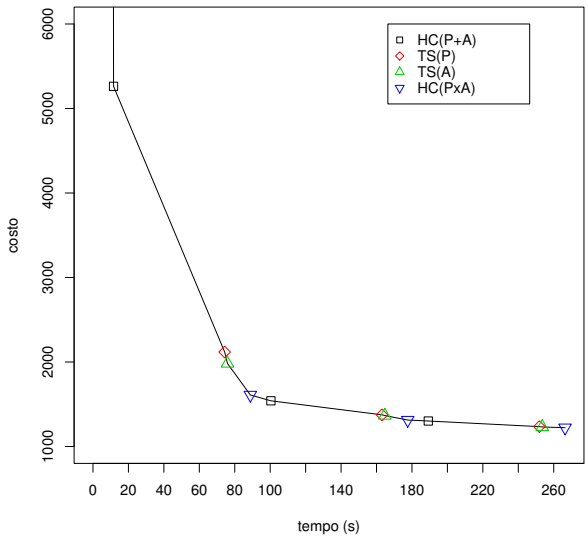


Figura 1: Andamento temporale della funzione costo

sia Windows. Il tempo medio di calcolo su istanze reali è di circa 490 secondi per ciascuna prova (su un processore Pentium 4 a 3.4 GHz).

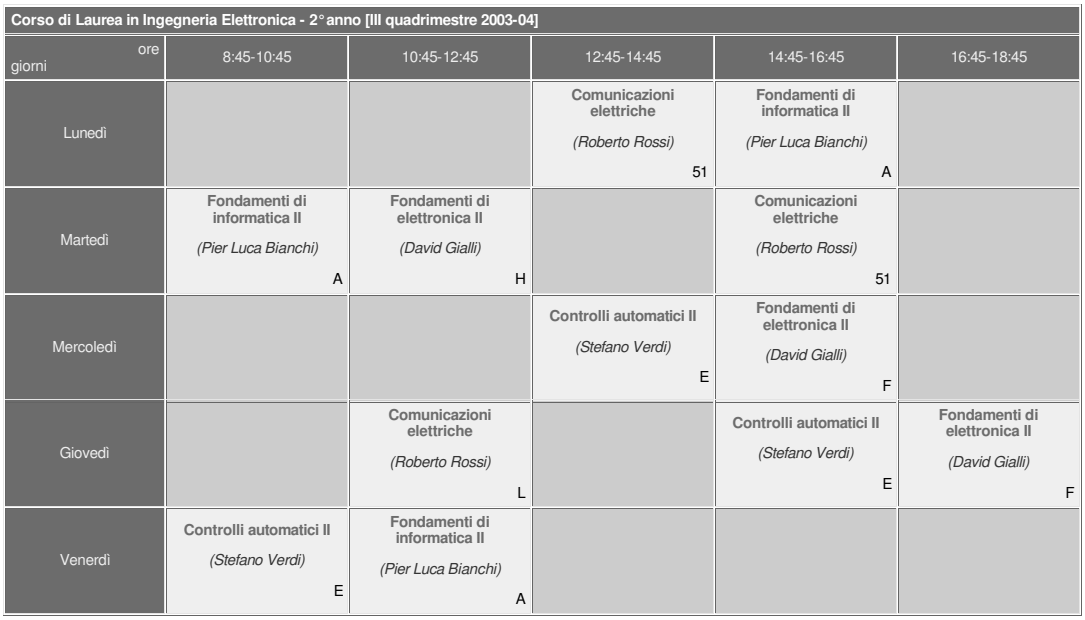
Il risolutore fa uso del *framework* EASYLOCAL++ [6, 8] che coadiuva il progetto, lo sviluppo e l'analisi di algoritmi di ricerca locale. Le classi astratte che compongono EASYLOCAL++ specificano e implementano la parte invariante dell'algoritmo, e vengono specializzate dall'utente con classi concrete che forniscono la parte del programma dipendente dal problema. EASYLOCAL++ quindi fornisce completamente le strutture di controllo degli algoritmi, e l'utente deve scrivere solo il codice specifico ed inserirlo nei punti prestabiliti. In aggiunta, EASYLOCAL++ offre un insieme di strumenti che semplificano il debugging e l'analisi degli algoritmi. Il vantaggio principale nell'usare EASYLOCAL++ è che la sua architettura fornisce una modularizzazione per la soluzione di problemi combinatori tramite ricerca locale ed aiuta l'utente a sviluppare uno schema concettuale chiaro dell'applicazione.

Per l'input/output dei dati vengono utilizzati semplici file di testo, mentre l'orario finale viene pubblicato tramite un insieme di file HTML di facile consultazione. Un esempio di file di output è mostrato in Figura 2. I vincoli e le preferenze dei docenti vengono inseriti direttamente dagli interessati attraverso una interfaccia web integrata con il sistema informativo di facoltà.

L'utilizzo pratico del risolutore prevede i seguenti passi:

1. Si collezionano i dati di ingresso dal sistema informativo della Facoltà, comprese le richieste e le preferenze dei docenti inserite via web.

2. Si esegue un numero a scelta di prove (dipendente dal tempo disponibile, tipicamente qualche centina-



Corso di Laurea in Ingegneria Elettronica - 2° anno [III quadrimestre 2003-04]

giorni \ ore	8:45-10:45	10:45-12:45	12:45-14:45	14:45-16:45	16:45-18:45
Lunedì			Comunicazioni elettriche *(Roberto Rossi)* 51	Fondamenti di informatica II *(Pier Luca Bianchi)* A	
Martedì	Fondamenti di informatica II *(Pier Luca Bianchi)* A	Fondamenti di elettronica II *(David Gialli)* H		Comunicazioni elettriche *(Roberto Rossi)* 51	
Mercoledì			Controlli automatici II *(Stefano Verdi)* E	Fondamenti di elettronica II *(David Gialli)* F	
Giovedì		Comunicazioni elettriche *(Roberto Rossi)* L		Controlli automatici II *(Stefano Verdi)* E	Fondamenti di elettronica II *(David Gialli)* F
Venerdì	Controlli automatici II *(Stefano Verdi)* E	Fondamenti di informatica II *(Pier Luca Bianchi)* A			

Figura 2: Esempio di orario di un curriculum generato dal risolutore

ia) partendo da stati iniziali casuali distinti, e si sceglie tra gli stati finali quello di costo minimo quale "bozza" dell'orario.

3. La bozza viene sottoposta ai docenti che possono evidenziare problemi e chiedere modifiche. Le modifiche richieste non vengono eseguite sull'orario, ma utilizzate per generare nuovi vincoli che vengono aggiunti ai precedenti modificando quindi l'istanza da risolvere.

4. Utilizzando la bozza come stato iniziale si esegue una nuova ricerca sull'istanza modificata e si ottiene così l'orario "definitivo".

5. L'orario definitivo viene modificato (solo eccezionalmente) in modo manuale, usando il risolutore solo come validatore dello stato e suggeritore di mosse migliorative.

Questa pubblicazione in due fasi (bozza–definitivo) è necessaria in quanto accade frequentemente che le richieste dei docenti non siano subito disponibili in modo completo e corretto. Questo è dovuto a dimenticanze ed errori di inserimento da parte dei colleghi che non sono facimente eliminabili e che vanno però tenuti in conto.

Si noti che il paradigma di ricerca locale è particolarmente indicato per il passo 4, in quanto utilizzando la soluzione precedente come stato iniziale è normalmente possibile trovare in tempi rapidi uno stato di buona qualità per i nuovi vincoli e senza stravolgere la soluzione corrente.

Il risolutore è stato finora impiegato per quattro anni accademici (12 quadrimestri) ed ha sempre trovato soluzioni ammissibili, cioè senza violazioni dei vincoli *hard*, e soddisfacenti dal punto di vista dei vincoli *soft*.

6 Discussione e conclusioni

Come osservato in precedenza, il risolutore ha dato risultati considerati soddisfacente dai soggetti interessati (preside, professori, studenti, ...). Sfortunatamente però, non è possibile dare una valutazione oggettiva delle soluzioni trovate. Infatti, da una parte, essendo il problema affrontato solo in questo lavoro, non esistono altri risultati con cui confrontarsi. Dall'altra, essendo presenti alcune componenti non-lineari (ad es. il vincolo di compattezza), lo sviluppo di un risolutore esatto che calcoli la soluzione ottima per istanze significative non è affatto semplice. Di conseguenza non è possibile calcolare la differenza rispetto all'ottimo in termini di qualità della soluzione.

Questa difficoltà di valutazione scientifica è comune a molti altri lavori di taglio pratico, di Intelligenza Artificiale e non, pubblicati nella letteratura in questo campo (per una discussione generale in tal senso cfr. [17]).

Un possibile modo per ovviare a questo problema è quello di creare istanze *ad hoc* per le quali si conosce la

soluzione ottima e valutare la differenza percentuale tra il valore ottimo ed il migliore trovato dal risolutore. Questa strada però non è del tutto soddisfacente in quanto non c'è alcuna garanzia che tali istanze artificiali mostrino lo stesso comportamento delle istanze reali che sono le uniche veramente interessanti per i nostri scopi.

In alternativa, l'algoritmo risolutivo può essere valutato in modo indiretto adattandolo a problemi teorici (semplificati) per i quali esistono istanze *benchmark* e risultati noti. Seguendo questa idea, abbiamo sviluppato due risolutori con le stesse caratteristiche per due problemi già considerati nella letteratura. Come risultato, il risolutore utilizzato per il problema dell'EXAMINATION TIMETABLING [7] ha ottenuto risultati sui benchmark tra i migliori noti in letteratura. Il risolutore [9] per la versione del CTTP utilizzato per la competizione TTPComp2002 (`www.idsia.ch/Files/ttcomp2002/`) si è classificato quarto su una ventina di gruppi partecipanti.

COme osservazione finale, è doveroso rimarcare che sono già disponibili, in Italia e nel mondo, numerosissime applicazioni sia commerciali sia gratuite per la risoluzione del problema dell'orario delle lezioni (cfr. ad es. [10]). Anche se uno studio completo su cosa offre il mercato è al di là degli scopi di questo lavoro, possiamo sicuramente affermare che queste applicazioni sono valide dal punto di vista dell'interfaccia utente, ma tendenzialmente carenti dal punto di vista della qualità delle soluzioni. Tra l'altro in quasi tutti i prodotti gli algoritimi risolutivi sono nascosti e non documentati, e quindi l'unico modo per compararli sarebbe attraverso l'esecuzione di numerose (e laboriose) prove a scatola nera.

Nel nostro caso invece sia il codice sorgente di EASYLOCAL++ che quello dell'applicazione (in una versione semplificata ma meglio documentata) sono disponibili gratuitamente per tutti gli interessati alla pagina `http://www.diegm.uniud.it/schaerf/projects/local++/`.

Ringraziamenti

Desideriamo ringraziare innanzitutto i due revisori anonimi per i loro commenti costruttivi e precisi che ci hanno permesso di migliorare il lavoro. Ringraziamo inoltre il Preside della Facoltà di Ingegneria dell'Università di Udine Andrea Stella per il supporto e l'incoraggiamento nello sviluppo di questo lavoro. Grazie anche a Erica Peressini e Mauro Rainis che hanno sviluppato l'interfaccia web per l'immissione dei dati da parte dei docenti e della segreteria.

Questo lavoro è stato supportato dal progetto "Progetto e realizzazione di un risolutore basato su tecniche di ricerca locale per l'esecuzione di specifiche dichiarative di problemi combinatori" finanziato dal MIUR (Prin 2003).

Riferimenti bibliografici

[1] Emile Aarts and Jan Karel Lenstra. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, 1997.

[2] M. W. Carter. A comprehensive course timetabling and student scheduling system at the university of Waterloo. In E. Burke and W. Erben, editors, *Proc. of PATAT-2000*, volume 2079 of *LNCS*, pages 64–82. Springer-Verlag, Berlin-Heidelberg, 2001.

[3] M. W. Carter and G. Laporte. Recent developments in pratical examination timetabling. In E. K. Burke and P. Ross, editors, *Proc. of ICPTAT-95*, volume 1153 of *LNCS*, pages 3–21, Berlin-Heidelberg, 1996. Springer-Verlag.

[4] J. Csima and C. C. Gotlieb. Tests on a computer method for constructing school timetables. *Communications of the ACM*, 7(3):160–163, 1964.

[5] Luca Di Gaspero and Andrea Schaerf. Tabu search techniques for examination timetabling. In E. Burke and W. Erben, editors, *Proc. of PATAT-2000*, volume 2079 of *LNCS*, pages 104–117. Springer-Verlag, Berlin-Heidelberg, 2001.

[6] Luca Di Gaspero and Andrea Schaerf. Writing local search algorithms using EASYLOCAL++. In Stefan Voß and David L. Woodruff, editors, *Optimization Software Class Libraries*, OR/CS series, pages 155–176. Kluwer Academic Publishers, Boston, 2002.

[7] Luca Di Gaspero and Andrea Schaerf. Multi-neighbourhood local search with application to course timetabling. In Edmund Burke and Patrick De Causmaecker, editors, *Proc. of PATAT-2002*, volume 2740 of *LNCS*, pages 262–275, Berlin-Heidelberg, 2003. Springer-Verlag.

[8] Luca Di Gaspero and Andrea Schaerf. EASYLOCAL++: An object-oriented framework for flexible design of local search algorithms. *Software—Practice and Experience*, 33(8):733–765, 2003.

[9] Luca Di Gaspero and Andrea Schaerf. A multineighbourhood local search solver for the timetabling competition ttcomp-2002. In *Proc. of PATAT-2004*, 2004. to appear.

[10] Google Directory. Scheduling utilities. URL: `http://directory.google.com/Top/Computers/Software/Educational/Administration_and_School_Management/Scheduling_Utilities`. Viewed: May, 2004.

[11] S. Even, A. Itai, and A. Shamir. On the complexity of timetabling and multicommodity flow problems. *SIAM J. of Computation*, 5(4):691–703, 1976.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability—A guide to NP-completeness*. W.H. Freeman and Company, San Francisco, 1979.

[13] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.

[14] Fred Glover and Manuel Laguna. *Tabu search*. Kluwer Academic Publishers, 1997.

[15] C. Guéret, N. Jussien, P. Boizumault, and C. Prins. Building university timetables using constraint logic programming. In E. K. Burke and P. Ross, editors, *Proc. of ICPTAT-95*, volume 1153 of *LNCS*, pages 393–408, Berlin-Heidelberg, 1996. Springer-Verlag.

[16] A. Hertz. Tabu search for large scale timetabling problems. *European J. of Operational Research*, 54:39–47, 1991.

[17] D. S. Johnson. A theoretician's guide to the experimental analysis of algorithms. In M. H. Goldwasser, D. S. Johnson, and C. C. McGeoch, editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, pages 215–250. American Mathematical Society, 2002.

[18] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44:2245–2269, 1965.

[19] Helena Ramalhino Lourenço, Olivier Martin, and Thomas Stützle. Applying iterated local search to the permutation flow shop problem. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*. Kluwer, 2001. to appear.

[20] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proc. of AAAI-90*, pages 17–24. AAAI Press/MIT Press, 1990.

[21] Practice and theory of automated timetabling (PATAT). Series of Conferences. `http://www.asap.cs.nott.ac.uk/ASAP/patat/`.

[22] Andrea Schaerf. Local search techniques for large high-school timetabling problems. *IEEE Trans. on Systems, Man, and Cybernetics*, 29(4):368–377, 1999.

[23] Andrea Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.

[24] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proc. of AAAI-92*, pages 440–446, 1992.

[25] A. Tripathy. School timetabling – A case in large binary integer linear programming. *Management Science*, 30(12):1473–1489, 1984.

7 Contatti

Luca Di Gaspero e Andrea Schaerf
Dipartimento di Ingegneria Elettrica,
Gestionale e Meccanica
Università di Udine
via delle Scienze 206
33100, Udine
email:{l.digaspero|schaerf}@uniud.it

8 Biografia

Luca Di Gaspero ha conseguito la laurea in Scienze dell'Informazione (1998) e il Dottorato di Ricerca in Informatica (2003) presso l'Università di Udine, dove è attualmente Ricercatore (senza presa di servizio). La sua ricerca riguarda l'applicazione della ricerca locale a problemi di scheduling.

Andrea Schaerf si è laureato con lode in Ingegneria Elettronica nel 1990 ed ha conseguito il Dottorato di Ricerca in Informatica nel 1994 all'Universita di Roma "La Sapienza". Dal 1998 è Professore Associato presso l'Università di Udine. Si interessa principalmente di algoritmi, linguaggi di specifica e strumenti software per problemi di scheduling e timetabling.