APPROCCI IBRIDI AL PROBLEMA DEL ROSTERING: UN CASO DI STUDIO NELL'INTEGRAZIONE DI PROGRAMMAZIONE CON VINCOLI E RICERCA LOCALE

HYBRID APPROACHES FOR ROSTERING: A CASE STUDY IN THE INTEGRATION OF CONSTRAINT PROGRAMMING AND LOCAL SEARCH

Raffaele Cipriano Luca Di Gaspero Agostino Dovier

Sommario

In letteratura esistono diversi approcci riguardo l'ibridizzazione delle tecniche di programmazione con vincoli e di ricerca locale. In questo articolo prendiamo in considerazione due di essi. In particolare studiamo l'uso della ricerca locale per migliorare una soluzione trovata per mezzo della programmazione con vincoli e lo sfruttamento di un modello a vincoli per l'esplorazione del vicinato. Questi approcci sono stati applicati al problema reale di rostering ospedaliero del dipartimento/reparto di neurologia del policlinico universitario di Udine. Riportiamo i risultati di studi computazionali degli algoritmi sia su istanze reali che su istanze casuali strutturate. I risultati mostrano i benefici degli approcci ibridi rispetto agli algoritmi che li compongono.

Different approaches in the hybridization of constraint programming and local search techniques have been recently proposed in the literature. In this paper we investigate two of them, namely the employment of local search to improve a solution found by constraint programming and the exploitation of a constraint model to perform the exploration of the local neighborhood. We apply the two approaches to a real-world rostering problem arising at the department of neurology of the Udine University and we report on computational studies on both real-world and randomly generated structured instances. The results highlight the benefits of the hybridization approach w.r.t. their component algorithms.

Keywords: Rostering, Constraint Programming, Local Search, Hybridization.

1 Introduzione

I problemi di soddisfacimento di vincoli (Constraint Satisfaction Problems, CSP) consentono di modellare molti problemi reali quali planning, scheduling, timetabling, ecc., discreti e non. Un CSP è in genere definito da un insieme di variabili associati a domini di valori possibili e da un insieme di vincoli su tali variabili; risolvere un CSP consiste nel trovare un assegnamento alle variabili che soddisfi tutti i vincoli. Qualora sia definita una funzione di costo sugli assegnamenti e si cerchi una soluzione che minimizzi tale funzione si parla di problema di ottimizzazione vincolata (Constrained Optimization Problems, COP). I metodi risolutivi per CSP e possono essere suddivisi in due categorie:

- i *metodi completi* che esplorano tutto lo spazio delle soluzioni alla ricerca di una soluzione ammissibile (CSP) o di una soluzione ottima (COP);
- i *metodi incompleti* che si basano sull'uso di euristiche, per un'esplorazione di alcune aree interessanti dello spazio delle soluzioni, con lo scopo di trovare una soluzione ammissibile (CSP) o una soluzione buona (COP).

I linguaggi di programmazione con vincoli [2] inglobano metodi completi che analizzano lo spazio di ricerca facendo uso di alternanze di fasi deterministiche di propagazione e di fasi non deterministiche di assegnamento. I metodi di ricerca locale [1] invece si basano su una definizione di prossimità (o di vicinato) ed esplorano solo specifiche zone dello spazio di ricerca.

In letteratura sono presenti due tipologie di approcci per sfruttare la combinazione delle potenzialità offerte dalla programmazione con vincoli e dalla ricerca locale [6, 7] (in [8] constraint programming e ricerca locale sono invece ibridate in un contesto più libero):

1. si effettua un algoritmo di ricerca sistematica basato sulla programmazione con vincoli, migliorato con l'inserimento della ricerca locale in qualche punto del processo di ricerca, ad esempio:
   (a) quando si raggiunge una foglia dell'albero di ricerca (quindi su assegnamenti completi) o sui

nodi interni (quindi su assegnamenti parziali), per migliorare la soluzione ottenuta, completa o parziale che sia;

(b) per restringere la lista dei figli da esplorare di un nodo dell'albero di ricerca;

(c) per generare in modo greedy un cammino da intraprendere nell'albero di ricerca;

2. si esegue un algoritmo di ricerca locale, e si usa la programmazione con vincoli come supporto alla ricerca locale, ad esempio:

(a) per analizzare il vicinato e scartare quei vicini che non soddisfano i vincoli del problema;

(b) per esplorare il vicinato relativo a un frammento della soluzione corrente;

(c) per definire la ricerca del miglior vicino come un problema di ottimizzazione con vincoli (COP).

Nel presente lavoro abbiamo adottato le due tecniche ibride 1(a) e 2(b) al problema del rostering ospedaliero. Viene trovata una soluzione ammissibile tramite programmazione con vincoli e tale soluzione viene migliorata tramite ricerca locale (approccio ibrido 1(a)). Abbiamo inoltre realizzato un algoritmo di ricerca locale che sfrutta la programmazione con vincoli nell'esplorazione di porzioni di vicinato (approccio ibrido 2(b)).

Utilizzare la programmazione con vincoli per trovare una prima soluzione ammissibile è risultato molto vantaggioso rispetto alla generazione di una prima soluzione in modo casuale. L'utilizzo della programmazione con vincoli permette di trovare in tempi molto brevi (dell'ordine di qualche secondo) una soluzione in cui i vincoli principali del problema risultano soddisfatti, dalla quale il processo di ricerca locale può partire.

Il metodo di ricerca locale che sfrutta la programmazione con vincoli ha evidenziato come la programmazione con vincoli possa essere utilizzata nell'esplorazione esaustiva di porzioni di vicinato ottenendo procedimenti di ricerca locale competitivi; è stato evidenziato che questo tipo di approccio risulta vantaggioso solo con spazi di ricerca di grandi dimensioni, in quanto per problemi piccoli l'alto overhead computazionale della programmazione con vincoli non è compensato da notevoli miglioramenti della funzione obiettivo.

Il resto del lavoro è organizzato nel seguente modo: in Sezione 2 presentiamo la formalizzazione del problema di rostering analizzato. Nella Sezione 3 viene illustrata l'organizzazione della nostra implementazione e nella Sezione 4 sono riportati i confronti tra le varie tecniche utilizzate. Alcune indicazioni per possibili lavori futuri sono discussi in Sezione 5.

2 Il Rostering Ospedaliero

Il problema del *rostering* consiste nel disporre risorse soggette a vincoli all'interno di slot temporali, cercando di trovare un'allocazione ammissibile delle risorse o di minimizzare una funzione obiettivo; le risorse in questione spesso ruotano sulla base di un elenco.

Questo problema tocca tutte le strutture sanitarie, dove si devono gestire in modo ottimale le forze disponibili, bilanciare i turni di lavoro e di pausa, rispettare le regole stabilite nei contratti sindacali, equilibrare il carico di lavoro tra i vari operatori, andare incontro alle preferenze di ciascun operatore, soddisfare le esigenze tecniche e logistiche dei reparti, garantire ai cittadini un servizio medico costante ed efficiente.

Sebbene nel corso degli ultimi decenni vi siano stati diversi studi al problema (ad esempio, mediante programmazione matematica, programmazione multi-obiettivo, programmazione con vincoli, sistemi esperti, metodi euristici e metaeuristici [3]), allo stato attuale nelle strutture ospedaliere italiane il problema viene tipicamente risolto a mano da un responsabile (*self scheduling*).

In questo lavoro abbiamo studiato una tipologia di problema di rostering che chiamiamo NEUROSTERING (in breve NR) in grado di modellare il problema di rostering relativo al dipartimento/reparto di Neurologia del policlinico universitario di Udine che è stato oggetto di studio nel presente lavoro.

Dati m medici, n giorni e k turni possibili, il problema NR consiste nell'assegnare a ciascun medico i turni da svolgere nei giorni in esame. In particolare, ogni giorno devono essere coperti alcuni turni, in base al tipo di giorno (festivo o feriale) e al giorno della settimana (alcuni turni sono richiesti solo di lunedì, altri solo di martedì, ecc.). Ad ogni turno da coprire deve essere assegnato uno ed un solo medico. Alcuni turni (quelli di carattere generale, come le urgenze) possono essere svolti da tutti i medici, altri possono essere svolti solo da un numero ristretto di medici in base alle loro competenze. Ogni medico ha una lista di giorni in cui chiede di essere esonerato dal lavoro (ferie, corsi d'aggiornamento...). Esistono infine una serie di restrizioni "temporali"che regolano la copertura dei turni: ogni turno viene svolto in una data fascia oraria; ogni medico può coprire turni consecutivi purché non superi un dato numero di ore di lavoro consecutive; tra un turno e un altro un medico deve avere un dato numero minimo di ore di pausa.

2.1 Formulazione di NR come CSP

Sia $E = \{e_1, \ldots, e_k\}$ l'insieme di tutti i possibili turni; consideriamo, per ognuno degli n giorni, gli insiemi $G_1, ..., G_n$ ove $G_i \subseteq E$ è così definito:

$$G_i = \{e_k \in E \mid e_k \text{ è un turno da coprire nel giorno } i\}$$

Ad esempio, considerando l'intervallo di tempo di una settimana e i turni UM (Urgenze Mattino), RM (Reparto

Mattina Udine) e ED (EcoDoppler). Supponiamo di voler modellare la seguente richiesta: il turno UM deve essere effettuato tutti i giorni; il turno RM dal lunedì al venerdì, il turno ED solo di martedì e giovedì. Avremo:

$$E = \{\mathsf{UM}, \mathsf{RM}, \mathsf{ED}\} \quad G_1 = \{\mathsf{UM}, \mathsf{RM}\}$$
$$G_2 = \{\mathsf{UM}, \mathsf{RM}, \mathsf{ED}\} \quad G_3 = \{\mathsf{UM}, \mathsf{RM}\}$$
$$G_4 = \{\mathsf{UM}, \mathsf{RM}, \mathsf{ED}\} \quad G_5 = \{\mathsf{UM}, \mathsf{RM}\}$$
$$G_6 = \{\mathsf{UM}\} \quad G_7 = \{\mathsf{UM}\}$$

Le competenze di ogni medico $j \in 1..m$ sono rappresentate con un insieme $M_j \subseteq E$, così definito:

$M_j = \{e_k \in E \mid e_k$ è un turno che il medico j può coprire$\}$

Consideriamo inoltre m insiemi $F_j \subseteq \{1, \ldots, n\}$ che rappresentano i giorni di ferie richiesti dal medico j.

Ad esempio, considerando 5 medici: il dott. Jekyll (1) e il dott. Jones (2) sono in grado di coprire solo i turni UM; il dott. Zivago (3) e il dott. Freud (4) sono in grado di coprire i turni UM e RM; il dott. Watson (5) può coprire i turni RM e ED. Inoltre supponiamo il dott. Jones abbia richiesto ferie per i giorni 6 e 7 e il dott. Zivago per il giorno 4. Avremo:

$$M_1 = \{\mathsf{UM}\} \quad F_1 = \emptyset$$
$$M_2 = \{\mathsf{UM}\} \quad F_2 = \{6, 7\}$$
$$M_3 = \{\mathsf{UM}, \mathsf{RM}\} \quad F_3 = \{4\}$$
$$M_4 = \{\mathsf{UM}, \mathsf{RM}\} \quad F_4 = \emptyset$$
$$M_5 = \{\mathsf{RM}, \mathsf{ED}\} \quad F_5 = \emptyset$$

Definiamo il vincolo $C \subseteq E \times E$ come l'insieme delle coppie (e_i, e_j) di turni che possono essere effettuati nello stesso giorno dallo stesso medico, cioè $(e_i, e_j) \in C$ se e solo se è consentito che lo stesso medico copra il turno e_i e il turno e_j in uno stesso giorno. Definiamo l'insieme T come la famiglia di insiemi dei turni che possono essere coperti da un medico in uno stesso giorno:

$$T = \{\{e_i\} \mid e_i \in E\} \cup \{\{e_i, e_j\} \mid (e_i, e_j) \in C\}$$

Ad esempio in uno stesso giorno lo stesso medico può coprire i turni UM e ED, oppure RM e ED, ma non UM e RM. Avremo quindi $C = \{(\mathsf{UM}, \mathsf{ED}), (\mathsf{ED}, \mathsf{UM}), (\mathsf{RM}, \mathsf{ED}), (\mathsf{ED}, \mathsf{RM})\}$ e di conseguenza:

$$T = \{\{\mathsf{UM}\}, \{\mathsf{RM}\}, \{\mathsf{ED}\}, \{\mathsf{UM}, \mathsf{ED}\}, \{\mathsf{RM}, \mathsf{ED}\}\}$$

Definiamo la relazione $S \subseteq E \times E$ come l'insieme delle coppie (e_i, e_j) di turni che possono essere coperti dallo stesso medico in giorni successivi, cioè $(e_i, e_j) \in S$ se e solo se è consentito che lo stesso medico copra il turno e_i il giorno k e il turno e_j il giorno $k + 1$.

Ad esempio supponiamo che l'unico vincolo sulla successione di due turni in due giorni consecutivi coperti dallo stesso medico sia che il giorno successivo ad un turno ED ci sia riposo obbligatorio. Avremo:

$$S = \{(\mathsf{UM}, \mathsf{RM}), (\mathsf{RM}, \mathsf{UM}), (\mathsf{UM}, \mathsf{ED}), (\mathsf{RM}, \mathsf{ED}), (\mathsf{UM}, \mathsf{UM}), (\mathsf{RM}, \mathsf{RM})\}$$

Gli insiemi appena definiti costituiscono di fatto l'input del problema.

Per codificare i vincoli del problema consideriamo per ogni $i \in 1..n, j \in 1..m$ le variabili $O_{i,j}$ a valori in $T \cup \{\emptyset\}$. Un assegnamento per tali variabili è soluzione (ammissibile) del problema se si verificano le seguenti *condizioni di ammissibilità*:

1. $\forall i \in 1..n \forall j \in 1..m \quad O_{i,j} \subseteq M_j$ (*competenze corrette*: ogni medico concorre a coprire solo i turni di cui è competente);
2. $\forall i \in 1..n \forall j \in 1..m \quad i \in F_j \Rightarrow O_{i,j} = \emptyset$ (*ferie soddisfatte*: ogni richiesta di ferie di ciascun medico è stata rispettata);
3. $\forall i \in 1..n \forall j \in 1..m \quad O_{i,j} \in T$ (*norme lavorative rispettate*: ogni medico effettua ogni giorno turni che non superano il massimo consentito di ore di lavoro consecutive);
4. $\forall i \in 1..n-1 \forall j \in 1..m \quad \forall t \in O_{i,j} \forall t' \in O_{i+1,j} \quad (t, t') \in S$ (*norme lavorative rispettate*: ogni medico ha una certa pausa tra un turno e un altro);
5. $\forall i \in 1..n \quad \cup_{j=1}^{m} O_{i,j} = G_i$ (*copertura dei turni*: ogni giorno tutti i turni richiesti per quel giorno siano coperti);
6. $\forall i \in 1..n \forall j_1 \in 1..m \forall j_2 \in 1..m (j_1 \neq j_2 \rightarrow O_{i,j_1} \cap O_{i,j_2} = \emptyset)$ (*esclusività*: ogni turno è coperto da al più un medico).

In tabella 1 è riportato un assegnamento agli insiemi $O_{i,j}$ che costituisce una soluzione ammissibile per il problema d'esempio.

$O_{i,j}$	$\rightarrow i$: giorni						
$\downarrow j$: med	1	2	3	4	5	6	7
1 Jekill	{UM }	∅	∅	{UM }	∅	∅	{UM }
2 Jones	∅	{UM }	∅	∅	{UM }	∅	∅
3 Zivago	{RM}	∅	{UM }	∅	{RM}	∅	∅
4 Frued	∅	{RM}	{RM}	∅	∅	{UM }	∅
5 Watson	∅	{ED}	∅	{RM, ED}	∅	∅	∅

Tabella 1: Soluzione ammissibile

Per codificare le variabili insiemistice $O_{i,j}$ utilizziamo $m \cdot n \cdot k$ variabili Booleane con il seguente significato: $\forall i \in 1..m \forall j \in 1..n \forall z \in 1..k \, X_{i,j,z} = 1$ se e solo se il medico i copre nel giorno j il turno z. Mediante una riduzione da 3-GRAPHCOLORING è possibile dimostrare che il problema di esistenza di una soluzione ad una istanza di NR è un problema NP-completo.

2.2 Modello di COP per NR

Abbiamo arricchito il modello per il NR introducendo una *funzione obiettivo* atta a modellare alcuni soft-constraint,

relativi a diversi aspetti di bilanciamento del carico di lavoro tra gli operatori. La funzione obiettivo è costituita da cinque componenti che descriviamo nel seguito. Useremo alcuni nomi per i turni come usati al policlinico udinese, che descriveremo.

Weekend, Notti e Guardie. Si vuole bilanciare il lavoro nei weekend. Nei weekend ci sono solo due tipi di turni: Urgenze Mattina–Urgenze Pomeriggio (UMUP) e Urgenze Notte (UN).

A tal fine si vuole individuare un'espressione che assuma un valore sempre più alto quanto più i turni sono mal distribuiti tra i medici. La nostra proposta per un medico i è di sommare tutte le variabili del medico i relative ai turni UMUP e UN nei giorni festivi e di sabato, ed elevare l'espressione al quadrato. La formula risultante è la seguente

$$\mathsf{Wk} \;=\; \sum_{i=1}^{m}\Big(\sum_{j\in GWE, z\in TWE} X_{i,j,z}\Big)^2$$

dove i indica il medico, j indica il giorno, che può variare solo nell'insieme GWE: questo è l'insieme dei giorni del mese in cui si trovano i sabati e i giorni festivi (ad esempio un mese di dicembre che inizi di lunedì avrà $n = 31$ e $GWE = \{6, 7, 13, 14, 20, 21, 25, 27, 28\}$); infine z indica il turno, che può variare solo nell'insieme TWE, costituito dai soli turni UMUP e UN.

Per bilanciare il lavoro nei turni notturni UN o nei turni UM si procede in maniera analoga definendo le formule

$$\begin{aligned}\mathsf{Nt} &= \sum_{i=1}^{m}\Big(\sum_{j\in 1..n\setminus GWE} X_{i,j,\mathsf{UN}}\Big)^2\\ \mathsf{Gu} &= \sum_{i=1}^{m}\Big(\sum_{j\in 1..n\setminus GWE} X_{i,j,\mathsf{UM}}\Big)^2\end{aligned}$$

I turni UM sono turni di guardia da coprire nei giorni feriali escluso il sabato, quindi in tutti i giorni del mese esclusi quelli dell'insieme GWE. Nei giorni di sabato e nei festivi questo turno è inglobato nel turno UMUP, considerato nella componente Wk.

Doppi. Questa componente della funzione di costo si occupa di contare i giorni in cui i medici sono al lavoro sia al mattino che al pomeriggio, situazione talvolta necessaria, ma sempre sgradita, specie da chi esercita anche la professione privata. Per contare i casi di turni 'doppi' associamo alla variabile Dp la formula:

$$\mathsf{Dp} \;=\; \sum_{\substack{i\in 1..m,\\ j\in 1..n}} \left(\sum_{z\in MAT} X_{i,j,z}\cdot \sum_{z\in POM} X_{i,j,z}\right)$$

dove MAT (risp., POM) rappresenta l'insieme di tutti i turni di mattina (pomeriggio). Il prodotto tra le due sommatorie interne alle parentesi assume valore 1 nel caso in cui un medico svolga un turno al mattino e uno al pomeriggio dello stesso giorno: ricordiamo infatti che le variabili sono booleane e che ogni medico può svolgere un solo turno al mattino e un solo turno al pomeriggio.

Consecutivi. Quest'ultima componente della funzione costo conta i casi in cui un medico compia lo stesso turno per tre giorni consecutivi; questo evento è contato col valore +1 eccetto per due turni (nello specifico denominati RMu e RMg) per cui è invece preferibile che sia lo stesso medico a coprirli in giorni consecutivi; nel caso in cui lo stesso medico copra un turno RMu o RMg per tre giorni consecutivi l'evento è contato col valore -1. Assegnamo pertanto alla variabile Cn la formula:

$$\begin{aligned}\mathsf{Cn} \;=\; &\sum_{\substack{i\in 1..m,\, j\in 1..n-2,\\ z\in E\setminus\{\mathsf{RMu},\,\mathsf{RMg}\}}} (X_{i,j,z}\cdot X_{i,j+1,z}\cdot X_{i,j+2,z}) -\\ &\sum_{\substack{i\in 1..m,\, j\in 1..n-2,\\ z\in\{\mathsf{RMu},\,\mathsf{RMg}\}}} (X_{i,j,z}\cdot X_{i,j+1,z}\cdot X_{i,j+2,z})\end{aligned}$$

dove E è l'insieme di tutti i possibili turni.

Funzione obiettivo. Ad ognuna delle componenti è stato associato un peso che indica l'importanza della componente nel rendere l'orario squilibrato. La funzione obiettivo risultante, nel caso reale studiato ove $m = 20$, $k = 28$ e $n \in 28..31$ (piano mensile dei turni, corrispondente in media a circa 16000 variabili) è la seguente:

$$\mathrm{FObj} = 50\mathsf{Wk} + 40\mathsf{Nt} + 30\mathsf{Dp} + 20\mathsf{Gu} + 10\mathsf{Cn}$$

I pesi sono stati scelti a seguito di un processo di *elicitazione della funzione obiettivo*: sono stati generati una serie di orari per i turni dell'anno 2005, per i quali si sono calcolate le varie componenti della funzione obiettivo. Tali soluzioni sono state poi mostrate al responsabile degli orari del reparto di Neurologia ed in base ai commenti emersi dal confronto si sono riadattati i pesi per tenere conto della diversa influenza di ciascuna componente nella soddisfazione globale. La versione attualmente utilizzata pare incontrare il gradimento del responsabile.

3 Realizzazione

Per risolvere il problema NR abbiamo adottato tecniche integrate di programmazione con vincoli e ricerca locale. L'idea è di ottenere una prima soluzione ammissibile tramite programmazione con vincoli, per poi migliorare la soluzione ottenuta con tecniche di ricerca locale che esplorino lo spazio delle soluzioni alla ricerca di vicini migliori; tra la varie tecniche di ricerca locale realizzate, una effettua l'esplorazione del vicinato utilizzando ancora la programmazione con vincoli.

3.1 Struttura dell'applicazione

Abbiamo quindi realizzato un sistema che risolve il problema di NR costituito da due moduli principali:

1. il modulo *FirstSolution*, costituito da un programma in SICStus Prolog usando il pacchetto `clpfd` [4]

che modella il problema di NR. Prende come input un'istanza del problema specificata dall'utente e lancia un'elaborazione che trova una soluzione ammissibile per l'istanza. Nella ricerca della soluzione, le variabili sono state raggruppate per giorni e per ogni giorno riordinate ciclicamente in modo da variare il primo elemento. Abbiamo dunque all'interno di ogni blocco di variabili utilizzato i parametri *first-fail* per la selezione della variabile e *down* per l'assegnamento del valore. Nel caso in cui non esista una soluzione ammissibile il modulo può segnalare un errore e sospendere l'esecuzione o, se possibile, rilassare parti del modello per permettere all'elaborazione di giungere al termine, segnalando comunque all'utente i problemi riscontrati.

2. il modulo *LocalSearch*, realizzato con il framework JEasyLocal (versione Java del framework EASYLOCAL++ [5]) e contenente algoritmi di ricerca locale implementati ad hoc per NR. Prende in input la soluzione ammissibile trovata dal modulo FirstSolution e la migliora, applicando un algoritmo di ricerca locale selezionato dall'utente; la soluzione ottenuta da questo modulo può essere migliorata applicando ulteriormente un algoritmo di ricerca locale e iterando il processo a piacere. Sono state implementate le tecniche Hill Climbing, Steepest Descent e Tabu Search. È inoltre stato sviluppato un algoritmo di ricerca locale in SICStus Prolog; anche questo algoritmo permette di migliorare una soluzione ottenuta dal modulo FirstSolution e adotta una strategia che ibrida la tecnica Steepest Descent con la programmazione con vincoli.

3.1.1 Ricerca Locale

Per realizzare gli algoritmi di ricerca locale abbiamo definito un concetto di mossa che permette di spostarsi da un elemento dello spazio delle soluzioni ad un altro ad esso vicino. È stato individuato il seguente concetto di mossa, che chiamiamo mossa di *scambio*:

"Scambiare i turni di due medici relativi allo stesso giorno e alla stessa fascia oraria"

Per esempio, se il dott. Freud copre il turno UM nella mattina del giorno 2 e il dott. Jones copre il turno DH (Day Hospital) nella mattina del giorno 2, una mossa possibile consiste nello scambiare i turni UM e DH, in modo che nella mattina del giorno 2 il dott. Freud copra il turno DH e il dott. Jones copra il turno UM, come mostrato in figura 1.

I turni dei due medici possono essere indifferentemente turni lavorativi o di riposo. Mosse di scambio in cui entrambi i medici hanno un turno di riposo non modificano in nessun modo la soluzione e sono dette *mosse di scambio inutili*; tutte le altre mosse di scambio sono dette *utili*.

Una mossa è quindi individuata da: i due medici che partecipano allo scambio, il giorno del mese selezionato e la fascia oraria (che può essere 'mattina' o 'pomeriggio')

Dott	1	2	...
Jones	...	DH	...
...	...	...	...
Freud		UM	...
...	...	...	...

$\Longrightarrow$

Dott	1	2	...
Jones	...	UM	...
...	...	...	...
Freud		DH	...
...	...	...	...

Figura 1: Esempio di una mossa di scambio

Si osservi che data una soluzione, la dimensione dello spazio di ricerca, ovvero il numero di vicini della soluzione corrente è pari a $\frac{m*(m-1)}{2} * n * 2 = O(m^2 * n)$.

È possibile ampliare il concetto di mossa di scambio con il concetto di mossa di scambio *composta*:

"Una mossa di scambio composta è costituita da una o più mosse di scambio"

Le mosse composte risultano molto utili per turni consecutivi che si devono (o si vogliono) spostare assieme.

È importante notare che le mosse di scambio così definite *mantengono l'integrità della soluzione rispetto ai vincoli di copertura*: infatti se la soluzione A soddisfa i vincoli di copertura e ad essa applichiamo una qualsiasi mossa di scambio, la soluzione B risultante soddisferà ancora i vincoli di copertura, in quanto non vengono tolti né aggiunti turni dalla colonna, ma semplicemente scambiati tra due medici. Quindi, iniziando l'esplorazione del vicinato da una soluzione ammissibile (come quella fornita dal modulo FirstSolution) e applicando solo mosse di scambio si può tralasciare di esplicitare i vincoli di copertura, in quanto non verranno mai violati. Facciamo infine notare che l'applicazione di una mossa di scambio può però portare alla violazione di vincoli di altro tipo: per esempio un medico potrebbe vedersi assegnato un turno che non è tra le sue competenze, o risultare occupato in un giorno in cui aveva chiesto ferie. Questo aspetto è preso in considerazione da una funzione di costo che include la funzione obiettivo $FObj$ e aggiunge delle componenti che penalizzano una tale situazione.

Descriviamo ora brevemente gli algoritmi di ricerca locale sviluppati.

Hill Climbing. La strategia Hill Climbing elaborata consiste nel generare una mossa di scambio in modo casuale e applicarla alla soluzione corrente. Se la nuova soluzione migliora o mantiene invariata la funzione di costo senza violare alcun vincolo, allora questa diventa la nuova soluzione corrente; se invece la nuova soluzione peggiora la funzione di costo oppure viola qualche vincolo, si tiene la soluzione di partenza. Questo procedimento viene iterato fino allo scadere del tempo stabilito dall'utente.

L'unico aspetto cruciale che abbiamo dovuto tener presente nell'implementare questo metodo riguarda la generazione casuale delle mosse, che ha il rischio di generare molte mosse di scambio inutili. La generazione random di

una mossa è stata quindi leggermente pilotata imponendo che il primo medico che concorre allo scambio non possa avere un turno di riposo, evitando così la generazione di mosse inutili.

Steepest Descent. La strategia Steepest Descent consiste nell'enumerare tutte le possibili mosse applicabili alla soluzione corrente, calcolare la funzione obiettivo che si otterrebbe applicando ciascuna mossa e poi applicare la mossa che comporta il maggior decremento della funzione obiettivo. Il processo viene iterato fino a quando alla soluzione corrente non si può applicare alcuna mossa che migliori la funzione obiettivo. Rispetto all'Hill Climbing risulta più dispendioso individuare la mossa da applicare, però si individuano generalmente mosse che migliorano notevolmente la funzione obiettivo.

L'aspetto chiave di questo metodo è il procedimento di enumerazione delle mosse: poiché calcolare il miglioramento della funzione obiettivo è un'operazione molto dispendiosa computazionalmente, bisogna evitare di eseguirla alla cieca su tutte le possibili mosse. Molte mosse portano alla violazione di vincoli e quindi non serve calcolare la funzione obiettivo risultante dalla loro applicazione, perché queste mosse non saranno mai selezionate. La strategia di enumerazione implementata contiene quindi una certa conoscenza dei vincoli ed evita di enumerare mosse che li violerebbero, portando così ad un notevole risparmio del tempo di computazione.

Tabu Search. Questo metodo, data una soluzione corrente, esplora una parte delle possibili mosse applicabili, e applica quella che porta al minor valore della funzione di costo, indipendentemente dal fatto che il valore sia minore o maggiore di quello precedente. Questo permette di uscire dai minimi locali, ma porta al rischio di ciclare continuamente in un insieme di soluzioni. Per evitare ciò viene usata la cosiddetta 'tabu list', una lista che contiene le mosse recentemente applicate e proibisce l'applicazione di mosse che risultino inverse di quelle in essa contenute e che quindi riporterebbero la ricerca a visitare uno stato già considerato precedentemente. Abbiamo stabilito la lunghezza della tabu list tramite due valori k_{min} e k_{max} che indicano la lunghezza minima e massima che la lista può assumere. Quando una mossa entra nella lista le viene assegnato un numero casuale compreso tra k_{min} e k_{max}, che indica per quante iterazioni la mossa deve essere tenuta dalla tabu list. Sperimentalmente, si è riscontrato che l'algoritmo dà i migliori risultati con $k_{min} = 5$ e $k_{max} = 10$.

Hybrid Steepest Descent. È stato implementato infine un algoritmo di ricerca locale interamente in SICStus Prolog, che quindi non si appoggia sul framework JEasyLocal ma sulla programmazione con vincoli. L'idea che sta alla base di questo algoritmo è di esplorare porzioni di vicinato di un data soluzione, per ciascuna porzione selezionare un rappresentante (il vicino migliore della porzione) ed infine spostarsi nel migliore dei rappresentanti individuati. Le porzioni di vicinato devono nel loro insieme riuscire a coprire tutto il vicinato della soluzione corrente. Si tratta quindi di un algoritmo di tipo Steepest Descent, in quanto ad ogni mossa, dopo avere esplorato più vicinati, ci si sposta verso un vicino migliore; poiché l'esplorazione di una porzione di vicinato alla ricerca del miglior rappresentate viene effettuata con la programmazione con vincoli, ovvero dal risolutore di SICStus Prolog tramite il predicato `labeling`, abbiamo chiamato questo algoritmo Hybrid Steepest Descent.

Il concetto di porzione di vicinato che abbiamo elaborato è quello di *vicinato rispetto al giorno*: data una soluzione S e un giorno del mese $x \in 1..31$, il vicinato di S rispetto al giorno x consiste in tutte le soluzioni S' che sono identiche ad S per tutti i turni dei giorni g tali che $g \neq x$ e differiscono da S per tutti i turni del giorno x. Quindi ogni soluzione avrà 31 porzioni di vicinato possibili. Esplorare una porzione di vicinato relativa al giorno x significa esplorare tutte le possibili disposizioni di turni di quel giorno: la migliore disposizione sarà il rappresentante di quella porzione di vicinato.

L'algoritmo così ottenuto sfrutta pertanto la programmazione con vincoli come supporto alla ricerca locale (nell'esplorazione del vicinato rispetto ad un giorno).

4 Esperimenti

Abbiamo effettuato degli esperimenti allo scopo di confrontare i quattro diversi metodi di ricerca locale implementati per la risoluzione del problema (Hill Climbing, Steepest Descent, TabuSearch, Hybrid Steepest Descent). Sono stati effettuati due tipi di esperimenti: il primo su 50 istanze con struttura reale di dimensione variabile, generate in modo casuale e raggruppate in 5 serie da 10; il secondo su due istanze di dimensione reale, anch'esse generate in modo casuale. In entrambi gli esperimenti abbiamo considerato un team di 20 medici.

Primo test—metodologia. Il primo test si prefigge lo scopo di analizzare come si comportano i quattro metodi di ricerca locale su più istanze di diversa dimensione simili alle istanze reali. Sono state generate in modo casuale 50 istanze di dimensione variabile, raggruppate in serie da 10 con orizzonte temporale n di 5, 10, 20, 30 e 40 giorni, rispettivamente.

Tutte le istanze sono state risolte utilizzando i quattro diversi algoritmi per la ricerca locale, per un totale di 200 esecuzioni, ciascuna limitata da un tempo proporzionale alla dimensione dell'istanza, precisamente da $80 \cdot n$ secondi.

In ciascuna esecuzione siamo interessati a tenere traccia delle variazioni del valore della funzione obiettivo, che corrispondono a miglioramenti della soluzione corrente. Ogni volta che l'algoritmo trova una soluzione nuova che modifica il valore della funzione obiettivo, viene registrato

l'*istante temporale* in cui è stata trovata la nuova soluzione e il *valore* della funzione obiettivo.

Dopo aver raccolto i dati, questi sono stati opportunamente aggregati per poter analizzare l'andamento medio dei vari metodi sui vari gruppi di istanze. Per ciascuno dei 5 gruppi di 10 istanze è stato effettuato il seguente procedimento: per ciascuno dei 4 metodi l'andamento della funzione obiettivo di ciascuna istanza è stato discretizzato su intervalli di tempo regolari; successivamente sugli stessi intervalli di tempo è stata effettuata la media dei valori delle funzioni obiettivo delle 10 istanze; in questo modo abbiamo ottenuto la media dell'andamento della funzione obiettivo, per ogni metodo e per ogni gruppo di istanze.

Primo test—Risultati. In figura 2 vediamo l'andamento delle funzioni obiettivo relative ai quattro metodi per le istanze da 10, 20, 30 e 40 giorni.

Il risultato più evidente è che il metodo Hill Climbing risulta essere il migliore su tutte le tipologie di istanze: infatti è quello che fa scendere la funzione obiettivo nel modo più veloce. Questo è dovuto al fatto che il tempo per trovare una mossa migliorativa è molto basso, in quanto non si effettua un'esplorazione esaustiva del vicinato, ma ci si accontenta di spostarsi nel primo vicino generato casualmente che migliora la funzione obiettivo o la lascia immutata; inoltre, grazie a questa caratteristica, l'Hill Climbing non si blocca mai in minimi locali, ma continua a perturbare la soluzione corrente con nuove soluzioni dello stesso valore e quindi prima o poi riesce ad individuare un mossa migliorativa.

Il metodo Steepest Descent classico risulta invece il peggiore: è infatti più lento dell'Hill Climbing, in quanto impiega molto tempo nell'esplorazione del vicinato prima di decidere quale mossa effettuare, e il tempo impiegato nell'esplorare il vicinato non è compensato da un sostanziale decremento della funzione obiettivo ad ogni passo. Inoltre presenta il limite intrinseco di bloccarsi nei minimi locali, e questo comporta che dopo un po' di miglioramenti, al primo minimo locale non riesce più a proseguire.

Il Tabu Search è sicuramente migliore dello Steepest Descent: nelle prime fasi delle elaborazioni i metodi si equivalgono (infatti nel grafico si nota che le due linee sono inizialmente sovrapposte), poi entra in gioco il meccanismo di proibizione del Tabu Search che evita di cadere nei minimi locali e consente di continuare a esplorare il vicinato e a migliorare la soluzione. Questo metodo non riesce però a mostrare un andamento vantaggioso quanto quello dell'Hill Climbing, soprattutto sulle istanze medio-grandi (da 20 giorni in poi) in cui si nota che le linee dei due metodi sono ben distanziate.

Il metodo Hybrid Steepest Descent merita un'analisi più approfondita, in quanto il suo comportamento varia in base alla dimensione delle istanze su cui viene applicato. Nelle istanze più piccole (10 giorni) il suo andamento è il peggiore in termini di velocità di decremento della funzione obiettivo: questo è dovuto all'alto overhead computazionale di cui questo metodo soffre per esplorare in modo esaustivo tutte le porzioni di vicinato; sulle istanze piccole questo overhead non viene pagato da un alto decremento della funzione obiettivo per mossa. Nelle istanze medie (20 e 30 giorni) lo sforzo computazionale (in termini di tempo) dell'esplorazione delle porzioni di vicinato viene compensato da decrementi notevoli della funzione obiettivo e questo metodo risulta subito più competitivo del Tabu Search, già dopo le prime decine di secondi. Questo metodo però scala male sulle istanze grandi (40 giorni): pur mantenendo la sua tendenza a rivelarsi più competitivo del Tabu Search, soffre dell'alta quantità di tempo necessaria all'esplorazione delle porzioni di vicinato, e quindi riesce a superare effettivamente l'andamento del Tabu Search (comunque non in modo netto) solo dopo qualche centinaio di secondi.

Secondo test—Metodologia. Questo esperimento prevede di analizzare l'andamento dei quattro algoritmi di ricerca locale su due istanze casuali di dimensione reale, ovvero di 30 giorni, con un timeout di 12 ore. Questo al fine di valutare il valore della funzione obiettivo che questi metodi riescono a raggiungere con molto più tempo a disposizione. Sono state utilizzate due istanze casuali di orizzonte temporale da 30 giorni, sono state risolte con i quattro metodi e a ciascuna delle otto esecuzioni è stato fissato un timeout di 12 ore. I dati registrati sono gli stessi dell'esperimento precedente, ovvero l'istante di tempo in cui la soluzione migliora e il relativo valore della funzione obiettivo. Non si sono dovute effettuare medie di alcun tipo, trattandosi di istanze singole: i dati ottenuti non sono dunque stati elaborati.

Secondo test—Risultati. L'andamento della funzione obiettivo delle due istanze è analogo all'andamento medio delle istanze da 30 giorni riportato nel primo esperimento. Riportiamo i valori della funzione obiettivo raggiunti dai quattro metodi per le due istanze dopo il periodo di elaborazione di 12 ore; questi valori sono riportati nella tabella 2. Nell'ultima riga è riportato l'ottimo raggiunto dalla ricerca (vincolata ma esaustiva) del solo constraint solver in 12 ore. Tali valori, quasi doppi di quelli trovati dai metodi ibridi, costituiscono di fatto la giustificazione a questo lavoro.

Metodo	FObj	
	Istanza 1	Istanza 2
Hill Climbing	3500	3240
Steepest Descent	4120	3760
Tabu Search	3520	3350
Hybrid Steepest Descent	3630	3460
CLP(FD)	7590	5990

Tabella 2: Funzioni obiettivo ottenute in 12 ore di elaborazione

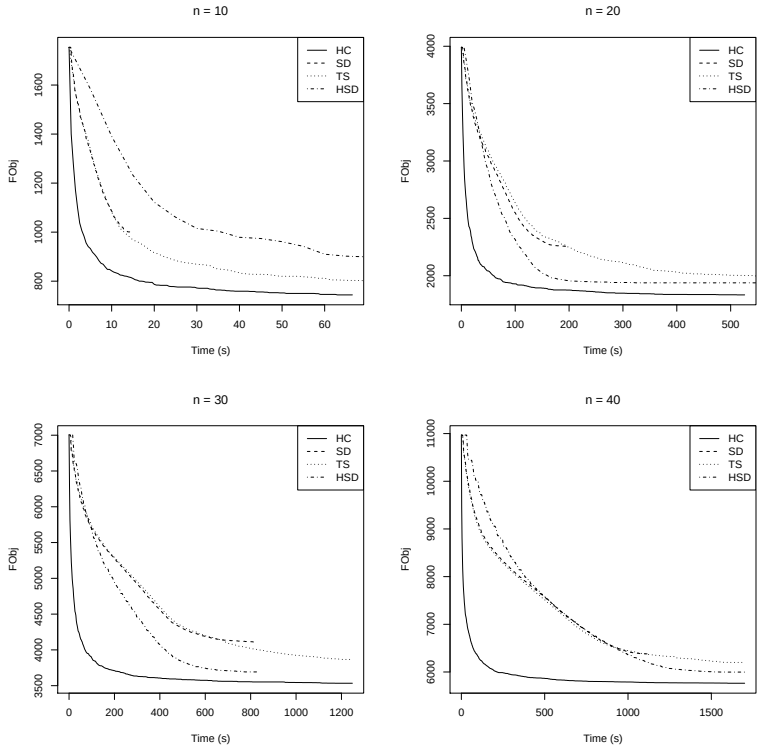

Figura 2: Andamento medio dei risultati ottenuti sulle istanze da 10, 20, 30 e 40 giorni

Ora vale la pena di soffermarsi sul diverso valore della funzione obiettivo raggiunto dai vari metodi, come riportato nella tabella 2: il metodo Hill Climbing è quello che individua la soluzione migliore; a seguire troviamo il metodo Tabu Search, che non si discosta nettamente dall'Hill Climbing; quindi l'Hybrid Steepest Descent, anche se non molto distante dal Tabu Search; infine lo Steepest Descent, fanalino di coda, notevolmente staccato dagli altri tre metodi.

Per finire, nella Tabella 3 è riportato il tempo di raggiungimento del valore del 105%, 102% e 101% rispetto al valore calcolato in 12 ore per le istanze 1 e 2. Come si può osservare la convergenza è molto rapida.

	Istanza 1			Istanza 2		
Metodo	+5%	+2%	+1%	+5%	+2%	+1%
HC	0.1	0.1	0.5	0.3	0.8	2.8
SD	0.3	0.3	0.3	0.3	0.4	0.4
TS	1.0	1.3	1.4	1.1	1.6	1.7
HSD	0.6	0.6	0.7	0.4	0.4	0.4

Tabella 3: Tempo (in secondi) di approssimazione del miglior valore trovato in 12 ore

5 Sviluppi futuri

Gli sviluppi futuri che si delineano per l'attuale lavoro sono principalmente due:

1. implementare nuovi metodi di ricerca locale;
2. migliorare l'attuale modello su cui si basano gli algoritmi di risoluzione del problema.

I risultati emersi dagli esperimenti evidenziano che il metodo migliore risulta l'Hill Climbing, in quanto impiega meno tempo degli altri nell'individuare una mossa migliorativa. Si può quindi pensare di sviluppare algoritmi di ricerca locale che possano essere competitivi con l'Hill Climbing in termini di sforzo computazionale richiesto per ogni mossa; ad esempio potrebbe risultare utile l'impiego dell'algoritmo Tabu Search con strategie di tipo First Improvement o Elite Strategy, che non esplorano esaustivamente l'intero vicinato di una soluzione. Un'ulteriore possibilità è l'implementazione di un algoritmo di tipo Simulated Annealing per il problema che comporterebbe, però, una fase di tuning dei parametri alquanto laboriosa.

Si desidera inoltre realizzare un constraint solver su domini finiti autonomo per rendere la piattaforma indipendente da linguaggi proprietari e per ottimizzare la gestione dell'interazione tra le due tecniche.

Dal punto di vista metodologico si intende, in un lavoro futuro, effettuare un'analisi più precisa del comportamento

degli algoritmi utilizzando strumenti statistici più raffinati di valutazione delle performance assieme ad un tuning più preciso dei parametri degli algoritmi.

Il sistema è attualmente utilizzato nel reparto di neurologia. Le soluzioni fornite in un paio di minuti sono ritenute soddisfacenti anche dal punto di vista pratico dal personale addetto. Prevediamo comuqneu ulteriori aggiustamenti nella funzione obiettivo e confidiamo in un suo utilizzo generalizzato nella struttura ospedaliera. In tal caso dovremo sviluppare tecniche per la specializzazione più o meno automatica del tool.

Ringraziamenti

La ricerca svolta è parzialmente supportata dal progetto MIUR PRIN 2005015491. Ringraziamo il policlinico Universitario dell'Università di Udine ed in particolare il dr. Filì ed il prof. Bergonzi per il loro supporto in questa attività ed il direttore amministrativo prof. Brusaferro che ci ha consentito di condurre questa ricerca.

Riferimenti bibliografici

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

[2] Krzysztof R. Apt. *Principles of Constraint Programming*. Cambridge University Press, Cambridge (UK), 2003.

[3] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004.

[4] Mats Carlsson, Greger Ottosson, and Björn Carlson. An open-ended finite domain constraint solver. In H. Glaser, Hartel P., and Kucken H., editors, *Programming Languages: Implementations, Logics, and Programming*, number 1292 in Lecture Notes in Computer Science, pages 191–206. Springer-Verlag, Berlin-Heidelberg (Germany), 1997.

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

[6] Filippo Focacci, François Laburthe, and Andrea Lodi. Local search and constraint programming. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, chapter Local Search and Constraint Programming, pages 369–403. Kluwer Academic Publishers, 2003.

[7] Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristic. *Artificial Intelligence*, 139(1):21–45, 2002.

[8] Eric Monfroy, Frédéric Saubion, and Tony Lambert. On hybridization of local search and constraint propagation. In Bart Demoen and Vladimir Lifschitz, editors, *Proceedings of the 20th International Conference on Logic Programming (ICLP 2004)*, number 3132 in Lecture Notes in Computer Science, pages 299–313. Springer-Verlag, 2004.

6 Contacts

Raffaele Cipriano
Dipartimento di Matematica e Informatica - Università degli Studi di Udine
via delle Scienze 208
I-33100 Udine, Italy
e-mail: raffaele.cipriano@gmail.com

Luca Di Gaspero
Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica - Università degli Studi di Udine
via delle Scienze 206
I-33100 Udine, Italy
phone: +39-0432-558242
e-mail: l.digaspero@uniud.it

Agostino Dovier (Corresponding Author)
Dipartimento di Matematica e Informatica - Università degli Studi di Udine
via delle Scienze 208
I-33100 Udine, Italy
phone: +39-0432-558494
e-mail: dovier@dimi.uniud.it