UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI SCIENZE MM.FF.NN.

Corso di Laurea in Matematica

Dipartimento di Matematica Pura ed Applicata

TESI DI LAUREA

Soluzione e apprendimento automatico di vincoli temporali con preferenze

Relatori: Prof.ssa FRANCESCA ROSSI
Prof. ALESSANDRO SPERDUTI

Controrelatore: Prof. GILBERTO FILÈ

Laureanda: KRISTEN B. VENABLE

ANNO ACCADEMICO 2000-2001

Ringraziamenti

- Ringrazio la Prof.ssa Francesca Rossi per avermi seguito durante tutta le tesi e per avermi aiutato a capire ed apprezzare cosa significhi fare ricerca.
- Ringrazio il Prof. Alessandro Sperduti per aver curato la parte di questa tesi sull'apprendimento automatico e per il corso tenuto che mi ha appassionato a questo argomento.
- Ringrazio Lina Khatib, Robert Morris e Paul Morris per aver reso speciale dal punto di vista intellettuale e umano il periodo trascorso al centro di ricerche NASA Ames.

Indice

Capitolo 1

Introduzione

Il problema affrontato in questa tesi è la soluzione e l'apprendimento automatico di vincoli temporali con preferenze.

1.1 Motivazioni

I problemi di vincoli temporali con preferenze sono un formalismo che è stato introdotto recentemente [17] e che permette di modellare problemi che riguardano decisioni temporali. In questa classe di problemi ricadono molti esempi di applicazioni reali. Pensiamo ad esempio una società che offra come servizio il trasporto di pazienti, ad esempio anziani, alle visite mediche specialistiche o ai laboratori di analisi. Il problema del trasporto di un paziente ad un appuntamento può essere schematizzato nel seguente modo: il paziente deve arrivare in tempo per l'appuntamento, tuttavia per sfruttare il più possibile i mezzi e gli autisti non converrà portarlo troppo presto. Supponiamo che l'appuntamento con lo specialista sia alle ore 11:00. È ragionevole pensare che l'arrivo del paziente allo studio debba avvenire tra le 10:00 e le 11:05. Tuttavia è preferibile che il paziente arrivi tra le 10:50 e le 11:00. Questa circostanza è rappresentabile da un vincolo temporale con preferenze. Esso infatti può essere modellato da un intervallo, i.e. una porzione di tempo, e una funzione di preferenza che mappa ogni istante dell'intervallo in un valore che indica quanto è auspicabile che l'evento avvenga in quell'istante. Il vincolo che rappresenta l'evento: "arrivo del paziente", dunque avrà come intervallo [10:00,11:05] e come funzione di preferenza una funzione che mappa gli istanti compresi tra le 10:50 e le 11:00 in valori di preferenza più alti rispetto a quelli associati agli altri elementi dell'intervallo.

Facciamo un altro esempio. Consideriamo il robot che NASA invierà su Marte in futuro: Mars Rover. Esso dovrà compiere diverse azioni tra le quali riprendere delle immagini di Marte e inviarle a terra, estrarre e analizzare dei

campioni di suolo marziano. Già questo piccolo esempio può essere scomposto in molte attività:

- cercare e scegliere un oggetto da fotografare;
- metterlo a fuoco;
- fare la fotografia;
- inviare l'immagine a terra;
- scegliere un oggetto da campionare;
- estrarre un reperto;
- eseguire l'analisi (chimica o fisica);
- inviare i dati a terra.

Tutti questi eventi hanno finestre di tempo diverse entro cui devono avvenire. Ad esempio certi oggetti possono essere fotografati solo con la luce solare, altri solo al buio. Per altri oggetti è preferibile avere un'immagine diurna ma anche una notturna avrebbe comunque qualche valore scientifico. Alcune attività come l'estrazione del reperto consumano molta energia e potranno essere effettuate solo in momenti in cui la batteria è carica e non in concomitanza con altre attività dispendiose. Le attività di comunicazione con la terra potranno avvenire in tempi legati alle posizioni dei satelliti orbitanti, etc.

1.2 Obiettivi e risultati

Un primo problema che ci siamo proposti di risolvere è il seguente: dato un numero di attività e le relazioni di tipo temporale con preferenze tra loro, trovare una schedulazione delle attività che globalmente porti al massimo grado di soddisfazione dei vincoli. Gli esempi menzionati sopra e molti altri possono essere inquadrati in questo schema. Questo problema è stato affrontato cercando prima una base teorica su cui fondare un processo di soluzione e poi implementando tale processo. Per affrontare la parte teorica del problema sono stati considerati gli aspetti teorici della soluzione di problemi di vincoli temporali classici, (TCSPs) [8] e di problemi di vincoli con preferenze [4]. I risultati teorici ottenuti sono stati un'estensione delle proprietà che si ottengono applicando algoritmi di consistenza locale a problemi di vincoli temporali con preferenze e la dimostrazione di come questi algoritmi permettano di trovare una soluzione ottima senza backtracking per alcune classi di problemi.

Sulla base dei risultati teorici è stato implementato un risolutore che trova la schedulazione ottima, cioè quella globalmente più preferibile.

Questa tesi affronta anche un secondo tipo di problemi. Nei problemi reali spesso è difficile avere informazioni sufficienti a definire in dettaglio tutti i vincoli del problema. Infatti definire tutti i vincoli del problema significa conoscere esattamente quale sia l'intervallo temporale che deve intercorrere tra un evento e un altro, quali siano le preferenze da associare a ciascun punto dell'intervallo, quale sia la finestra temporale in cui un evento può cominciare, quale sia quella in cui può finire, etc.

Facciamo un esempio. Supponiamo di dover schedulare 5 attività. A ciascuna di esse facciamo corrispondere due variabili, una che ne rappresenta l'inizio e una la fine. Per modellare questo problema in un problema di vincoli temporali con preferenze dobbiamo dare tutte le informazioni citate sopra per 20 vincoli. Si capisce come il problema diventi difficile da gestire già con poche variabili. Riconsideriamo l'esempio di Mars Rover: la relazione tra la messa a fuoco e la fotografia è ovvia ma la relazione e le preferenze tra la messa a fuoco e la trasmissione dei dati sull'analisi del reperto non sono altrettanto facile da definire con esattezza.

D'altra parte, può essere più facile dare una preferenza globale ad una soluzione. Ad esempio, una o più soluzioni in cui gli eventi avvengono tutti in contemporanea può non andare bene se si hanno poche macchine a disposizione per svolgere le attività. Tornando all'esempio di Mars Rover, una schedulazione che prevede di completare le attività tutte nelle prime ore del mattino è peggiore dal punto di vista dell'energia rispetto a una che concentra le attività all'inizio del pomeriggio e così via. L'importanza di risolvere questo problema è principalmente dovuta alla difficoltà di esplicitare in modo preciso e completo preferenze locali. In secondo luogo vi sono problemi con decine di migliaia di soluzioni e sarebbe troppo dispendioso associare ad ogni soluzione una preferenza e poi confrontarle tutte tenendo le migliori. Infine, spesso, non esiste una formula utilizzabile per associare la preferenza globale ad una soluzione del problema ma tale soluzione può essere valutata solo attraverso giudizi che provengono dall'esperienza e non sono esprimibili matematicamente. Questo è il caso, ad esempio delle schedulazioni per i telescopi dove solo astronomi esperti riescono a dire quali successioni di puntamenti e scansioni di immagini daranno risultati migliori.

Ci siamo dunque proposti di studiare l'applicabilità delle tecniche di apprendimento automatico a questo tipo di problemi. L'utilità di tali tecniche sarebbe quella di indurre da delle informazioni globali (cioè le preferenze delle soluzioni) informazioni locali (cioè le funzioni di preferenza sui vincoli). Per fare questo abbiamo scelto il metodo di discesa di gradiente la cui applicabilità appariva più immediata. In seguito si è cercato di identificare i

parametri rispetto ai quali formulare il problema. A tal fine è stata isolata una particolare classe di funzioni, parabole convesse, la cui parametrizzazione ha permesso di impostare l'apprendimento.

Dopo questo studio preliminare, è stato possibile implementare un modulo di apprendimento automatico che da un insieme di esempi, cioè soluzioni e preferenze associate, e da un problema di vincoli classico, ricostruisce un problema di vincoli temporali con preferenze che simuli globalmente, cioè a livello delle soluzioni, il comportamento desiderato.

Riassumendo, quindi, i principali risultati di questa tesi sono:

- alcuni risultati teorici sulla trattabilità di problemi di vincoli temporali con preferenze;
- l'implementazione di un risolutore per vincoli temporali semplici con preferenze;
- l'implementazione di un generatore random di tali problemi;
- la sperimentazione del risolutore su problemi generati random e su problemi reali;
- l'implementazione di un modulo di apprendimento automatico;
- la sperimentazione di tale modulo su problemi generati random e su problemi reali.

Questa tesi è stata realizzata in parte all'Università di Padova e in parte al laboratorio informatico di ricerca NASA Ames di Moffet Field, California. La NASA infatti ha dimostrato molto interesse per le possibili applicazioni spaziali di questa ricerca.

Una parte dei risultati ottenuti in questa tesi sono pubblicati in [32].

1.3 Struttura della tesi

La tesi è così strutturata. Il **Capitolo 2** descrive tutti i formalismi che sono serviti sia per i risultati teorici che per quelli implementativi. In particolare la sezione 2.1 descrive i problemi temporali classici nella formulazione proposta da Rina Dechter e Judea Pearl in [8]. I questa sezione vengono descritte le reti di vincoli temporali, gli operatori su vincoli e due algoritmi principali per la consistenza locale. Viene inoltre introdotta la nozione di rete minima e di decomponibilità che saranno molto utili. Questa sezione è molto importante perchè contiene risultati teorici per vincoli temporali senza preferenze che verranno estesi a vincoli temporali con preferenze in questa tesi. Inoltre Pearl

e Dechter hanno messo in evidenza una sottoclasse di problemi temporali la cui risolvibilità ha una complessità polinomiale.

Nella sezione 2.2 è riportata una breve descrizione della teoria dei vincoli con preferenze basati su semianelli di [4]. Questa teoria è alla base anche dei vincoli temporali con preferenze. Infatti in questa sezione si descrive come [4] hanno rappresentato l'aggiunta di preferenze ad un problema di vincoli con la struttura algebrica di semianello. In questa sezione vengono inoltre presentati dei risultati che legano le proprietà degli operatori del semianello all'applicabilità di algoritmi di consistenza locale. In particolare l'identificazione dell'importanza dell'idempotenza dell'operatore moltiplicativo ha giocato un ruolo fondamentale in molti risultati ottenuti in questa tesi.

Nella sezione 2.3 è descritto come in [17] abbiano fuso i due formalismi citati sopra in quello dei vincoli temporali con preferenze. I loro risultati sono direttamente alla base di questa tesi. Infatti in questa sezione vengono presentati dei risultati sulla complessità di questi problemi che in questa tesi vengono ripresi e dimostrati in modo alternativo. In [17] hanno anche identificato una classe di funzioni che utilizzate come funzioni di preferenze sui vincoli garantiscono la risolvibilità in tempo polinomiale.

Nella sezione 2.4 viene descritto brevemente il concetto di apprendimento automatico, ponendo particolare enfasi sull'apprendimento induttivo. Viene inoltre descritta la tecnica di discesa di gradiente e viene data l'approssimazione continua e derivabile della funzione *min* che verrà utilizzata nel corso della tesi.

Il **Capitolo 3** è dedicato ai risultati teorici che sono stati ottenuti. Vengono riprese le definizioni date in [17] di problema di vicoli temporali con preferenze semiconvesse e problema di vincoli temporali semplici con preferenze semiconvesse. Nella sezione 3.1 viene descritto l'algoritmo di consistenza locale, che utilizza la consistenza sui cammini, nella versione adattata ai problemi semplici con preferenze.

Nella sezione 3.2 viene concentrata l'attenzione su una sottoclasse trattabile di problemi, quella appunto dei problemi di vincoli temporali semplici con preferenze semiconvesse. Per questa classe vengono enunciati e dimostrati dei risultati teorici che descrivono gli effetti dell'applicazione dell'algoritmo di consistenza locale e un risultato fondamentale sulla risolvibilità di tali problemi in tempo polinomiale e senza bisogno di backtracking. Questo risultato è importante perché la dimostrazione è costruttiva nel senso che fornisce lo schema per l'algoritmo risolutore.

Nelle sezioni 3.3 e 3.4 vengono considerati due casi che si ottengono rilassando due ipotesi della sottoclasse trattabile. Nella 3.3 viene indicato come sia possibile decomporre un problema in cui le funzioni non appartengano alla classe identificata in [17] in un insieme di problemi con funzioni di tale classe.

Sulla base di tale fatto viene proposto un metodo per risolvere tali problemi. Nella sezione seguente viene invece affrontato il caso in cui i problemi non siano semplici, cioè ogni vincolo abbia più di un intervallo. Anche per questi problemi viene suggerita una tecnica di risoluzione basata sulla scomposizione in problemi trattabili.

La sezione 3.5 è dedicata ai risultati sulla complessità riguardante i problemi trattati nelle tre sezioni precedenti.

Il **Capitolo 4** è dedicato all'implementazione del risolutore.

La sezione 4.1 descrive l'algoritmo principale cioè quello che, dato un problema di vincoli temporali semplici con preferenze, trova una soluzione ottima in tempo polinomiale. In questa sezione vengono anche descritte due varianti che sono ottenute dallo schema principale, una rivolta a trovare tutte le soluzioni ottime, l'altra implementata per generare gli insiemi di esempi su cui allenare e testare il modulo di apprendimento.

La sezione 4.2 descrive le scelte implementative che sono state fatte durante la realizzazione del risolutore, cercando di giustificarne le motivazioni.

Il **Capitolo 5** riporta i risultati sperimentali del risolutore. Nella sezione 5.1 viene dato un esempio di come operi il risolutore. L'esempio è rappresentato graficamente e vengono mostrate tutte le varie fasi dell'elaborazione della rete di vincoli da parte dell'algoritmo.

La sezione 5.2 è dedicata all'implementazione del generatore random di problemi semplici con preferenze. Infatti tale implementazione si è resa necessaria sia per testare la performance del risolutore, che per generare dei problemi su cui simulare un processo di apprendimento. Viene descritto in dettaglio come avviene il processo di generazione di questi problemi e quali sono i parametri secondo i quali il generatore agisce.

Nella sezione 5.3 vengono proposti, descritti e valutati gli esperimenti condotti sul risolutore.

Il **Capitolo 6** riguarda l'apprendimento automatico di problemi temporali con preferenze.

Nella prima sezione vengono descritte in dettaglio l'insieme di formule che serviranno nella discesa di gradiente.

La sezione 6.2 descrive l'algoritmo secondo il quale procede il modulo di apprendimento automatico. Vengono fatte inoltre alcune considerazioni sull'utilizzo delle funzioni di preferenza da parte del modulo durante gli aggiornamenti, sui criteri di valutazione dei risultati, e sui criteri rispetto a cui ha senso far terminare l'apprendimento.

Nella sezione 6.3, in analogia con quanto fatto per il risolutore, vengono descritte le scelte implementative.

Il **Capitolo 7** è dedicato ai risultati sperimentali del modulo di apprendimento.

Nella prima sezione vengono descritte le varie tipologie di problemi su cui è stato testato il modulo. La sezione 7.2 descrive i risultati ottenuti su problemi generati dal generatore random. Vengono indicati i parametri che sono stati fatti variare e quelli che sono stati mantenuti fissi. Vengono, inoltre, fatte alcune considerazioni sui risultati ottenuti.

Le sezioni 7.3 e 7.4 invece contengono in dettaglio la descrizione di due esempi di problemi di interesse per applicazioni NASA. Tali esempi mettono in luce come sia possibile utilizzare il modulo e il risolutore in modo combinato per ottenere soluzioni sempre migliori di problemi molto grandi. Nell'ultimo capitolo vengono tratte le conclusioni e proposte alcune direzioni future di ricerca che appaiono particolarmente promettenti.

L'**Appendice** contiene i codici relativi a tutte le implementazioni degli algoritmi descritti in questa tesi.

Capitolo 2

Background

2.1 Vincoli Temporali

2.1.1 I modelli e le proprietà comuni

Problemi che coinvolgono i vincoli temporali si trovano in diverse aree dell'informatica, come lo scheduling, la verifica dei programmi e il calcolo parallelo. Ricerche nel campo del linguaggio naturale [1] [16] e del planning [26] hanno individuato nuovi problemi di tipo temporale, mirati ad applicazioni di Intelligenza Artificiale. Molti formalismi per codificare ed espletare ragionamenti di tipo temporale sono stati proposti. Tra di essi spiccano l'algebra degli intervalli di Allen [2], l' algebra dei punti di Vilain e Kautz [39], disequazioni lineari (Malik and Binford [24] , Valdès-Pérez [38]), e la mappa temporale di Dean e McDermott [7].

Le quattro parti fondamentali di un sistema che permetta di ragionare temporalmente sono: una base di nozioni temporali, una procedura che ne controlli la consistenza, un meccanismo che dia risposte a domande di tipo temporale e un meccanismo induttivo che possa scoprire informazioni nuove [36]. Le entità primitive della base di nozioni temporali sono *proposizioni*, e.g., "stavo guidando l'auto" oppure "il libro era sulla tavola", alle quali vengono usualmente associati degli intervalli temporali. Ogni intervallo rappresenta il periodo durante il quale la proposizione è vera. L'informazione temporale può essere relativa (e.g., "A è avvenuto prima di B"), oppure metrica (e.g., "A è cominciata almeno tre ore prima che B terminasse". Al fine di esprimere informazioni meno specifiche possono essere utilizzate frasi contenenti disgiunzioni come "puoi venire a trovarmi o prima o dopo pranzo". Inoltre è utile avere la possibilità di far riferimento a valori assoluti come le 4:00 del mattino oppure a durate come in " la conferenza è durata tre ore". Date informazioni di questo tipo, si vuole rispondere a domande come: è possibile che la proposizione P sia vera al tempo t? quali sono gli istanti in cui la

proposizione P è verificata ? quali sono le possibili relazioni temporali tra la proposizione P e la proposizione Q?
In generale un *Problema di Soddisfazione di Vincoli* (**CSP** Constraint Satisfaction Problem) è un insieme di *vincoli* su un insieme di *variabili*, dove ciascuna variabile ha un insieme di valori che può assumere detto *dominio*. Ogni vincolo specifica gli assegnamenti validi per un sottoinsieme di variabili. Un *Problema di Soddisfazione di Vincoli Temporali* (**TCSP** Temporal Constraint Satisfaction Problem) è un particolare CSP dove le variabili rappresentano istanti e i vincoli rappresentano insiemi di relazioni temporali permesse tra di essi. Varie classi di TCSP sono stati definiti. Essi si differenziano per il tipo di entità temporali che le variabili possono rappresentare, istanti assoluti, intervalli, durate e per la classe di vincoli utilizzata, vincoli di tipo qualitativo, metrico o entrambe.
Ci sono tuttavia delle caratteristiche comuni ai vari formalismi per i TCSP che ora discuteremo. Le **variabili** rappresentano istanti, intervalli o durate temporali. I **domini** sono definiti su un singolo insieme la cui struttura è isomorfa o all'insieme degli Interi o a quello dei Razionali. Questo insieme è detto *struttura temporale*. Il dominio degli istanti e delle durate è la struttura stessa, mentre il dominio delle variabili che rappresentano gli intervalli sarà l'insieme prodotto cartesiano della struttura per sè stessa. Le varie classi di vincoli, qualitativi, metrici,..., sono supportati da un insieme di relazioni temporali di base (**BTR** Basic Temporal Relations). Tutti i BTR soddisfano le seguenti due condizioni: (i) i loro elementi si escludono mutuamente, e (ii) l'unione degli elementi è il vincolo universale. Tutti i vincoli di un TCSP sono **binari** e della forma $C_{ij} = \{r_1, ..., r_k\}$ dove X_i, X_j sono variabili, $k > 0$, e $r_1, ..., r_k \in BTR$. La loro interpretazione è:

$$(X_i r_1 X_j) \vee \cdots \vee (X_i r_k X_j)$$

È possibile avere vincoli metrici sui punti **unari** ma essi possono essere ridotti a vincoli binari. A tal fine viene introdotta una variabile speciale X_0 il cui dominio contiene un solo elemento:$D_0 = \{0\}$. Questo permette di trasformare il vincolo unario $C_i = \{r_1, ..., r_k\}$ nel vincolo binario $C_{0i} = \{r_1, ..., r_k\}$. La forma dei vincoli temporali è una caratteristica fondamentale dei TCSP. Infatti mentre i vincoli nei CSP standard sono descritti mediante le loro estensioni, i vincoli temporali sono descritti in termini di elementi di BTR. Questo significa che si possono trovare soluzioni manipolando sottoinsiemi di BTR e non specifiche estensioni. Un vincolo esprimibile da un'unica disgiunzione viene chiamato **singoletto**. Una *etichettatura dei singoletti* di un TCSP assegna ad ogni paio di variabili X_i, X_J una relazione temporale di base r tale che $r \subseteq C_{ij}$. Le **soluzioni** di un TCSP sono le sue etichettature a singoletti *consistenti*. La consistenza di una etichettatura a singoletti è definita secondo

la semantica di ciascuna classe di TCSP. Poiché le soluzioni di un TCSP sono etichettature di singoletti, e non assegnamenti alle variabili, la nozione di valore realizzabile viene sostituita da quella di **relazione realizzabile**. Una relazione $r \in BTR$ è realizzabile per la coppia (X_i, X_j) se e solo se esiste una soluzione nella quale r è assegnata alla coppia. È importante notare che nei TCSP qualitativi, r realizzabile per (X_i, X_j) implica che $r \in C_{ij}$ mentre nel caso metrico diventa $r \subseteq C_{ij}$. Il *vincolo minimo* C_{ij}^{min} è l'insieme delle relazioni realizzabili tra X_i e X_j. Come per i CSP, un TCSP i cui vincoli sono tutti minimi è detto **minimo** e, dato un TCSP, è sempre possibile ridurlo ad uno minimo *equivalente*.

I due problemi principali che ci si pone quando si trattano TCSP sono : (i) deciderne la consistenza, cosa che è strettamente collegata con il compito di trovare una soluzione, e (ii) trovare la rappresentazione minima, la quale, in un certo senso, rappresenta tutte le soluzioni. Trovare la rappresentazione minima è in generale più difficile che provare la consistenza ma permette di rispondere a molte domande in generale con un costo minimo. La ricerca è progredita seguendo le seguenti linee:

• *Identificazione di sottoclassi trattabili e lo sviluppo di algoritmi specializzati per esse.* Queste classi sono definite da due tipi di parametri : (i) proprietà sulla struttura del grafo dei vincoli (grado, larghezza,...), (ii) la classe dei vincoli.

• *Ottimizzazione di tali algoritmi.* A questo riguardo ci sono due metodi conosciuti: (i) ricerca con backtracking, e (ii) e il raffinamento iterativo. Il primo metodo termina sicuramente con la risposta corretta ma ha una complessità esponenziale. Il secondo ha una complessità più bassa ma non è detto che termini nè che dia una soluzione anche se ce ne sono. • *Sviluppo di algoritmi che ottengano una approssimazione polinomiale, che siamo affidabili anche se non completi.*

Tutte la tecniche che elaborano vincoli temporali utilizzano gli **operatori** di intersezione e composizione di vincoli. Tra queste varie tecniche:

Imporre la consistenza locale. L'idea è di imporre un certo grado di consistenza localmente i.e. *k-consistenza* per qualche $k \leq n$, per eliminare le etichettature non realizzabili dai vincoli del problema. In molti casi questo può essere fatto in tempo polinomiale. Le nozioni più semplici e note di consistenza locale sono la consistenza per archi *(arc-consistency)* e la consistenza sui cammini *(path-consistency)*. **Path-consistency** ha un carattere meno locale di **Arc-consistency** in quanto coinvolge tutti i cammini tra due variabili. Tuttavia è noto come sia sufficiente la 3-consistency per garantire la consistenza sui cammini [9]. Path-consistency è molto efficace e talvolta è sufficiente per decidere la consistenza, e.g. il caso dell'etichettattura a singoletti. La complessità dell'algoritmo classico è $O(v^3)$ dove v è il numero delle

variabili.

Metodi con ricerca. Poiché in generale i TCSP non sono trattabili, algoritmi completi devono espletare un qualche tipo di ricerca. La nozione di *etichettatura parziale a singoletti* è definita come un assegnamento nel quale certi vincoli vengono etichettati con un singoletto e altri no. Lo spazio in cui avviene la ricerca è definito su tutte le possibile etichettature parziali. Un controllo ulteriore è ottenuto imponendo la consistenza locale. Un assegnamento alle variabili cosiste nel selezionare un particolare BTR per il vincolo che si sta considerando. Intuitivamente, un algoritmo di ricerca con backtracking etichetta ciascun vincolo con una delle sue BTR fino a quando l'etichettatura parziale è consistente. Ogni volta che si trova una inconsistenza l'algoritmo fa backtracking. Il numero di volte in cui è costretto a ritornare sulle proprie decisioni dipende fortemente dalla strategia con la quale si decide l'ordinamento. Tuttavia per la maggior parte delle classi trattabili path- consistency è sufficiente per trovare una soluzione senza fare backtracking [36].

Ci sono tre classi principali di TCSP. Parleremo brevemente delle prime due e approfonditamente della terza, essendo essa fondamentale per la nostra trattazione.

Vincoli Qualitativi sui Punti
Le variabili rappresentano istanti e la BTR= $\{<, =, >\}$. Sono state studiate tre algebre per questa classe:
- *Algebra Puntuale di Base* in cui le relazioni sono $<, =, >, ?$
- *Algebra Puntuale Convessa* in cui le relazioni sono $\oslash, <, =, >, \leq, \geq, ?$
- *Algebra di Base* in cui le relazioni sono $\oslash, <, =, >, \leq, \geq, ?, \neq$

Vincoli Qualitativi sugli Intervalli
Le variabili rappresentano intervalli temporali e l'insieme delle relazioni temporali di base è così definito:
BTR={ prima, dopo, incontra, incontra entro, sovrappone, sovrappone entro, durante, contiene, eguaglia, comincia, comincia entro, finisce, finisce entro}

2.1.2 Vincoli puntuali metrici

I TCSP di questa classe sono costituiti da:

- un insieme di **variabili** $X_1, ..., X_n$, a dominio continuo (non discretizzato), ciascuna delle quali rappresenta un istante, cioè un punto sull'asse temporale;

- un insieme di **vincoli** sulle variabili.

Ogni vincolo è rappresentato da un insieme di intervalli[1]

$$\{I_1, ..., I_n\} = \{[a_1, b_1], ..., [a_n, b_n]\}.$$

Un **vincolo unario**, T_i, restringe il dominio della variabile $X - i$ all'insieme di intervalli che lo compongono. Formalmente rappresenta la disgiunzione

$$(a_1 \leq X_i \leq b_1) \vee ... \vee (a_n \leq X_i \leq b_n).$$

Un **vincolo binario**, T_{ij}, indica i valori ammissibili per la distanza $X_j - X_i$ e quindi rappresenta la disgiunzione

$$(a_1 \leq X_j - X_i \leq b_1) \vee ... \vee (a_n \leq X_J - X_i \leq b_n).$$

Si assume inoltre che i vincoli vengano sempre dati in *forma canonica* cioè che tutti gli intervalli siano a due a due disgiunti.

Una **rete di vincoli binari** (un *TCSP binario*) consiste in un insieme di variabili $X_1, ..., X_n$, e in un insieme di vincoli unari e binari. Tale rete può essere rappresentata da un *grafo orientato di vincoli*, nel quale i nodi rappresentano la variabili e un arco $i \longrightarrow j$ indica la specifica del vincolo T_{ij}: l'arco viene etichettato con l'insieme degli intervalli. Per ogni vincolo T_{ij} che viene specificato si ha il vincolo equivalente T_{ji}: tuttavia, per semplicità uno solo di essi viene indicato nel grafo. Una variabile speciale X_0 viene introdotta per rappresentare l'istante in cui si fa partire il tempo al quale, da ora in poi, faremo riferimento come "inizio del mondo". Tutti gli istanti diventano così relativi a X_0, e quindi ogni vincolo unario T_i può essere trasformato nel vincolo binario T_{0j}, avente lo stesso insieme di intervalli. Per semplicità si pone $X_0 = 0$.

Una n-upla $X = (x_1, ..., x_n)$ è una *soluzione* se l'assegnamento $\{X_1 = x_1, ..., X_n = x_n\}$ soddisfa tutti i vincoli. Un valore v è *ammissibile* per la variabile X_i se esiste una soluzione in cui $X_i = v$. L'insieme di tutti i valori ammissibili per una variabile è detto il *dominio minimo*. La rete è consistente se esiste almeno una soluzione.
Definiamo ora tre operatori binari su vincoli: unione, intersezione e composizione . Queste definizioni rispettano le rispettive nozioni nella teoria classica degli insiemi.

Definizione 1 *Siano $T = \{I_1, ..., I_l\}$ e $S = \{J_1, ..., J_m\}$ due vincoli cioè insiemi di intervalli di una variabile reale t (t corrisponde a $X_j - X_i$ nel caso di vincoli binari).*

[1]Per semplicità consideriamo solo intervalli chiusi ma nulla vieta di utilizzare intervalli aperti o semi-aperti.

1. *L'**unione** di T e S, $T \cup S$, ammette solo valori che sono ammessi da almeno uno dei due vincoli. Formalmente,*

$$T \cup S = \{I_1, ..., I_l, J_1, ..., J_m\}$$

2. *L'**intersezione** di T e S, indicata con $T \oplus S$, ammette solo i valori che sono ammessi da entrambi. Formalmente,*

$$T \oplus S = \{K_1, ..., K_n\}$$

dove $K_k = I_i \cap J_j$ per qualche i e j e $n \leq l + m$.

3. *La **composizione** di T e S, indicata con $T \otimes S$, ammette solo i valori r per cui esiste $t \in T$ e $s \in S$ tali che $t = s = r$. Formalmente:*

$$T \otimes S = \{K_1, ..., K_n\}$$

dove $K_k = [a = c, b + d]$ per qualche $I_i = [a, b]$ e $J_j = [c, d]$ e $n \leq l \times m$.

Si osservi che per alcune di queste operazioni gli intervalli risultanti non sono in forma canonica, e dunque sarà necessaria una ulteriore riduzione. Le tre operazioni definite seguono le rispettive sui vincoli classici. In particolare, se T e S sono vincoli binari rispettivamente sulle distanze $X_j - X_i$ e $X_k - X_j$, allora $T \otimes S$ ammette solo coppie di valori (x_i, x_j) per le quali esiste un valore x_j tale che (x_i, x_j) è ammessa da T e (x_j, x_k) è ammessa da S.
Queste operazioni vengono estese a operazioni su reti di vincoli nel modo usuale. Date le reti T e S, definite sullo stesso insieme di variabili, vengono definite:

$$(T \cup S)_{ij} = T_{ij} \cup S_{ij}$$

$$(T \oplus S)_{ij} = T_{ij} \oplus S_{ij}$$

dove i e j variano nell'insieme delle coppie di variabili.
È possibile definire un ordine parziale sui vincoli nel seguente modo. Un vincolo binario T è più stretto di S, $T \subseteq S$, se ogni coppia di valori ammissibile per T lo è anche per S, cioè per ogni intervallo $I \in T$ esiste un intervallo $J \in S$ tale che $I \subseteq J$. Il vincolo più stretto è il *vincolo vuoto* $\emptyset$ e se esso è contenuto in una rete allora essa è banalmente inconsistente. Il vincolo più largo è il *vincolo universale* $(-\infty, +\infty)$. Gli archi corrispondenti a vincoli universali sono omessi nella rappresentazione del grafo.

È possibile definire un ordine parziale su reti di vincoli binari aventi lo stesso insieme di variabili nel seguente modo. Una rete T è più stretta di una rete S,$T \subseteq S$, se l'ordinamento parziale $\subseteq$ è soddisfatto da tutti i vincoli che si corrispondono, quindi per tutte le coppie i, j $T_{ij} \subseteq S_{ij}$. Due reti sono *equivalenti* se rappresentano lo stesso insieme di soluzioni. Ciascuna

rete può avere molte rappresentazioni equivalenti; in particolare ce n'è una che è minima rispetto a $\subseteq$, detta la **rete minima**. Quest'ultima è minima perché l'insieme delle reti equivalenti è chiuso rispetto l'intersezione. I vincoli corrispondenti agli archi della rete minima sono detti *vincoli minimi*.

Una rete è **decomponibile**, se ogni assegnamento parziale a un qualsiasi insieme di variabili S che è localmente consistente [2] può essere esteso ad una soluzione. Questo è un concetto fondamentale e la sua importanza sta nel fatto di essere alla base di procedure che trovano una soluzione senza backtracking.
Data una rete di vincoli il primo problema che ci si può porre è quello di determinarne la consistenza. Se poi la rete è consistente si possono cercare soluzioni specifiche, ciascuna delle quali può rappresentare uno scenario di particolare interesse o una risposta ad una specifica domanda. Tra le molte domande sono rappresentative:

- *Quali sono gli istanti in cui X_i può avvenire?*. In questa domanda si vuole apprendere in dominio minimo di X_i.

- *Quali sono tutte le possibili relazioni temporali tra X_i e X_j?*. In questa domanda si vuole apprendere il vincolo minimo tra le due variabili.
  Ad entrambe queste domande è possibile dare una risposta una volta che sia stata calcolata la rete minima.

Davis ha dimostrato che determinare la consistenza di un TCSP è in generale un problema NP-hard.

Teorema 1 (Davis) *(i) Decidere la consistenza di un TCSP è NP-hard; (ii) Decidere la consistenza di un TCSP con al più due intervalli per vincolo è NP-hard.*

Una sottoclasse trattabile: Problemi con Vincoli Temporali Semplici (STP)

Un TCSP in cui tutti i vincoli cono costituiti da un unico intervallo è detto Problema a Vincoli temporali Semplici, o più brevemente STP (dall'inglese Simple Temporal Problems). Nelle reti corrispondenti a tali problemi ogni arco è etichettato con un singolo intervallo $[a_{ij}, b_{ij}]$ il quale rappresenta il vincolo

$$a_{ij} \leq X_J - X_I \leq b_{ij}.$$

[2] Un assegnamento di valori a un insieme di variabili S è localmente consistente se soddisfa tutti i vincoli applicabili ad S, cioè i vincoli binari e unari che coinvolgono le variabili in S.

In alternativa i vincoli possono essere espressi come una coppia di disuguaglianze

$$\begin{cases} X_j - X_i \leq b_{ij} \\ X_i - X_j \leq a_{ij} \end{cases}$$

e da questo fatto si deduce che risolvere un STP è equivalente a risolvere un insieme di disequazioni lineari nelle X_i. Ci sono vari metodi per risolvere un sistema di disequazioni lineari, dal metodo del simplesso, che ha complessità esponenziale, all'algoritmo di Khachiyan , molto complicato in pratica. Fortunatamente gli STP sono caratterizzati da una classe speciale di disequazioni la quale ammette metodi risolutivi più semplici. Infatti una volta codificate le disequazioni in un grafo è sufficiente applicare a tale grafo l'algoritmo del cammino minimo. Nel campo dell'intelligenza artificiale una struttura dati simile, detta *mappa temporale*, è stata introdotta da Dean e McDermott ma non è mai stata formulata matematicamente.
Formalmente, viene associato ad un STP un grafo orientato e pesato $G_d = (V, E_d)$, detto **grafo delle distanze**, da distinguersi dal grafo dei vincoli G. Esso ha gli stessi nodi di G, e ogni arco $i \longrightarrow j$ è etichettato con un peso a_{ij}, che rappresenta il termine noto della disequazione $X_j - X_i \leq a_{ij}$.
Ogni cammino da i a j in G_d, $i_0 = i, i_1, ..., i_k = j$ induce i seguenti vincoli sulla distanza $X_j - X_i$:

$$X_j - X_i \leq \sum_{j=1}^{k} a_{i_{j-1}, i_j}.$$

Se c'è più di un cammino da i a j allora è facile verificare che l'intersezione di tutti i vincoli indotti dai vari cammini è

$$X_j - X_i \leq d_{ij}$$

dove d_{ij} è la lunghezza del cammino minimo tra i e j.
Sulla base di queste osservazioni sono state stabilite le seguenti *condizioni di consistenza* per un STP.

Teorema 2 (Shostak [37], Liao and Wong [20], Lieserson and Saxe [19]) *Dato un STP, T, esso è consistente se e solo se il suo grafo delle distanze, G_d, non ha cicli negativi.*

Enunciamo un teorema che verrà poi esteso in questa tesi.

Teorema 3 *Sia G_d il grafo delle distanze di un STP consistente. Allora i seguenti assegnamenti sono due soluzioni:*

$$S_1 = (d_{01}, ..., d_{0n})$$

$$S_1 = (-d_{10}, ..., -d_{n0})$$

I due assegnamenti corrispondono rispettivamente al verificarsi degli eventi il più presto e il più tardi possibile.

Da quanto detto è facile vedere come una rappresentazione efficace di un STP si ottenga con un grafo orientato e completo, detto ***d*-grafo** (dall'inglese complete directed graph), dove ogni arco $i \longrightarrow j$ viene etichettato con la lunghezza del cammino minimo, d_{ij}, in G_d. Enunciamo ora un teorema fondamentale per questa tesi la quale in parte si occupa proprio di estenderlo ai vincoli temporali con preferenze.

Teorema 4 (Dechter, Pearl) *Ogni STP è decomponibile rispetto ai vincoli nel proprio d-grafo.*

Il 4 è importante in quanto è alla base di un algoritmo efficiente per la costruzione di una soluzione dato un STP; l'algoritmo consiste semplicemente nell'assegnare a ciascuna variabile un qualsiasi valore che soddisfi i vincoli del *d*-grafo relativi agli assegnamenti precedenti (a partire da $X_0 = 0$). La decomponibilità garantisce l'esistenza di un tale valore, a prescindere dall'ordine degli assegnamenti. Una ulteriore conseguenza del 4 è che i domini caratterizzati dal *d*-grafo sono minimi.

Teorema 5 (corollario) *Sia G_d il grafo delle distanze di un STP consistente. L'insieme dei valori ammissibili di una variabile X_i è $[-d_{i,0}, d_{0,i}]$.*

Da quanto detto emerge che il *d*-grafo è una rappresentazione equivalente dell'STP originale, ma più stretta. È ora possibile concludere che questa nuova rete è la rete minima.

Teorema 6 (corollario) *Dato un STP, T, consistente, l'STP equivalente, M, definito come segue:*

$$\forall i, j M_{ij} = \{[-d_{j,i}, d_{i,j}]\}$$

è la rete minima di T.

Il *d*-grafo di un STP può essere costruito applicando l'algoritmo Floyd-Warshall al grafo delle distanze, trovando così il cammino minimo tra tutte le coppie.

La complessità di tale algoritmo è $O(n^3)$, ed esso inoltre è in grado di scoprire eventuali cicli negativi semplicemente guardando gli elementi diagonali d_{ii}. Floyd-Warshall, dunque, costituisce un algoritmo polinomiale che determina la consistenza di un STP e trova i domini minimi e la rete minima. Una volta che si ha a disposizione il *d*-grafo, trovare una soluzione richiede

Algoritmo Folyd-Warshall

1. for $i := 1$ to n do $d_{ii} \longleftarrow 0$;
2. for $i, j := 1$ to n do $d_{ij} \longleftarrow a_{ij}$
3. for $k := 1$ to n do
4. for $i, j := 1$ to n do
5. $\min\{d_{ij}, d_{ik} + d_{kj}\}$;

Figura 2.1: Algoritmo Floyd-Warshall.

solo $O(n^2)$, in quanto ogni assegnamento successivo deve essere controllato rispetto ai precedenti assegnamenti ed è garantito il fatto che successivamente rimane inalterato. Alternativamente le due soluzioni indicate dal 3 sono disponibili immediatamente senza ulteriori controlli.
Possiamo dunque concludere che *trovare una soluzione di un STP ha in totale una complessità di* $O(n^3)$.

Osservazioni sui Vincoli Temporali non semplici

Un modo diretto per risolvere un TCSP è quello di decomporlo in vari STP, di risolvere tali STP, e di ricombinare le soluzioni ottenute. Dato un TCSP binario, T, definiamo una *etichettatura* di T una scelta di un intervallo per ciascun vincolo. Ogni etichettatura definisce un STP i cui archi sono etichettati con gli intervalli selezionati. Possiamo dunque risolvere qualunque TCSP considerando tutti gli STP che da esso si possono ottenere. In particolare un TCSP sarà consistente se almeno una etichettatura dà origine ad un STP consistente. Inoltre *ogni soluzione di T è soluzione di uno dei suoi STP e viceversa.* È importante osservare che *la rete minima di T può essere costruita a partire dalle reti minime degli STP* come mostra il seguente teorema:

Teorema 7 *Data la rete minima, M, di un dato TCSP, T, allora si ha:*

$$M = \bigcup_l M_l$$

dove M_l *è la rete minima dell' STP definito dalla etichettatura l, e l'unione è su tutte le possibili etichettature.*

La complessità di risolvere un TCSP generando tutta le etichettature è $O(n^3 k^e)$ dove k è il massimo numero di intervalli di un vincolo ed e è il numero degli archi. In questa trattazione i risultati precedenti verranno estesi a problemi di vincoli temporali non semplici con preferenze.

2.1.3 L'algoritmo Path Consistency

Ai vincoli temporali, come a vincoli classici, è possibile applicare la tecnica risolutiva con backtracking. Essa consiste nel determinare la soddisfacibilità di problemi di vincoli scegliendo una variabile, e determinando per ogni valore del dominio di questa variabile la soddisfacibilità dei vincoli che risultano sostituendo alla medesima tale valore. Questo viene ottenuto applicando ricorsivamente questa procedura nota come *backtracking cronologico* [25].

Il fatto di imporre la consistenza locale, in questo caso intesa come consistenza sui cammini, può essere una utile pre-elaborazione al fine di diminuire i passi di backtrack. Inoltre fornisce un'ottima approssimazione che spesso porta alla rete minima. L'algoritmo *Floyd-Warshall* può essere visto come un algoritmo che procede per *rilassamenti* successivi, in quanto ad ogni passo l'etichetta di un arco viene aggiornata secondo le etichette degli archi adiacenti. Montanari [28] è stato il primo ad usare questo algoritmo nei problemi di soddisfazione di vincoli. Mackworth [21], e Mackworth e Freuder [23]hanno poi approfondito il tema.

Definizione 2 *Un cammino attraverso i nodi* $i_0, i_1, ..., i_m$ *è consistente (path consistent) se e solo se per qualsiasi coppia di valori,* v_0 *e* v_m*, tali che* $v_m - v_0 \in T_{i_0 i_m}$*, esiste una sequenza di valori,*$v_1, ..., v_{m-1}$*, tale che* $v_1 - v_0 \in T_{i_0 i_1}$*,* $v_2 - v_1 \in T_{i_1 i_2}$*,..., e* $v_m - v_{m-1} \in T_{i_{m-1} i_m}$*. Una rete è path consistent se ogni cammino è path consistent.*

Algoritmo PC-1

1. repeat
2. $S \longleftarrow T$;
3. for $k := 1$ to n do
4. for $i, j := 1$ to n do
5. $T_{ij} \longleftarrow T_{ij} \oplus T_{ik} \otimes T_{kj}$, and
6. if $T_{ij} = \emptyset$ then exit (inconsistenza);
7. until $S = T$;

Figura 2.2: Algoritmo PC-1.

Questo algoritmo propaga la consistenza locale tramite l'intersezione, $\oplus$, e la composizione, $\otimes$, di vincoli. Esso considera i triangoli di vincoli e rilassa i

vincoli fino a quando si raggiunge una determinata soglia o un vincolo diventa inconsistente. Chiaramente *la rete ottenuta con PC-1 è equivalente a quella data* . Montanari [28] ha provato che nel caso di domini discreti l'algoritmo termina e la rete finale è effettivamente consistente sui cammini. Quando si hanno domini continui tuttavia, non si può garantire che termini, ma comunque verrà ottenuta una rete limite. Ad ogni iterazione la rete corrente è più stretta di quella precedente e quindi la rete minima è la rete limite a cui l'algoritmo tende. PC-1 può essere visto come una generalizzazione di *Floyd Warshall* . Infatti il passo di rilassamento su T_{ij} è equivalente a due operazioni locali di aggiornamento del cammino minimo d_{ij}. Si può dunque asserire:

Teorema 8 (Montanari [28]) *Applicare PC-1 alla rete di un STP coincide esattamente con applicare Floyd-Warshall al grafo delle distanze.*

Un corollario immediato di questo teorema è che PC-1 termina e produce una rete consistente sui cammini. Per quanto riguarda l'applicazione di questo algoritmo ai TCSPs ci si può chiedere se termina, se la rete che si ottiene è consistente sui cammini e se è minima. È semplice provare che nel caso di TCSPs *interi*, cioè i cui intervalli hanno estremi interi, PC-1 termina. Questo è dovuto al fatto che ogni volta che l'intervallo si restringe diminuisce di una quantità intera. La stessa argomentazione si può applicare nel caso di TCSPs *razionali*, in quanto basta moltiplicare tutte le quantità per il massimo comun divisore. Questo è stato dimostrato più approfonditamente da Ladkin [18]. Poichè tutti i problemi pratici sono esprimibili con TCSPs razionali, possiamo ritenere PC-1 un algoritmo che termina. Per quanto riguarda la consistenza sui cammini, basta riapplicare la prova fornita da Montanari [28] in quanto il fatto che i domini siano continui non ha alcun effetto. È dunque possibile concludere:

Teorema 9 *L'algoritmo PC-1 produce una rete consistente sui cammini.*

Rimane la questione se la rete trovata sia o no minima. Montanari ha mostrato che quando i vincoli soddisfano la proprietà distributiva (cioè la composizione distribuisce sull'intersezione), ogni rete path consistent è al tempo stesso minima e decomponibile. In tale caso è sufficiente un'unica iterazione del ciclo principale. Se consideriamo STPs allora la proprietà distributiva vale e la rete elaborata è minima, e decomponibile e richiede una unica iterazione del ciclo principale. Un algoritmo più efficiente è PC-2, Mackworth [21].

Algoritmo PC-2

1. $Q \longleftarrow \{(i,j,k) \mid (i<j)\text{and}(k \neq i,j)\}$;
2. while $Q \neq \emptyset$ do
3. selezionare e cancellare un cammino (i,j,k) da Q
4. if $T_{ij} \neq T_{ik} \otimes T_{kj}$ then
5. $T_{ij} \longleftarrow T_{ij} \oplus (T_{ik} \otimes T_{kj})$
6. if $T_{ij} = \emptyset$ then exit (inconsistenza)
7. $Q \longleftarrow Q \cup$ RELATED_PATHS $((i,j,k))$
8. end-if
9. end-while

Figura 2.3: Algoritmo PC-2.

La funzione RELATED_PATHS$((i,j,k))$ restituisce tutti i cammini di lunghezza 2 che devono essere riconsiderati dopo l'aggiornamento di T_{ij}.

Definizione 3 *Sia $T_{ij} = \{[a_1,b_1],...,[a_n,b_n]\}$. L'ampiezza di T_{ij} è $b_n - a_1$. L'ampiezza di un TCSP è l'ampiezza massima dei suoi vincoli.*

Teorema 10 *Sia dato un TCSP T con ampiezza R espressa nell'unità di tempo più stretta. L'algoritmo PC-2 rende T consistente sui cammini in $O(n^3R)$ rilassamenti e in $O(n^3R^3)$ operazioni aritmetiche.*

Volendo fare un paragone con il backtracking cronologico, si osservi che R deve essere grande almeno quanto k, il numero di intervalli per vincolo. Tuttavia una complessità pari a $O(k^e)$ può comunque essere inferiore a $O(R^3)$ nel caso in cui gli archi siano costituiti da pochi intervalli. Anche se gli algoritmi di consistenza sui cammini non garantiscono di ottenere la rete minima, essi forniscono un'alternativa pratica e un approccio complementare alla tecnica della decomposizione. Su piccoli problemi (5-7 variabili), generati random, PC-1 ha sempre trovato la rete minima. Dunque sulla base di questi risultati appare chiaro che path consistency riduce in modo consistente il processo di backtracking.

2.2 Vincoli con Preferenze

I problemi di soddisfazione di vincoli classici, o CSP (Constraint Satisfaction Problems), sono molto espressivi e costituiscono un formalismo naturale per molti problemi pratici. Sono utili in molti campi come la visione, la robotica, la schedulazione, etc. È tuttavia evidente che non sono adatti a rappresentare problemi dove le conoscenze o le informazioni disponibili sono incomplete o imprecise. In tali casi dire se un assegnamento ad un insieme di variabili sia ammesso o no può non essere sufficiente né sensato. A fronte di queste limitazioni sono stati fatti dei tentativi per estendere i CSP.

In [30], [10] e [34] è stata aggiunta la possibilità di associare a ogni tupla rappresentante un assegnamento, o a ogni vincolo un livello di preferenza , con la possibilità di combinare i vincoli con le operazioni min-max. Si tratta dei **Fuzzy CSP** i quali costituiscono di fatto una estensione dei CSP classici.

Essi ricopriranno un ruolo fondamentale in questa trattazione. Tuttavia ci riferiremo ad essi come istanza particolare di un formalismo più ampio, sviluppato da Bistarelli, Montanari, Rossi [4] che verrà ora descritto in dettaglio.

Continuando la rassegna di estensioni ai CSP, in [11] e in [14] è presentata una tecnica per modellare la conoscenza incompleta di un problema reale, per risolvere problemi con un numero eccessivo di vincoli, e per rappresentare problemi di ottimizzazione dei costi. In [5] viene presentato un modello in cui possono essere inclusi i CSP classici stessi.

L'idea fondamentale del modello che ora prenderemo in considerazione è quella che la struttura algebrica di *semi-anello*, che verrà descritta in seguito, è sufficiente per descrivere molti schemi di soddisfazione di vincoli. Infatti il dominio del semianello fornisce i livelli di preferenze, da interpretarsi come costi, gradi di preferenze, probabilità, o altro, e le due operazioni definiscono un modo per combinare i vincoli. Precisamente viene introdotta la nozione di risolvere un problema di vincoli su un dato semi-anello. Cambiando il semianello si ottengono istanze diverse dello stesso problema, che possono corrispondere a schemi noti per la soddisfazione di vincoli o a schemi nuovi.

Tecniche basate sulla cosidetta consistenza locale [12] , [13] , [22], [28], [29], hanno una utilità provata per trovare una approssimazione di una soluzione ad un CSP classico. In [5] queste tecniche vengono estese al nuovo formalismo e vengono identificate le caratteristiche che il semi-anello deve soddisfare affinchè (1) l'algoritmo di consistenza locale termini (2) il problema ottenuto sia equivalente a quello dato e (3) il risultato sia indipendente dalle scelte non deterministiche operate durante l'esecuzione.

Il formalismo di cui si parla tratta CSP basati su semianelli **SCSP** (Semiring-based CSP), uno dei vantaggi è quello di permettere a tutti i problemi di

vincoli che rientrano in questo paradigma di ereditare i risultati e le caratteristiche originali. Ad esempio sarà immediatamente possibile verificare l'applicabilità o meno di algoritmi di consistenza locale.

2.2.1 C-semianelli

L'approccio con semianelli è volto ad estendere i CSP classici in modo da rendere possibile la gestione di nozioni non assolute e per dare una interpretazione più generale delle operazioni su nozioni di tale tipo. A tal fine si affianca alla nozione classica di CSP un semianello. L'aggiunta di questa struttura algebrica fornisce il dominio per i vari livelli di incertezza o preferenza e le operazioni per combinare tali livelli.

Definizione 4 (Semianello) *Un semianello è il dato di* $\langle A, sum, \times, \mathbf{0}, \mathbf{1}\rangle$ *dove:*

- A *è un insieme e* $\mathbf{0}, \mathbf{1} \in A$
- *sum, detto operatore additivo, è commutativo (i.e. sum(a,b)=sum(b,a)), associativo (i.e. sum(a,sum(b,c))=sum(sum(a,b),c)) e* $\mathbf{0}$ *è il suo elemento neutro (i.e. sum(a,0)=sum(0,a)=a);*
- $\times$*, detto operatore moltiplicativo, è associativo e ha* $\mathbf{1}$ *come elemento neutro e* $\mathbf{0}$ *come annichilatore (i.e.,* $a \times \mathbf{0} = \mathbf{0} \times a$*);*
- *vale la proprietà distributiva di* $\times$ *su sum (i.e.,* $\forall a, b, c \in A, a \times sum(b, c) = sum((a \times b), (a \times c))$*).*

Diamo ora la definizione di c-semianello, dove c sta per "constraint" (vincolo).

Definizione 5 (c-semianello) *Un c-semianello è il dato di* $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ *dove:*

- A *è un insieme e* $\mathbf{0}, \mathbf{1} \in A$
- $+$ *è definito sull'insieme (anche infinito) di elementi di A come segue*
  - $\forall a \in A, \sum(\{a\}) = a$*;*
  - $\sum(\emptyset) = \mathbf{0}$ *and* $\sum(A) = \mathbf{1}$*;*
  - $\sum(\bigcup A_i, i \in S) = \sum(\{\sum(A_i), i \in S\})$ *per ogni sottoinsieme S di indici (proprietà di livellamento);*
- $\times$ *è un operatore binario, commutativo e asssociativo,* $\mathbf{1}$ *è il suo elemento neutro,* $\mathbf{0}$ *l'annichilatore;*

- *vale la proprietà distributiva di* $\times$ *su* $+$ *(i.e.* $\forall a \in A$ *e* $B \subseteq A, a \times \sum(B) = \sum(\{a \times b, b \in B\})$)

In un c-semianello, quindi, l'operatore additivo è definito su un qualsiasi insieme di A, anche infinito. Questo fatto lo rende automaticamente commutativo, associativo e idempotente. È inoltre possibile mostrare che **0** è il suo elemento neutro. Questo significa che un c-semianello è un semianello con delle proprietà ulteriori. È inoltre possibile provare che **1** è l'annichilatore per $+$.

Il vantaggio di utilizzare c-semianelli invece di semianelli è il fatto che l'idempotenza dell'operatore additivo è necessaria per definire un ordine parziale $\leq_S$ su A, che ci permetterà di paragonare i vari elementi del semianello. Tale ordine è definito come segue: $a \leq_S b$ sse $a + b = b$. Intuitivamente $a \leq_S b$, significa che b è meglio di a, o, in alternativa, che tra a e b l'operatore $+$ sceglie b.

Il fatto che **0** sia l'elemento neutro dell'operatore additivo implica che **0** è l'elemento minimo rispetto all'ordinamento appena definito. Dunque $\forall a \in A$ $\mathbf{0} \leq_S a$.

È importante osservare che entrambe gli operatori sono monotoni rispetto all'ordinamento.

La commutatività di $\times$ è auspicabile quando tale operatore viene usato per combinare molti vincoli. Se non valesse tale proprietà significherebbe che ordinamenti diversi darebbero risultati diversi.

Poiché **1** è l'elemento neutro per $+$, si ha che $a \leq_S \mathbf{1} \forall a$. Quindi **1** è il massimo rispetto all'ordinamento parziale. Questo implica che $\times$ è un operatore *intensivo*, i.e., $a \times b \leq_S b$. Questo risultato è importante in quanto significa che combinare più vincoli porta ad un risultato peggiore, rispetto a $\leq_S$.

Teorema 11 *Dato un qualunque c-semianello* $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$*, si consideri la relazione* $\leq_S$ *definita su* A*. Allora* $\times$ *è intensivo, cioè,* $a, b \in A \Rightarrow a \times b \leq_S a$.

Viene mostrato ora come i c-semianelli possano essere interpretati come reticoli completi. Se, come spesso verrà richiesto anche $\times$ è idempotente, la struttura corrispondente sarà un reticolo distributivo. Si veda [DP90] per una trattazione più approfondita dei reticoli.

Teorema 12 ($\langle A, \leq_S \rangle$ è un reticolo completo) *Dato un c-semianello* $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$*, si consideri l'ordine parziale* $\leq_S$*. Allora* $\langle A, \leq_S \rangle$ *è un reticolo completo.*

Da questo teorema deriva che l'operatore additivo coincide con prendere l'inf nel reticolo completo $\langle A, \leq_S \rangle$.

Teorema 13 ($\times$ idempotente implica $\langle A, \leq_S \rangle$ distributivo e $\times$ = inf) *Da- to un c-semianello $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$, si consideri il reticolo completo corrispondente $\langle A, \leq_S \rangle$. Se $\times$ è idempotente allora si ha:*

1. *+ distribuisce su $\times$*
2. *$\times$ coincide con l'inf nel reticolo*
3. *$\langle A, \leq_S \rangle$ è un reticolo distributivo*

Si osservi che se $\times$ è idempotente e $\leq_S$ è totale, si ha che $a + b = max(a, b)$ e $a \times b = min(a, b)$.

Verranno ora ridefiniti i concetti di sistema di vincoli, vincolo, e problema di vincoli in modo parametrico rispetto al c-semianello.

Definizione 6 (Sistema di Vincoli) *Un sistema di vincoli è una terna $CS = \langle S, D, V \rangle$ dove S è un semianello, D è un insieme finito e V è un insieme ordinato di variabili.*

Un vincolo dato su un determinato sistema di vincoli specifica le variabili coinvolte e i valori ammissibili per esse. Più precisamente ad ogni tupla di valori, di D, per le variabili coinvolte viene assegnato un valore di A. Questo valore può essere come un peso, un costo o un livello di confidenza o altro.

Definizione 7 (Vincolo) *Dato un sistema di vincoli $CS = \langle S, D, V \rangle$, dove $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ un vincolo su CS è una coppia $\langle def, con \rangle$, dove:*

- *$con \subseteq V$, viene detto il* tipo *del vincolo*
- *$def : D^k \longrightarrow A$, dove k è la cardinalità di con, viene detto il* valore *del vincolo*

Un problema di vincoli sarà allora un insieme di vincoli su un dato sistema di vincoli, più un determinato insieme di variabili, il *tipo*. Queste rappresentano le variabili coinvolte nel problema, cioè quelle di cui si vogliono conoscere gli assegnamenti ammessi dall'insieme di vincoli.

Definizione 8 (Problema di vincoli) *Dato un sistema di vincoli $CS = \langle$ S,D,V $\rangle$, un problema di vincoli P su CS è una coppia $P = \langle C, con \rangle$, dove C è un insieme di vincoli su CS e $con \subseteq V$. Si assume dunque che $\langle def_1, con' \rangle \in C$ e $\langle def_2, con' \rangle \in C$ implica $def_1 = def_2$. In seguito un problema di vincoli così definito verrà indicato con* ***SCSP****.*

Quando tutte le variabili sono importanti, come accade molto spesso nel caso classico, *con* contiene tutte le variabili in ogni vincolo del problema

P. Tale insieme, chiamato $V(P)$, è un sottoinsieme di V che può essere ricavato guardando le variabili di ciascun vincolo . Formalmente si ha: $V(P) = \bigcup_{\langle def, con' \rangle \in C} con'$. Come accade per i problemi classici, è possibile rappresentare un SCSP graficamente sotto forma di ipergrafo etichettato dove i nodi corrispondono alle variabili, gli iperarchi ai vincoli, e ogni iperarco è etichettato dalla definizione del vincolo corrispondente (il quale può a sua volta essere rappresentato da un insieme di coppie $\langle n - pla, valore \rangle$. Le variabili di interesse possono poi essere messe in evidenza nel grafo.

Si osservi che la definizione data è parametrica rispetto al sistema di vincoli CS e quindi rispetto al c-semianello.

Nei SCSPs, i valori specificati per le n-uple di ciascun vincolo sono usati per calcolare i valori corrispondenti per le variabili in *con*, secondo le operazioni del semianello: l'operatore moltiplicativo è usato per combinare i valori delle tuple di ciascun vincolo per ottenere un valore per una tupla per tutte le variabili, l'operatore additivo, invece, serve ad ottenere il valore per una tupla di variabili nel tipo del problema. Più precisamente si possono definire due operazioni sui vincoli:

- *combinazione di vincoli* ($\otimes$);
- *proiezione sui vincoli* ($\Downarrow$).

Analoghe operazioni sono state definite per relazioni "fuzzy" [Zad75] e sono state utilizzate nei CSP fuzzy in [DFP93]. La definizione qui considerata è più generale in quanto non si riferisce a nessun c-semianello in particolare.

Definizione 9 (combinazione) *Dato un sistema di vincoli* $CS = \langle S, D, V \rangle$ *dove* $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$, *e due vincoli* $c_1 = \langle def_1, con_1 \rangle$ *e* $c_2 = \langle def_2, con_2 \rangle$ *definiti su CS, la loro combinazione si indica* $c_1 \otimes c_2$ *ed è in vincolo* $c = \langle def, con \rangle$ *dove*

$$con = con_1 \cup con_2$$

e

$$def(t) = def_1(t \downarrow_{con_1}^{con}) \times def_2(t \downarrow_{con_2}^{con}).$$

Essendo $\times$ sia commutativo che associativo anche $\otimes$ gode delle stesse proprietà.

Definizione 10 (proiezione) *Dato un sistema di vincoli* $CS = \langle S, D, V \rangle$, *dove* $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$, *e un vincolo* $c = \langle def, con \rangle$ *su CS, e un insieme I di variabili* ($I \subseteq V$), *la proiezione di c su I, si indica* $c \downarrow_I$ *ed è il vincolo* $\langle def', con' \rangle$ *su CS dove*

$$con' = I \cap con$$

e

$$def'(t') = \sum_{\{t|t\downarrow^{con}_{I\cap con}\}} def(t)$$

Utilizzando le nozioni di combinazione e proiezione di vincoli possiamo ora definire cos'è una soluzione di un SCSP. In seguito verrà sottointeso $CS = \langle S, D, V\rangle$ e $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$.

Definizione 11 (soluzione) *Dato un problema di vincoli $P = \langle C, con\rangle$ definito su un sistema di vincoli CS, la soluzione di P è un vincolo così definito:*

$$Sol(P) = (\bigotimes C) \Downarrow_{con} .$$

La soluzione di un SCSP è, dunque, il vincolo indotto sulle variabili di *con* da tutto il problema. Tale vincolo associa ad ogni tupla di valori di D per le variabili di *con*, un valore di A. Può talvolta essere sufficiente conoscere i miglior valore associato a tali tuple. Questo può a sua volta essere rappresentato da un vincolo su un insieme di variabili vuoto, e verrà chiamato il livello ottimo di consistenza di tutto il problema, dove il significato di "ottimo" dipende dall'ordinamento $\leq_S$ definito dall'operatore additivo.

Definizione 12 (livello ottimo di consistenza) *Dato un SCSP $P = \langle C,$ con $\rangle$, definiamo $blevel(P) \in S$ tale che $\langle blevel(P), \emptyset\rangle = (\bigotimes C) \Downarrow \emptyset$. Inoltre, P verrà detto consistente se $\mathbf{0} <_S blevel(P)$, e P è $\alpha-$consistente se $blevel(P) = \alpha$.*

Intuitivamente il livello ottimo di consistenza dà un'idea di quanto possiamo soddisfare i vincoli del problema. Si osservi che $blevel(P)$ non dipende dalla scelta delle variabili notevoli per la proprietà associativa dell'operatore additivo. Questo significa che, poiché un problema di vincoli è costituito da un insieme di vincoli e delle variabili di interesse, quest'ultime possono essere trascurate nel calcolo di *blevel*. Formalmente $blevel(C)$ indicherà $belevel(\langle C, con\rangle)\forall con$. Si osservi che $blevel(P)$ si può ottenere prima calcolando la soluzione e poi proiettandola, come vincolo, sull'insieme vuoto.

Proposizione 1 *Dato un SCSP P, si ha che $Sol(P) \Downarrow_\emptyset = \langle blevel(P), \emptyset\rangle$.*

2.2.2 Consistenza Locale

Calcolare il livello ottimo di consistenza, trovare una soluzione, o una soluzione astratta sono problemi NP-completi. Può dunque essere utile trovare una approssimazione delle cose cercate. Nei CSPs classici questo è fatto tramite tecniche di consistenza locale. In pratica si individuano alcuni sottoproblemi nei quali si eliminano incosistenze locali e si procede in questo modo,

selezionando sottoproblemi e rendendoli consistenti fino a quando non viene raggiunta la stabilità. Gli algoritmi che seguono questa procedura più noti sono la consistenza sugli archi [21], dove i sottoproblemi sono costituiti da un unico vincolo e la consistenza sui cammini[28] dove i problemi sono triangoli (grafi di tre varaiabili binari e completi). Il concetto di k-consistenza [12], in cui i sottoproblemi sono costituiti da tutti i vincoli che sussistono tra le k variabili scelte, generalizza entrambe questi algoritmi. Anche in questo schema, tuttavia, i sottoproblemi hanno la stessa grandezza, e ciò è una limitazione se si vogliono considerare sottoproblemi con dimensioni diverse. Questa generalizzazione è stata introdotta per i CSPs classici in [29] e verrà ora seguito il medesimo schema per SCSPs. In parole povere, applicare un algoritmo di consistenza locale ad un problema di vincoli significa esplicitare alcuni vincoli impliciti, e dunque scoprire l'esistenza di eventuali inconsistenze locali. Questo vale anche per i SCSPs.

Per poter identificare i sottoproblemi da considerare nella consistenza locale viene utilizzata la nozione di regola di consistenza locale. L'applicazione di tale regola consiste nel risolvere uno dei sottoproblemi. Per capire come avviene questo è necessario introdurre un concetto nuovo: *locazione tipata*. Informalmente una locazione tipata è semplicemente una locazione l (come nei linguaggi di programmazione store-based) che ha un insieme di variabili con come tipo, e quindi a cui si può solo assegnare il vincolo $c = \langle def, con \rangle$ avente lo stesso tipo. In quanto segue verrà assunto di avere una locazione per ogni insieme di variabili e dunque la locazione verrà identificata con il tipo. Si osservi che ad ogni punto dell'esecuzione del programma la memoria conterrà un problema di vincoli con un numero finito di vincoli, e dunque le locazioni di interesse saranno sempre in numero finito.

Definizione 13 (Locazione tipata) *Una locazione tipata l è un insieme di variabili.*

Definizione 14 (Valore di una locazione) *Dato un CSP $P = \langle C, con \rangle$, il valore $[l]_P$ della locazione tipata l in P è definito come il vincolo $\langle def, con \rangle \in C$ se esiste, $\langle 1, l \rangle$ altrimenti. Date n locazioni $l_1, \cdots, l_n$ il valore $[l_1, \cdots, l_n]_P$ di tali locazioni tipate in P è definito invece come i l'insieme di vincoli $\{[l_1]_P, \cdots, [l_n]_P\}$.*

Definizione 15 (Assegnamento) *Un assegnamento è una coppia $l := c$ dove $c = \langle def, con \rangle$. Dato un CSP $P = \langle C, con \rangle$, il risultato dell'assegnamento $l := c$ è il problema $[l := c](P)$ definito come:*

$$[l := c](P) = \langle \{\langle def', con' \rangle \in C \mid con' \neq l\} \cup c, con \rangle$$

L'assegnamento $l := c$ è può essere visto come una funzione che mappa problemi di vincoli in problemi di vincoli, che cambia un dato problema

modificandone un solo vincolo, quello di tipo l. La modifica consiste nel rimpiazzamento di tale vincolo con il vincolo c. Nel caso in cui non ci sia nessun vincolo di tipo l, il vincolo c è aggiunto al problema originario.

Definizione 16 (Regola di consistenza locale) *Dato un SCSP $P = \langle C, con\rangle$, una regola di consistenza locale r per P è definita come $r = l \leftarrow L$, dove L è un insieme di locazioni, l è una locazione e $l \notin L$.*

Applicare una regola di consistenza locale r a P significa assegnare alla locazione l il vincolo ottenuto risolvendo il sottoproblema di P contenente i vincoli specificati dalle locazioni si $L \cup \{l\}$.

Definizione 17 (Applicare una regola) *Dato una regola di consistenza locale $r = l \leftarrow L$ e un problema di vincoli P, applicando r a P si ottiene*

$$[l \leftarrow L](P) = [l := Sol(\langle [L \cup l_P, l\rangle)](P)$$

Poiché applicare una regola è definito come una funzione da problemi in problemi, l'applicazione di una sequenza S di regole ad un problema si può considerare come la composizione di funzioni. Da cui si ha:

$$[r_1; S](P) = [S]([r_1](P))$$

Si noti, ad esempio, che $[l \leftarrow \emptyset](P) = P$.

Teorema 14 (Regole equivalenti) *Dato un SCSP $P = \langle C, con\rangle$, e una regola r per P, si ha che $P \equiv [r](P)$ se $\times$ è idempotente.*

Definizione 18 (Problema stabile) *Dato un SCSP $P = \langle C, con\rangle$ e un insieme R di regole di consistenza locale per P, P è stabile rispetto a R se per ogni $r \in R, [r](P) = P$.*

È già stato indicato come un algoritmo di consistenza locale termini quando viene raggiunta la stabilità del problema rispetto alle regole. Ad ogni passo dell'algoritmo un'unica regola viene applicata. Le regole verranno, dunque applicate in un certo ordine che verrà indicato come strategia.

Definizione 19 (Strategia) *Dato un insieme R di regole di consistenza locale per un SCSP, una strategia per R, è una sequenza infinita $S \in R^\infty$ Una strategia S è ragionevole se ogni regola di R compare in S infinite volte.*

Definizione 20 (Algoritmo di consistenza locale) *Dato un SCSP P, un insieme di regole di consistenza locale R per P, e una strategia ragionevole S per R, un algoritmo di consistenza locale applica a P le regole in R nell'ordine dato da S. Quindi, se $S = s_1 s_2 s_3 \cdots$, il problema risultante è*

$$P' = [s_1; s_2; s_3; \cdots](P)$$

L'algoritmo si ferma quando il SCSP corrente è stabile rispetto a R.

Si noti che questa definizione di consistenza locale estende di fatto quella usuale per gli algoritmi di k-consistenza. Questi ultimi, infatti, possono essere visti come algoritmi di consistenza locale in cui le regole di R hanno la forma $l \leftarrow L$, con $L = \{l_1, \cdots, l_n\}$ e $\mid \bigcup_{i=1,\cdots n} l_i \mid = \mid l \mid = k-1$. Esattamente $k-1$ variabili, dunque, sono coinvolte nelle locazioni in L.

Quando un algoritmo di consistenza locale termina, si ottiene un nuovo problema che ha lo stesso grafo di quello iniziale con possibilmente nuovi archi corrispondenti ad eventuali nuovi vincoli introdotti, ma in cui la definizione di altri vincoli può essere cambiata. Si assuma $R = \{r_1, \cdots, r_n\}$ e che $r_i = l_i \leftarrow L_i \forall i = 1, \cdots, n$. L'algoritmo utilizzerà, tra le altre cose, N locazioni tipate l_i di tipo con_i. Supponiamo che ciascuna di queste locazioni tipate abbia valore def_i quando l'algoritmo termina. Si consideri inoltre l'insieme di vincoli $C(R)$ con tipo con_i, cioè, $C(R) = \{\langle def, con\rangle$ t.c. $con = con_i \exists i \in \{1, \cdots, n\}\}$. Questi sono vincoli che potrebbero essere stati modificati dal meccanismo delle locazioni tipate dell'algoritmo. Se, poi, il SCSP iniziale era $P = \langle C, con\rangle$, il SCSP risultante sarà $P' = local - cons(P, R, S) = \langle C', con\rangle$ dove $C' = C - C(R) \cup (\bigcup_{i=1,\cdots,n} \langle def_i, con_i\rangle)$.
Nei CSP classici tutti gli algoritmi di consistenza locale godono di alcune proprietà importanti:

1. Ogni algoritmo di consistenza locale restituisce un problema equivalente a quello dato;

2. Ogni algoritmo di consistenza locale termina in un numero finito di passi;

3. La strategia, purchè ragionevole, non influenza il risultato.

Ora verrà considerata la validità di tali proprietà nell'ambito dei SCSPs.

Teorema 15 (Equivalenza [4]) *Si consideri un SCSP P e il problema $P' = local - cons(P, R, S)$. Allora $P \equiv P'$ se l'operatore moltiplicativo $(\times)$ del semianello è idempotente.*

Teorema 16 (Terminazione [4]) *Si consideri un SCSP $P = \langle C, con\rangle$ su un sistema di vincoli $CS = \langle S, D, V\rangle$ e l'insieme $AD = \bigcup_{\langle def, con\rangle} R(def)$ dove $R(def) = \{a \mid \exists t, def(t) = a\}$. Se valgono queste ipotesi l'applicazione di un algoritmo di consistenza a P termina in un numero finito di passi se AD è contenuto in un insieme I finito e tale che $+$ e $\times$ sono I-chiusi.*

Teorema 17 ([4]) *Si consideri un qualunque problema P su un sistema di vincoli $CS = \langle S, D, V\rangle$ dove $A = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ con A finito. Allora l'applicazione di un algoritmo di consistenza locale a P termina in un numero finito di passi.*

Teorema 18 (Indipendenza dalla strategia [4]) *Si consideri un SCSP P e due differenti applicazioni dello stesso algoritmo di consistenza locale a P. Si supponga che dalla prima si ottiene $P' = local - cons(P, R, S)$ e dalla seconda $P'' = local - cons(P, R, S')$. Allora $P' = P''$ se l'operatore moltiplicativo del semianello $(\times)$ è idempotente.*

Da questi teoremi si evince l'importanza dell'idempotenza di $\times$ per la consistenza locale. Nonostante ciò, si osservi che non vi è alcuna restrizione sull'ordine $\leq_S$, il quale può anche essere parziale. Si osservi anche come la definizione di applicazione di una regola implichi che l'algoritmo cambia le definizioni dei vincoli in un modo molto specifico. Infatti, anche nel caso più generale in cui $\leq_S$ è parziale, i nuovi valori assegnati alle tuple sono sempre minori o uguali a quelli precedenti. In altre parole gli algoritmi di consistenza locale non "saltano" nella struttura dell'ordine parziale da un certo valore ad un altro che non sia confrontabile con il precedente.

2.2.3 Istanze di SCSPs

CSPs classici

Un problema di soddisfazione di vincoli classico è costituito, semplicemente, da un insieme di variabili e da uno di vincoli, in cui ogni vincolo specifica le tuple che sono ammesse per quanto riguarda le variabili coinvolte. Se si assume la presenza di un sottoinsieme di variabili di particolare importanza, la soluzione di un CSP è l'insieme di tuple che rappresentano gli assegnamenti a tali variabili che possono essere estesi ad assegnamenti totali soddisfando tutti i vincoli. Nei CSPs una tupla o è ammessa o non lo è, dunque sono necessari due soli livelli di preferenza 1 e 0: le tuple ammesse verranno associate al valore 1, quelle non ammesse al valore 0. Nei CSPs classici la composizione di vincoli viene ottenuta combinando le tuple ammesse. Questo può essere ottenuto facendo corrispondere l'operatore moltiplicativo alla congiunzione logica. Per rappresentare la proiezione su alcune variabili, ad esempio su quelle notevoli, si puó assumere che l'operatore additivo sia la disgiunzione logica. Dunque il semianello caratteristico per CSP classici sarà:

$$S_{CSP} = \langle \{0, 1\}, \vee, \wedge, 0, 1 \rangle$$

È facile provare che S_{CSP} è effettivamente un semianello. L'ordine $\leq_S$ viene ridotto a $0 \leq_S 1$. Essendo $\wedge$ idempotente, applicano i teoremi provati, si ritrovano proprietà dei CSPs note, cioè che applicando un algoritmo di consistenza locale si ottiene un problema equivalente, e inoltre la strategia è irrilevante. Inoltre dal fatto che $\{0, 1\}$ è ovviamente finito si ottiene che

l'algoritmo termina in un numero finito di passi. Una rappresentazione alternativa è quella con il c-semianello $\langle \wp(\{a\}), \bigcup, \bigcap, \emptyset, \{a\} \rangle$ dove $\wp(S)$ è l'insieme delle parti di S e a un qualsiasi valore.

Fuzzy CSPs

I CSPs Fuzzy, indicati anche con FCSPs, [30], [10] [34], sono una estensione dei CSPs classici che permette di trattare vincoli non categorici, cioè vincoli che associano un determinato livello di preferenza a ciascuna tupla di valori. Tale valore è sempre compreso tra 0 e 1, dove 1 rappresenta il livello ottimo (tupla ammessa) e 0 quello pessimo (tupla non ammessa). La soluzione di un FCSP è definita come l'insieme delle tuple di valori, per tutte le variabili, che hanno il valore massimo. Il modo in cui viene associato un valore ad una n-upla è prendendo il minimo tra valori associati a tutte le sue sottotuple. Il significato dello schema **max-min**, che sarà fondamentale in questa trattazione, è quello di massimizzare il peggior livello associato ad una tupla. In parole povere si vuole ottimizzare il caso pessimo. FCSPs sono una estensione molto espressiva dei CSPs classici in quanto possono gestire problemi di soddisfazione parziale di vincoli [FW92], problemi sovra-vincolati e problemi in cui vi sono vari livelli di priorità tra i vincoli. Il semianello associato ai FCSPs nello schema dei SCSPs è

$$S_{FCSP} = \langle \{x \mid x \in [0,1]\}, max, min, 0, 1 \rangle$$

L'ordinamento $\leq_S$ si riduce all'ordinamento $\leq$ sui reali. I problemi di soddisfazione di vincoli fuzzy sono, evidentemente, più vicini allo schema dei SCSPs di quanto lo siano i problemi classici in quanto hanno già in sè la nozione di preferenza. Tuttavia il fatto che il sistema presentato non faccia assunzioni su quale debba essere il semianello lo rende molto più generale. Vediamo quali delle proprietà enunciate per i SCSPs vengono ereditate dai FCSPs. **L'operatore moltiplicativo, min, è idempotente**. Quindi applicando un algoritmo di consistenza locale si ottiene un problema equivalente e la strategia non ha alcuna influenza sul risultato. Inoltre *min* è AD-chiuso per ogni sottoinsieme AD di [0,1], e dunque l'algoritmo termina. È importante osservare come, pur essendo i FCSPs una estensione notevole dei problemi classici, ad essi si possono applicare gli stessi algoritmi. Le complessità, saranno tuttavia diverse, in quanto i livelli di preferenza sono più di due. Si può provare, a questo riguardo che, se i livelli di preferenza utilizzati sono p allora la complessità di un algoritmo di consistenza locale è $O(p)$ volte quella sugli CSPs, [10].

CSPs pesati

Mentre i CSPs fuzzy associano un livello di preferenza ad ogni tupla in ogni vincolo, nei CSPs pesati alle tuple viene associato un costo. Questo è particolarmente adatto a modellare dei problemi di ottimizzazione in cui lo scopo è quello di minimizzare il costo totale (tempo, spazio, numero di risorse,...) di una data soluzione. Nei WCSPs, dall'inglese weighted, la funzione di costo è definita sommando i costi di tutti i vincoli (intesi come costi delle tuple scelte per ciascun vincolo). Quindi il goal è quello di trovare le n-uple (dove n è il numero di vincoli) che minimizzano la somma totale dei costi delle loro sottotuple.

Un altro modo per confrontare i FCSPs con i WCSPs è quello di osservare che mentre i FCSPs hanno un approccio egualitaristico ai problemi di ottimizzazione, nel senso che lo scopo è di massimizzare il livello complessivo di consistenza bilanciando i livelli dei vari vincoli, i WCSPs l'approccio è utilitaristico, in quanto lo scopo diventa quello di ottenere il minimo costo globalmente, anche trascurando alcuni vincoli che posso perciò avere un costo molto alto.

Si può dunque concludere che il semianello che meglio rappresenta problemi di vincoli pesati è:

$$S_{WCSP} = \langle \Re^+, min, +, +\infty, 0 \rangle$$

L'ordinamento $\leq_S$ si riduce all'ordinamento sui reali $\geq$. Questo fa notare come i valori preferiti siano quelli minori. L'operatore moltiplicativo del c-semianello, $+$, non è idempotente e questo, come già osservato per i Prob-CSPs, rende difficile applicare gli algoritmi di consistenza locale.

2.3 Vincoli temporali con preferenze

La soluzione e l'apprendimento di vincoli temporali con preferenze sono l'oggetto di questa tesi sia da un punto di vista teorico che da un punto di vista pratico.

I vincoli temporali con preferenze [17] sono un modello che serve per descrivere problemi in cui sono coinvolte decisioni temporali. Una *decisione temporale* è una scelta fatta su quanto dovrebbe durare un'attività, quando questa attività dovrebbe avvenire, in che ordine deve porsi rispetto ad altre attività.

Per esempio, l'antenna di un satellite orbitante come Landsat 7 deve essere ruotata in modo tale che punti ad una stazione terrestre per rendere possibile la registrazione di dati scientifici o telemetrici. Si assuma che nella schedulazione giornaliera di Landsat 7 venga aperta una finestra $W = [s, e]$ entro la

quale può cominciare la rotazione di una delle antenne verso il puntamento ad una stazione terrestre. Dare questa finestra equivale a rendere disponibili varie scelte per l'inizio del puntamento. È stato constatato che le immagini prese con lo strumento di scansione contemporaneamente al movimento dell'antenna vengono danneggiate da quest'ultimo. Dunque sarà preferibile che non ci sia sovrapposizione tra puntamenti e scansioni, tuttavia avendo l'effetto detrimentale sull'immagine carattere intermittente questa situazione non viene descritta in modo soddisfacente da vincoli hard. Quello che si vuole esprimere è infatti una preferenza su un istante t della finestra W tale che se la rotazione dell'antenna ha inizio a t allora non si hanno sovrapposizioni con l'attività dello scanner. Ovviamente è necessario tenere in conto tutti gli effetti che provocherebbe porre l'inizio della rotazione a t. Ad esempio scegliere t anzichè un istante anteriore in W potrebbe risultare in un tempo di contatto con la stazione terrestre minore e questo potrebbe limitare la quantità di dati che possono essere trasmessi. In altre parole, la scelta di cominciare il puntamento a t potrebbe essere in conflitto con la preferenza della stazione terrestre rivolta a scelte che permettano di ottenere più dati possibili.

Questo è un esempio di problema che il formalismo dei vincoli temporali con preferenze si propone di rappresentare.

Problemi di vincoli temporali con preferenze, **TCSPPs**, sono ottenuti dalla composizione di due formalismi già noti: problemi di soddisfazione di vincoli temporali, TCSPs che abbiamo descritto alla sezione 2.1 [8] e problemi di soddisfazione di vincoli con preferenze [4], SCSPs che abbiamo descritto nella sezione precedente. Un *vincolo temporale soft* è una coppia costituita da un insieme di intervalli disgiunti ed una funzione di preferenza: $\langle\{[a_1, b_1], \cdots, [a_n, b_n]\}, f\rangle$ in cui si assume $b_i < a_{i+1}$ e ponendo $I = [a_1, b_n]$ si ha $f : I \longrightarrow A$ dove A è un insieme di valori di preferenze. In analogia con i TCSPs l'insieme di intervalli che compone un vincolo temporale soft rappresenta la restrizione o degli inizi degli eventi, nel qual caso si avrà un vincolo unario, oppure della distanza tra gli eventi, nel qual caso si avrà un vincolo binario. Ricordiamo il significato temporale di questi vincoli[3]. Ad ogni evento X vengono fatte corrispondere due variabili X_s e X_e, le quali rappresentano rispettivamente l'inizio e la fine dell'attività. Porre un vincolo su quando può avere inizio X significa dare un vincolo unario sulla variabile X_s in cui l'insieme di intervalli sta a significare che devono essere soddisfatte le seguenti disequazioni: $(a_1 \leq X_s \leq b_1) \vee \cdots \vee (a_n \leq X_s \leq b_n)$. Porre invece un vincolo sulle possibili distanze che possono intercorrere tra due eventi significa porre un insieme di intervalli che corrisponde a delle disequazioni tra l'inizio o la fine di un evento e l'inizio o la fine dell'altro. Per uniformare i vincoli unari a quelli binari, viene introdotta una variabile fittizia, X_0, che

[3]Il significato temporale sarà analogo a quello dei TCSPs dai quali viene ereditato.

indica l'inizio del mondo. Questo permettere di trasformare il vincolo unario su X_s nel vincolo binario $X_s - X_0$.

Una *soluzione* di un TCSPP è un assegnamento di tutte le variabili che soddisfa tutti i vincoli. Ogni soluzione ha un *valore di preferenza globale*, ottenuto combinando i valori di preferenza locali posti sui singoli vincoli. Per formalizzare il processo di combinazione di preferenze locali in preferenze globali, e di confronto delle soluzioni viene utilizzata la struttura del c-semianello che i TCSPPs ereditano dalla loro componente relativa ai SCSPs [4]. L'ordine sull'insieme A del c-semianello, $\langle A, +, \times, 0, 1\rangle$ viene indotto dall'operatore additivo, come abbiamo visto nella sezione precedente, in modo tale che $a \leq_S b$ se e solo se $a + b = b$, significando con ciò che b è da preferirsi ad a.

Una volta che sia stato scelto, dunque, un c-semianello, e quindi un insieme di preferenze A, ogni funzione di preferenza f associata al vincolo $\langle I, f\rangle$, mappa ogni elemento di I in uno di A. Le operazioni sul semianello permettono di associare un valore di preferenza alle soluzioni, i.e. assegnamenti completi, attraverso i valori associati localmente. Infatti, sia s una soluzione di un TCSPP con semianello $\langle A, +, \times, 0, 1\rangle$ e sia $T_{ij} = \langle I_{ij}, f_{ij}\rangle$ un vincolo temporale soft sulle variabili X_i, X_j, e sia (v_i, v_j) la proiezione di s sui valori assegnati alle variabili X_i e X_j, cioè $(v_i, v_j) = s \downarrow_{X_i,X_j}$. Allora f_{ij} verrà applicata a $v_j - v_i \in I_{ij}$ ottenendo $f_{ij}(v_j - v_i)$. Sulla base di questo il valore di preferenza globale assegnato alla solzione s sarà:

$$val(s) = \prod\{f_{ij}(v_j - v_i) | (v_i, v_j) = s \downarrow_{X_i,X_j}\}.$$

Le soluzioni ottime dei TCSPPs sono le soluzioni a cui è associato il miglior livello di preferenza, dove "migliore" va inteso rispetto all'ordinamento sull'insieme del c-semianello. Se per esempio viene utilizzato il semianello fuzzy [4] $S_{FCSP} = \langle [0, 1], max, min, 0, 1\rangle$ il valore di preferenza associato ad una soluzione sarà il minimo tra tutte i valori di preferenza associati alle proiezioni di tale soluzione su tutti i vincoli.

Un altro esempio può essere quello del c-semianello del CSPs classici, $S_{CSP} = \langle \{false, true\}, \vee, \wedge, false, true\rangle$. In questo caso i valori di preferenza possibili sono solo 2: $true$ e $false$ e il valore globale di preferenza di una soluzione sarà determinato dalla congiunzione logica delle preferenze locali. Le soluzioni ottime saranno quelle a cui è associata la preferenza $true$ essendo essa migliore di $false$ nell'ordine indotto da $\vee$. In questo caso il c-semianello ha permesso una interpretazione dei TCSPs classici come un caso particolare di TCSPPs.

Data una rete di vincoli è spesso utile trovare la rete minima in cui sono state tolte tutte le informazioni ridondanti. La rete minima [8] si può ottenere attraverso algoritmi di consistenza locale. Per i TCSPP è possibile dare la

definizione di *consistenza sui cammini*. Dati due vincoli temporali soft $\langle I_1, f_1\rangle$ e $\langle I_2, f_2\rangle$ e un c-semianello S si definisce:

- *intersezione* di due vincoli temporali soft $T_1 = \langle I_1, f_1\rangle$ e $T_2 = \langle I_2, f_2\rangle$, indicata con $T_1 \oplus_S T_2$, il vincolo temporale soft $\langle I_1 \oplus I_2, f\rangle$, dove:
  - $I_1 \oplus I_2$ è l'intervallo ottenuto facendo l'intersezione dei due intervalli;
  - $f(a) = f(a) \times f(a)$ per tutte le $a \in I_1 \oplus I_2$.
- *composizione* di due vincoli temporali soft $T_1 = \langle I_1, f_1\rangle$ e $T_2 = \langle I_2, f_2\rangle$, indicata con $T_1 \otimes_S T_2$, è il vincolo temporale soft $T = \langle I_1 \otimes I_2, f\rangle$ dove:
  - $r \in I_1 \otimes I_2$ se e solo se esistono $t_1 \in I_1$ e $t_2 \in I_2$ tali che $r = t_1 + t_2$;
  - $f(a) = \sum\{f_1(a_1) \times f_2(a_2) | a = a_1 + a_2, a_1 \in I_1, a_2 \in I_2\}$.

Un vincolo indotto dai cammini sulle variabili X_i e X_j è:

$$R_{ij}^{path} = \oplus_S \forall k (T_{ik} \otimes_S T_{kj}).$$

Esso è il vincolo che si ottiene applicando $\oplus_S$ a tutti i modi di comporre un cammino di lunghezza 2 tra i e j. Un vincolo T_{ij} è consistente sui cammini se $T_{ij} \subset R_{ij}^{path}$, in altre parole se esso è stretto almeno quanto R_{ij}^{path}. Un TCSPP è consistente sui cammini se tutti i suoi vincoli lo sono.

Se l'operatore moltiplicativo del semianello è idempotente, è facile dimostrare che applicare l'operazione di rilassamento:

$$T_{ij} := T_{ij} \oplus_S (T_{ik} \otimes_S T_{kj})$$

a qualsiasi vincolo di un TCSPP restituisce un TCSPP equivalente. Inoltre l'idempotenza garantisce anche il fatto che la strategia, cioè l'ordine con cui vengono fatti i rilassamenti non è influente. Applicare la consistenza sui cammini ha una complessità di $O(n^3R^3)$ dove n è il numero delle variabili e R è il range dei vincoli [8]. Una classe particolare di TCSPPs è quella in cui tutti i vincoli sono costituiti da un unico intervallo. Questi problemi vengono chiamati problemi temporali semplici con preferenze, **STPPs**. Si hanno i seguenti risultati sulla complessità di risolvere un TCSPs e un STPPs [17].

Teorema 19 *[17] Risolvere un TCSPP è un problema NP-hard.*

Questo si ottiene facilmente considerando che risolvere un TCSP è NP-hard [8], e osservando che l'aggiunta delle preferenze non può che far aumentare la complessità.

Teorema 20 *[17] Risolvere un STPP è un problema NP-hard.*

Questo si ottiene dimostrando che un qualunque TCSP può essere ridotto ad un STPP avente sui vincoli come intervallo l'unione degli intervalli del vincolo corrispondente e come funzione di preferenza la costante a 1.

2.3.1 STPPs semi-convessi

È facile osservare come la complessità di risolvere un STPP è correlata al semianello e alla forma delle funzioni di preferenza.

In [17] viene introdotta una classe di funzioni che, qualora venga adottata per le funzioni di preferenza di un STPP, porta ad avere dei problemi risolvibili in tempo polinomiale. Si osservi prima che se tutte le funzioni di preferenza sono lineari e le operazioni del semianello mantengono la linearità allora si può trasformare la risoluzione di un tale STPP in un problema di programmazione lineare, che si può risolvere in tempo polinomiale [6]. Sia infatti dato un TCSPP. Per ogni coppia di variabili X e Y si prenda ciascun intervallo, $[a, b]$ del vincolo tra X e Y con la funzione di preferenza associata f. L'informazione fornita da ciascuno di questi intervalli può essere rappresentata dalle seguenti disequazioni: $X - Y \leq b$ e $Y - X \leq -a$ e $f_{X<Y} = c_1(X - Y) + c_2$. Se come semianello viene scelto quello fuzzy S_{FCSP} la preferenza globale V soddisferà la disuguaglianza $V \leq f_{X,Y}$ per ogni funzione di preferenza $f_{X,Y}$ definita nel problema e l'obbiettivo sarà $max(V)$. Se invece come semianello viene scelto $\langle \Re, min, +, +\infty, 0 \rangle$, l'obbiettivo diventa quello di minimizzare la somma dei livelli di preferenza e si ha $V = f_1 + \cdots + f_n$ e l'obiettivo è $min(V)$. Come si vede in entrambi i casi l'insieme di formule ottenute costituisce un problema di programmazione lineare.

Sicuramente molte situazioni possono essere rappresentate da funzioni di preferenza lineari. Tuttavia in molti casi esse non sono sufficientemente espressive. Un esempio banale potrebbe essere voler rappresentare un preferenza dei valori vicini ad un determinato istante. Un altro caso è quello in cui i valori da preferirsi sono molto vicini ad un determinato valore per la distanza e ci sono degli intervalli in cui il livello di preferenza si mantiene costante. In questo caso la rappresentazione più efficace è quella attraverso delle funzioni a scalino, le quali non sono lineari.

La classe di funzioni proposta in [17] include sia il caso di funzioni lineari sia quelle a scalino. Le funzioni contenute in tale classe sono dette *funzioni semi-convesse*. La proprietà che le caratterizza è la seguente: se esse vengono tagliate ad un certo livello da una linea orizzontale, che corrisponde ad una funzione costante K, allora l'insieme dei valori che vengono mappati dalla funzione in valori maggiori o uguali a K costituiscono un intervallo. In altre parole $\exists a, b$ tali che $\{X, f(X) \geq K\} = [a, b]$. Questa classe include funzioni non lineari e a scalino come ad esempio la funzione che assegna preferenze più alte a valori vicini ad un istante q: essa potrebbe essere $f(x) = x$ se $x \leq q$ e $f(x) = 2q - x$ per $x > k$. In figura riportiamo alcuni esempi di funzioni semi-convesse e alcune non semi-convesse.

Sempre in [17] sono state dimostrate le seguenti proprietà di questa classe

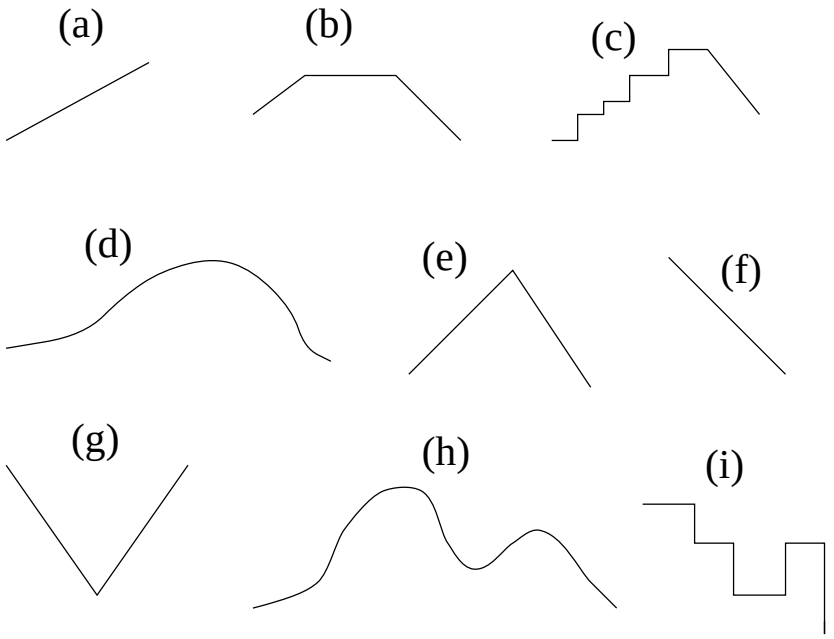

Figura 2.4: Esempi di funzioni semi-convesse (a)-(f) e non-semi-convesse (g)-(i).

di funzioni che saranno molto utili in seguito.

Teorema 21 *La classe delle funzioni semi-convesse è chiusa rispetto all'intersezione. Cioè, date due funzioni semi-convesse f_1 e f_2 con immagine in un insieme totalmente ordinato di un semianello, sia f definita come $f(a) = f_1(a) \times f_2(a)$ dove $\times$ è l'operatore moltiplicativo del semianello. Allora f è una funzione semi-convessa a sua volta.*

Teorema 22 *La classe delle funzioni semi-convesse è chiusa rispetto alla composizione. Questo significa che dato un insieme totalmente ordinato di un semianello con un operatore moltiplicativo idempotente, se f_1 e f_2 sono due funzioni semi-convesse a valori nel semianello, allora f così definita*

$$f(a) = \sum_{b+c=a} (f_1(b) \times f_2(c))$$

è a sua volta semi-convessa.

Come già osservato, in [17] viene dimostrato come STPPs in cui le funzioni di preferenza siano tutte semiconvesse sono risolvibili in tempo polinomiale. Osserviamo come sia possibile associare ad un STPP semi-convesso vari STPs che corrispondono a vari livelli di taglio effettuati sulle funzioni. Più precisamente consideriamo una funzione costante $y = k$. Allora si può ottenere da

un STPP un STP che indicheremo con STP_k nel seguente modo: l'intervallo associato a ciascun vincolo di STP_k sarà l'intervallo dei valori che vengono mappati dalla funzione di preferenza del vincolo corrispondente nell'STPP in valori di preferenza maggiori o uguali a k. La semi-convessità della funzione garantisce che l'insieme di questi valori costituisce effettivamente un intervallo. Supponiamo che *opt* sia il livello di taglio massimo a cui corrisponda un STP consistente. La dimostrazione del risulatato sulla complessità consiste proprio nel mostrare che le soluzioni ottime dell'STPP semi-convesso sono tutte e sole le soluzioni dell'STP_{opt}.

Teorema 23 *Si consideri un qualunque STPP con funzioni di preferenza semi-convesse a valori in un insieme totalmente ordinato di un semianello con operatore moltiplicativo idempotente. Sia opt il livello massimo di taglio a cui corrisponda un STP consistente. Allora l'insieme delle soluzioni di STP_{opt} corrisponde all'insieme delle soluzioni ottime dell'STPP.*

Da questo risultato si evince che la ricerca di una soluzione ottima di un STPP può essere effettuata in due fasi: una prima fase consiste nella ricerca di *opt*, mentre la seconda fase consiste nel trovare una soluzione di STP_{opt}. In [8] è dimostrato che la seconda fase si può fare in tempo polinomiale. Vi sono alcuni casi in cui anche la prima può richiedere un tempo polinomiale. Ad esempio si può implementare utilizzando una ricerca binaria sui livelli di preferenza. In tal caso se:

- il semianello ha un numero finito di elementi, che sia al più esponenziale nel numero delle variabili, allora la ricerca richiede un numero polinomiale di passi.
- se l'insieme del semianello ha un'infinità numerabile di elementi e le funzioni di preferenza non tendono mai all'infinito, sia l il massimo livello di preferenza associato da una funzione. Se il numero dei valori dell'insieme minori di l è al più esponenziale nel numero di variabili, la ricerca comporterà un numero di passi polinomiale nel numero di variabili.

2.4 Apprendimento automatico

Il campo dell'apprendimento automatico [27] riguarda lo studio di programmi informatici che possano automaticamente migliorare tramite l'esperienza.

Sin da quando furono inventati i computer, ci si è posti la domanda se essi sarebbero un giorno stati in grado di apprendere. L'esistenza di problemi in cui un approccio con programmazione classica si era rivelato inutile ha dato impulso allo studio di questa tecnica alternativa.

Ad oggi la realizzazione di programmi capaci di simulare il processo cognitivo hanno permesso ai computer di svolgere attività come imparare dalle cartelle cliniche quali trattamenti sono più efficaci per nuove malattie, imparare come ottimizzare il consumo di energia in abitazioni o industrie, conoscere i gusti dei propri utenti mentre questi si evolvono e proporre argomenti di particolare interesse. L'apprendimento automatico ha aperto nuovi orizzonti per quanto riguarda l'uso dei computer, introducendo nuovi livelli di competenze e usi. Inoltre la comprensione approfondita del processo di elaborazione dell'informazione da parte dei computer può portare a nuove conoscenze di come questo processo avviene nella mente umana.

Certo non vi sono ancora macchine in grado di apprendere in modo complesso come l'uomo. Sono stati inventati, tuttavia, molti algoritmi che sono efficaci in certi tipi di apprendimento ed inoltre sta emergendo una vera e propria teoria che descrive tali meccanismi. Molti algoritmi di apprendimento automatico si sono rivelati utili in pratica e sono alla base di applicazioni commerciali di successo. Ad esempio nel campo del *riconoscimento vocale* questo tipo di approccio ha dato risultati migliori di tutte le altre tecniche con cui è stato affrontato il problema. Nel campo del *data mining* algoritmi di questo tipo sono utilizzati per scoprire informazioni utili da grandi database contenenti registri di manutenzione del macchinario, affitti, transazioni finanziarie, cartelle cliniche, etc.

L'apprendimento automatico da parte di un computer è così definito in [27]:

Definizione 21 *Si dice che un computer* ***apprende*** *da una esperienza E rispetto ad una classe di compiti T e misura di performance P, se la sua performance su compiti in T, misurata da P, migliora attraverso l'esperienza E.*

Per avere, dunque, un problema di apprendimento ben definito devono essere identificati i seguenti tre oggetti:

- la classe dei compiti;
- la misura di performance;
- la fonte di esperienza;

2.4.1 Apprendimento automatico induttivo

In questa trattazione verrà utilizzato l'apprendimento induttivo [33], in quanto è il più efficace per risolvere il tipo di problemi che affronteremo. L'*apprendimento induttivo* può essere definito come l'abilità di indurre la corretta struttura di

una funzione d che è nota solo rispetto a particolari input. Più precisamente, se un esempio viene definito come una coppia $(s, d(s))$, l'obiettivo computazionale sarà il seguente: *dato un insieme di esempi di d, i.e. il training set Tr, si restituisca una funzione h che approssima il più fedelmente possibile d.* La funzione h viene chiamata *ipotesi.*

Un approccio comune per effettuare l'apprendimento induttivo, specialmente nel campo delle reti neurali, è di valutare la qualità di un'ipotesi h (sul training set) attraverso una *funzione errore.* Una funzione errore molto comune e che può essere utilizzata sui reali è:

$$E = \frac{1}{2} \sum_{s \in Tr} (d(s) - h(s))^2. \tag{2.1}$$

Se invece il dominio di input è un insieme discreto totalmente o parzialmente ordinato, deve essere costruita una nozione di metrica ad hoc e di modo che la distanza tra le coppie di elementi (i.e., d(s) e h(s)) possa essere definita e si possa costruire una formula per la funzione errore.

Data una ipotesi iniziale h_0, lo scopo dell'apprendimento è quello di minimizzare la funzione errore E modificando h_0. Ci sono molti approcci che possono essere seguiti per minimizzare E. Una tecnica molto semplice e comunemente usata consiste nell'usare una definizione di h che dipenda da un insieme di parametri interni W, cioè $h \equiv h_W$, e poi nell'aggiustare questi parametri. La modifica stessa dei parametri può essere formulata in modi diversi, a seconda che il dominio sia o meno isomorfo a $\Re$.

La tecnica che viene usata più spesso, qualora il dominio sia in effetti incluso in $\Re$ e h_W sia continua e derivabile è di seguire la direzione opposta a quella indicata dal *gradiente* di E rispetto a W. Questa tecnica, nota come **discesa di gradiente** [27], e prevede un' aggiustamento dei parametri attraverso il calcolo del gradiente della funzione errore E rispetto ai parametri stessi. In particolare l'insieme dei parametri W viene inizializzato a piccoli valori random al tempo $\tau = 0$ e viene aggiornato al tempo $\tau + 1$ secondo la seguente formula:

$$W(\tau + 1) = W(\tau) + \Delta W(\tau) \tag{2.2}$$

dove

$$\Delta W(\tau) = -\eta \frac{\partial E}{\partial W(\tau)} \tag{2.3}$$

e η è il passo usato nella discesa di gradiente. Se invece non è possibile usare la discesa di gradiente, allora si può ricorrere a qualche forma di ricerca locale, in modo tale da passare dai valori correnti a nuovi valori che erano adiacenti nel dominio rendendo la funzione errore più piccola. L'apprendimento si ferma quando si raggiunge un minimo di E. Si osservi che in generale non vi è

alcuna garanzia che il minimo trovato sia globale. La tecnica che useremo noi sarà un particolare tipo di discesa di gradiente detta *discesa di gradiente stocastica* o incrementale [27]. Infatti la tecnica generale di discesa di gradiente rappresenta una strategia di ricerca in spazi di ipotesi molto vasti o infiniti che può essere applicata ogni qual volta:

1. lo spazio delle ipotesi contiene ipotesi parametrizzate con continuità;
2. l'errore può essere differenziato rispetto ai parametri dell'ipotesi.

Tuttavia, l'applicazione di tale tecnica può comportare le seguenti difficoltà:

1. la convergenza ad un minimo può essere talvolta molto lenta, richiedendo molte migliaia di passi di discesa;
2. non c'è nessuna garanzia di convergere ad un minimo globale.

Se esplicitiamo nella formula 2.2 la funzione errore otteniamo:

$$\Delta W(\tau) = -\eta \frac{\partial}{\partial W(\tau)} \frac{1}{2} \sum_{s \in Tr} (d(s) - h(s))^2. \tag{2.4}$$

Nell'approssimazione stocastica di gradiente l'aggiornamento viene calcolato dopo che un singolo esempio del training set è stato esaminato. In altre parole la $\Delta - rule$ [4] diventa:

$$\Delta W(\tau) = -\eta \frac{\partial}{\partial W(\tau)} \frac{1}{2} (d(s) - h(s))^2. \tag{2.5}$$

dove come si può osservare è sparito il simbolo di sommatoria. Un modo di interpretare la discesa di gradiente stocastica è proprio quello di considerare una funzione errore distinta E_s definita per ogni singolo esempio del training set:

$$E_s = \frac{1}{2}(d(s) - h(s))^2 \tag{2.6}$$

dove d(s) è il valore target dell'esempio, cioè il valore che è associato a quell'esempio nel training set, e $h(s)$ è il valore dato dall'ipotesi per quell'esempio. Il processo di discesa stocastica fa una scansione di tutti gli esempi $s \in Tr$, e ad ogni iterazione modifica i parametri secondo il gradiente calcolato su E_s. La sequenza di questi aggiornamenti dei parametri, quando viene ripetuta su tutti gli esempi del training set, fornisce una approssimazione ragionevole della discesa di gradiente eseguita rispetto alla funzione errore originale E. Rendendo il valore η, cioè il passo di discesa, abbastanza piccolo,

[4] Con $\Delta - rule$ viene indicata la formula 2.2.

la discesa di gradiente stocastica può approssimare bene a piacere la discesa originaria.

Le principali differenze tra la discesa di gradiente stocastica e quella standard sono le seguenti:

- Nella discesa standard l'errore considerato è cumulativo, cioè viene sommato su tutti gli elementi del training set prima di effettuare gli aggiornamenti. Nella discesa di gradiente stocastica invece i parametri sono aggiornati dopo l'esame di ciascun esempio.

- Il fatto di sommare l'errore su tutti gli esempi, come accade nel caso della discesa standard, rende ogni passo di aggiornamento computazionalmente più pesante. D'altra parte, visto che viene utilizzato il gradiente effettivo, la discesa di gradiente standard può essere utilizzata con un passo di discesa maggiore.

- Nei casi in cui ci siano molti minimi locali di E, la discesa di gradiente stocastica può in qualche caso evitare di cadere in uno di questi minimi in quanto usa i vari ∇E_s invece di ∇E per guidare la ricerca.

Entrambi i tipi di discesa vengono utilizzati in pratica. Dopo un certo numero di iterazioni un criterio di stop pone fine all'apprendimento. Tale criterio consiste di solito in una soglia sull'errore. Quando l'errore scende al di sotto di tale soglia l'allenamento del modulo di apprendimento termina e viene effettuata una verifica sottoponendo al modulo un insieme di esempi nuovi. Questo insieme viene detto *test set* e l'errore medio calcolato su di esso dà una misura di quanto sia stato effettivamente appreso.

2.4.2 Apprendimento automatico di CSPs

Illustriamo ora il lavoro che è il precursore più diretto della parte di questa ricerca sull'apprendimento automatico. Si tratta di un modello di apprendimento di preferenze per soluzioni di problemi di vincoli [31],[3]. in questo caso l'apprendimento è stato utilizzato per trovare un insieme di valori adatti ad essere associati a tuple di vincoli di un dato CSP. In tal modo si ottiene un SCSP dove il valore associato a ciascuna soluzione è consistente con la preferenza assegnata a quella soluzione.

Si consideri un CSP P. Come abbiamo già osservato nella Sezione (2.2), esso può essere interpretato come un problema di vincoli con preferenze, SCSP, basato sul c-semianello: $\langle \{true, false\}, \vee, \wedge, false, true\} \rangle$. In altre parole alle tuple ammissibili viene associato il valore $true$ e a quelle non ammesse il valore $false$, e le soluzioni di P saranno le n-uple t che sono ammesse cioè tali che $val_P(t) = true$. Supponiamo ora che a qualche soluzione di

P venga associato un certo valore di preferenza $d(t)$ che dica quanto buona è ritenuta tale soluzione. Supponiamo che il numero di soluzioni a cui viene associata una preferenza sia m. Allora possiamo costituire un training set $Tr = \{(t_1, d(t_1)), \cdots, (t_m, d(t_m))\}$. Osserviamo come la funzione $d : \{n - uple\} \longrightarrow D$ dove D è l'insieme delle preferenze associate alle soluzioni è definita solo parzialmente tramite le immagini solo di alcuni elementi. Tuttavia per poter effettuare l'apprendimento dobbiamo definire un modo di combinare e confrontare i valori di preferenza, quindi, oltre agli esempi serviranno anche i due seguenti oggetti:

- un semianello il cui insieme contenga i valori di preferenza associati alle soluzioni. Cioè un semianello $\langle A, +, \times, 0, 1\rangle$ tale che $D \subset A$.
- una funzione di distanza su un tale semianello. Una tale funzione sarà rappresentata così $dist : A^2 \longrightarrow \Re$, e darà la distanza tra due elementi del semianello. Ovviamente dovrà soddisfare le solite proprietà delle distanze [31].

Avendo dunque a disposizione un CSP, un training set e i due oggetti menzionati sopra, si vuole definire un SCSP P' corrispondente sul semianello dato tale che:

1. P e P' sono rappresentabili dalla stessa rete di vincoli, cioè hanno le stesse variabili e vincoli che si corrispondono hanno lo stesso tipo;
2. per ogni n-upla t tale che $(t, d(t))$ è un esempio $dist(val_{P'}(t), d(t)) < \epsilon$ con $\epsilon > 0$ e piccolo.

Se, come di solito accade, l'insieme del semianello è l'insieme dei reali allora la distanza viene definita nel modo usuale $dist(val_{P'}(t), d(t)) = |val_{P'}(t) - d(t)|$. Inoltre se si vuole usare la tecnica di discesa di gradiente l'operatore moltiplicativo del semianello $\times$ deve essere continuo e derivabile. Il fatto di volere che P e P' abbiano la stessa rete di vincoli lascia come unici parametri liberi modificabili i valori da associare alle tuple di un vincolo. Per ogni vincolo $c_i = \langle def_i, con_i\rangle$ in P, si consideri $S_i = \{t_{ij}$ tali che $def_i(t_{ij}) = true\}$. L'idea è quella di associare un peso $w_{ij} \in A$, ad ogni $s_{ij} \in S_i$. Alle altre tuple, quelle non ammesse associamo la costante **0** che è il "peggior" elemento del semianello. Associando in questo modo i pesi alle tuple, il valore assegnato ad ogni n-upla in P' sarà:

$$val_{P'} = \prod_{i=1}^{k} (\sum_{j=1}^{|S_i|} subtuple(t_{ij}, t, i) \times w_{ij}), \tag{2.7}$$

dove k è il numero di vincoli e:

$$subtuple(t_{ij}, t, i) = 1 \text{ se } t \downarrow^{V}_{con_i} = t_{ij},\ 0 \text{ altrimenti.}$$

Si osservi inoltre che per ogni i esiste ed è unico j tale che $subtuple(t_{ij}, t, i) = 1$. Sia l_i quel j. Allora:

$$\sum_{j=1}^{|S_i|} subtuple(t_{ij}, t, i) \times w_{ij} = w_{il_i}$$

e dunque

$$val_{P'}(t) = w_{1l_1} \times \cdots w_{k,l_k}.$$

La discesa di gradiente sulla funzione errore verrà eseguita sui pesi w_{ij} utilizzando la funzione distanza sul semianello. Nel caso di un semianello generico la funzione errore sarà del tipo:

$$f_{error} : \Re^m \longrightarrow \Re$$

dove m è il numero di elementi del training set. Questa funzione prende m distanze tra gli esempi e le n-uple correnti e calcola l'errore totale dato d tali distanze. Ovviamente si dovrà controllare che ad ogni aggiornamento i valori dei parametri liberi si mantengano nel semianello.

2.4.3 Approssimazione del *min*

In questa trattazione verrà proposto un modulo di apprendimento automatico che applica la tecninca di discesa di gradiente ad una funzione composta da varie altre tra cui *min*. Come è noto *min* è una funzione che ha una discontinuità di prima specie e questo costituisce un problema per il calcolo di funzioni composte che la contengano.

Vogliamo qui introdurre una approssimazione di classe C^1, cioè continua e derivabile con derivata continua della funzione *min* [15]. Introduciamo prima una approssimazione del caso in cui vi siano solo 2 argomenti.

La funzione che mappa una coppia di reali in quello minore dei due è così rappresentabile:

$$min : \Re^2 \longrightarrow \Re$$

$$(x, y) \mapsto \begin{cases} x & \text{se } x \leq y \\ y & \text{se } y < x \end{cases}$$

La funzione *min* con due argomenti ha un'unica discontinuità per $x = y$. Questo significa che non è differenziabile. Consideriamone le derivate parziali:

$$\frac{\partial min}{\partial x} = \begin{cases} 1 & \text{se } x \leq y \\ 0 & \text{se } y < x \\ \text{non definita} & \text{se } x = y \end{cases}$$

$$\frac{\partial min}{\partial y} = \begin{cases} 0 & \text{se } x \leq y \\ 1 & \text{se } y < x \\ \text{non definita} & \text{se } x = y \end{cases}$$

Le derivate parziali della funzione *min* sono delle *funzioni a scalino* e la discontinuità di *min* corrisponde al salto delle derivate parziali da 0 a 1. Indicheremo l'approssimazione di classe C^1 con min_2^β dove l'indice corrisponde al numero di argomenti e β è un parametro di cui verrà chiarito il significato ora. L'idea di fondo è di approssimare le derivate parziali con una *sigmoide* e di integrare quello che si è ottenuto. La formula della sigmoide è:

$$\sigma(x) = \frac{1}{1 + e^{-\beta x}}.$$

Il parametro β regola quanto è buona l'approssimazione cioè, tanto esso è più alto tanto più la sigmoide si schiaccia contro la funzione a scalino. Per $\beta > 0$ la sigmoide tende a 1 per $x \to +\infty$ e tende a 0 per $x \to -\infty$. Per valori del parametro β piccoli essa va da 0 a 1 in modo continuo per x che va da $-\infty$ a $+\infty$; per valori di β elevati invece compie un innalzamento improvviso, simile ad un salto da 0 a 1 in un range di x molto stretto vicino a 0. Utilizzando la sigmoide per approssimare le derivate parziali di *min* otteniamo:

$$\frac{\partial min_2^\beta}{\partial x} \simeq \frac{1}{1 + e^{-\beta(y-x)}} = \frac{e^{-\beta y}}{e^{-\beta x} + e^{-\beta y}} \tag{2.8}$$

$$\frac{\partial min_2^\beta}{\partial y} \simeq \frac{1}{1 + e^{-\beta(x-y)}} = \frac{e^{-\beta x}}{e^{-\beta x} + e^{-\beta y}} \tag{2.9}$$

Si osservi che:

$$(x - y) \to \infty \Longrightarrow (\frac{\partial min_2^\beta}{\partial x} \to 0 \text{ e } \frac{\partial min_2^\beta}{\partial y} \to 1)$$

Ora basta fare l'integrazione delle derivate parziali:

$$\int \frac{\partial min_2^\beta}{\partial x}\, dx = \int \frac{\partial min_2^\beta}{\partial y}\, dy = -\frac{1}{\beta} ln(\frac{e^{-\beta x} + e^{-\beta y}}{2}) + C.$$

Per determinare la costante C basta osservare che essendo $min(z, z) = z$ è ragionevole imporre che $min_2^\beta(z, z) = z$, da cui si ottiene $C = 0$. In conclusione l'approssimazione di classe C^1 della funzione *min* con due argomenti è:

$$min_2^\beta(x, y) = -\frac{1}{\beta} ln(\frac{e^{-\beta x} + e^{-\beta y}}{2}) \tag{2.10}$$

Il parametro di pendenza β determina l'accuratezza dell'approssimazione, visto che determina il grado in cui la sigmoide approssima la funzione a scalino: più schiacciata è la sigmoide più accurata sarà l'approssimazione. Infatti il valore assunto da min dipende solo dall'argomento minore. In min_2^β questa dipendenza sarà tanto più accentuata quanto più sarà accurata l'approssimazione.

Se invece si considera la funzione min con n argomenti le sue derivate parziali saranno approssimate nel seguente modo:

$$\frac{\partial min_n^\beta(x_1, \cdots, x_n)}{\partial x_j} = \frac{e^{-\beta x_j}}{\sum_{i-1}^n e^{-\beta x_j}}$$

da cui l'approssimazione C^1 di min a n parametri:

$$min_n^\beta(x_1, \cdots, x_n) = -\frac{1}{\beta} ln(\frac{1}{n} \sum_{i=1}^n e^{-\beta x_i})$$

Facciamo un'ultima osservazione su quale potrebbe essere il valore di β ideale. Sempre in [15] è dimostrato che se noi vogliamo una approssimazione tale che $min_n^\beta(0, \cdots, m) = \frac{p}{q} m$ dove m è il minimo valore possibile per un suo argomento allora una buona scelta per β è:

$$\beta \approx \frac{q ln(n)}{m}$$

Capitolo 3

Proprietà e complessità di STPPs

Si è già osservato come, mentre risolvere un STPP generico è un problema NP-hard, imponendo certe condizioni sulla forma delle funzioni si ottiene una sottoclasse trattabile. La proprietà che devono soddisfare tali funzioni è la semi-convessità, cioè ad un qualsiasi taglio orizzontale del grafico della funzione deve corrispondere un unico intervallo di valori con ordinata maggiore o uguale al livello del taglio. Ricordiamo che risolvere un STPP significa trovare una soluzione ottima, cioè il cui livello di preferenza sia il massimo rispetto all'ordinamento indotto dall'operatore additivo sul semianello. In [17] questo è stato dimostrato attraverso l'esistenza di un STP le cui soluzioni sono tutte e sole le soluzioni ottime dell'STPP. Questo STP si ottiene operando vari tagli a vari livelli delle funzioni. A ciascuno di questi tagli corrisponde un STP, quello individuato dagli intervalli corrispondenti ai valori che hanno ordinata maggiore o uguale al livello di taglio. Si noti come qui sia fondamentale la semi-convessità che garantisce il fatto che per ogni vincolo si ottenga un unico intervallo. In pratica si cerca il livello di taglio massimo a cui corrisponda un STP consistente. Questo può essere ottenuto con una ricerca binaria nei valori tra 0 e 1.

Pur essendo questo metodo efficace, esso non mette in evidenza la relazione tra gli STPPs e gli STPs, nel senso che non chiarisce come gli ultimi siano infatti una estensione dei primi e come, viceversa i primi siano un caso particolare degli ultimi. Ci proponiamo dunque di dimostrare come si possa trovare una soluzione ottima di un STPP senza fare backtracking utilizzando un algoritmo di consistenza sui cammini. Vogliamo cioè riottenere il risultato sulla complessità in [17] applicando un algoritmo di consistenza sui cammini, cioè seguendo quanto è stato fatto per gli STP in [8].

Il primo passo è quello di definire in modo preciso un algoritmo di consistenza sui cammini per STPPs. Tra i vari algoritmi abbiamo scelto di

estendere PC-2 [8].

3.1 Algoritmo STPP_PC-2

Sempre in [17] è stata dimostrata la chiusura delle funzioni semi-convesse rispetto ai due operatori sui vincoli, che ricordiamo sono l'intersezione, $\oplus$, e la composizione, $\otimes$. Questa proprietà delle funzioni semi-convesse è però rimasta inutilizzata dagli autori nei risultati dell'articolo. Essa, invece, è fondamentale per l'interpretazione degli STPPs come estensione degli STPs. Ci proponiamo dunque di mostrare come sia possibile applicare l' algoritmo di consistenza sui cammini PC-2 agli STPPs.

Ricordiamo in breve cosa fa PC-2 su problemi di vincoli temporali semplici. Esso considera tutti i triangoli, o meglio i sotto-grafi con tre variabili orientati e completi del grafo delle distanze che rappresenta il problema, e vi applica la seguente operazione di rilassamento:

$$T_{ij} = T_{ij} \oplus \{T_{ik} \otimes T_{kj}\}$$

dove T_{ij} è il vincolo tra la i-esima e la j-esima variabile, T_{ik} è il vincolo tra la i-esima e la k-esima variabile, T_{kj} è il vincolo tra la k-esima e la j-esima variabile. In pratica il vincolo T_{ij} viene aggiornato con il vincolo ottenuto intersecando T_{ij} stesso con la composizione di T_{ik} con T_{kj}. Nel nostro caso è necessario sostituire $\oplus$ e $\otimes$ con le operazioni su STPPs $\oplus_S$ e $\otimes_S$ [17]. Il passo di rilassamento diventa ora:

$$T_{ij} = T_{ij} \oplus_S \{T_{ik} \otimes_S T_{kj}\}.$$

Per capire meglio come funziona supponiamo che il c-semianello sia $S_{FCSP} = \langle \{x \mid x \in [0,1]\}, max, min, 0, 1\rangle$, [4] e vediamo in un esempio come funzionano i due operatori. Supponiamo che il vincolo T_{ik} sia costituito dall'intervallo $I_{ik} = [3,5]$ e che la funzione di preferenza sia $f_{ik}(x) = x/10$, e.g. $f(3) = 0.3$.

Supponiamo che T_{kj} sia costituito dall'intervallo $I_{kj} = [4,6]$ e che la sua funzione di preferenza f_{kj} sia definita come quella di T_{ik}.

Allora il vincolo $T_{ik} \otimes_S T_{kj}$ avrà come intervallo $I = [3+4, 5+6] = [7,11]$ e la sua funzione di preferenza, diciamo f, sarà cos'í definita:

$$f(a) = \sum\{f_{ik}(a_1) \times f_{kj}(a_2) \mid a = a_1 + a_2, a_1 \in I_{ik}, a_2 \in I_{kj}\}.$$

Sfruttiamo ora il fatto che $+ = max$ e $\times = min$ nel caso del c-semianello fuzzy. Otteniamo:

$$f(a) = max\{min(f_{ik}(a_1), f_{kj}(a_2)) \mid a = a_1 + a_2, a_1 \in I_{ik}, a_2 \in I_{kj}\}$$

In pratica per ogni valore dell'intervallo composto devo guardare tutte le sue possibili decomposizioni in un valore dell'intervallo del primo vincolo più un valore dell'intervallo del secondo vincolo. Per ciascuna decomposizione si prende il minimo tra i due valori di preferenza, poi si prende il massimo tra questi considerando tutte le possibili decomposizioni.

Ad esempio, il valore 8 può essere ottenuto facendo 4+4. Ora, $f_{ik}(4) = 0.4 = f_{kj}$. Inoltre, banalmente, $min(0.4, 0.4) = 0.4$ quindi alla decomposizione $a_1 = 4$, $a_2 = 4$ verrà associato il valore 0.4. Tuttavia 8 può essere scritto anche come 5+3 e $f_{ik}(5) = 0.5$ mentre $f_{kj}(3) = 0.3$. Dunque essendo $min(0.5, 0.3) = 0.3$ alla decomposizione $a_1 = 5$, $a_2 = 3$ verrà associato il valore 0.3. Infine $f(8)$ sarà il massimo tra i valori associati alle decomposizioni, cioè, si avrà $f(8) = 0.4$.

Illustriamo ora l'intersezione. Supponiamo di voler intersecare il vincolo composto ottenuto nell' esempio precedente con il vincolo T_{ij} avente intervallo $I_{ij} = [6, 10]$ e funzione di preferenza $f_{ij}(x) = 0.8\ \forall x$. Il vincolo $T_{ij} \oplus_S \{T_{ik} \otimes_S T_{kj}\}$ avrà come intervallo I' l'intersezione $I \cap I_{ij} = [7, 10]$ e a ciascun elemento d di questo intervallo verrà associata come funzione di preferenza $f'(d) = (f_{ij}(d) \times f(d))$. Ricordando che il semianello è S_{FCSP} si ottiene: $f'(d) = min(f_{ij}(d), f(d))$. Se prendiamo $d = 8$, $f'(8) = min(f_{ij}(8), f(8)) = min(0.4, 0.8) = 0.4$. Se vogliamo ora completare l'operazione di rilassamento sul triangolo delle variabili i, j, k, basta assegnare a T_{ij} il vincolo ottenuto intersecandolo con la composizione di T_{ik} e T_{kj}.

Avendo chiarito il passo fondamentale di rilassamento, riflettiamo ora su come funziona PC-2 globalmente. Prima viene creata una coda di "Paths", cioè una coda i cui elementi sono i triangoli di vincoli su cui si faranno i rilassamenti, ovviamente seguendo la politica FIFO della coda. Ad esempio un elemento della coda potrebbe essere così rappresentato $(i, k, j) = (2, 4, 3)$. Esso rappresenta il triangolo di vincoli nella figura 3.1:

cioè il rilassamento che gli corrisponde sarà:

$$T_{23} = T_{23} \oplus_S \{T_{24} \otimes_S T_{43}\}$$

questo supponendo, come sarà sempre per noi, che le variabili vengano indicate, o meglio identificate da numeri interi maggiori o uguali a 0. Ovviamente, affinchè si tratti di grafi a tre nodi non degeneri, dovrà essere $k \neq i, j$ e $i \leq j$. L'algoritmo non termina fino a quando la coda non si svuota. Fino ad allora, l'algoritmo ad ogni passo estrae un elemento (i, k, j) della coda, e controlla se l'eventuale rilassamento apporterà modifiche o meno. Il controllo verrà indicato:

$$T_{ij} \neq_S T_{ij} \oplus_S (Tik \otimes_S T_{kj}).$$

Si noti come la S sia stata apposta anche al segno di uguaglianza e non

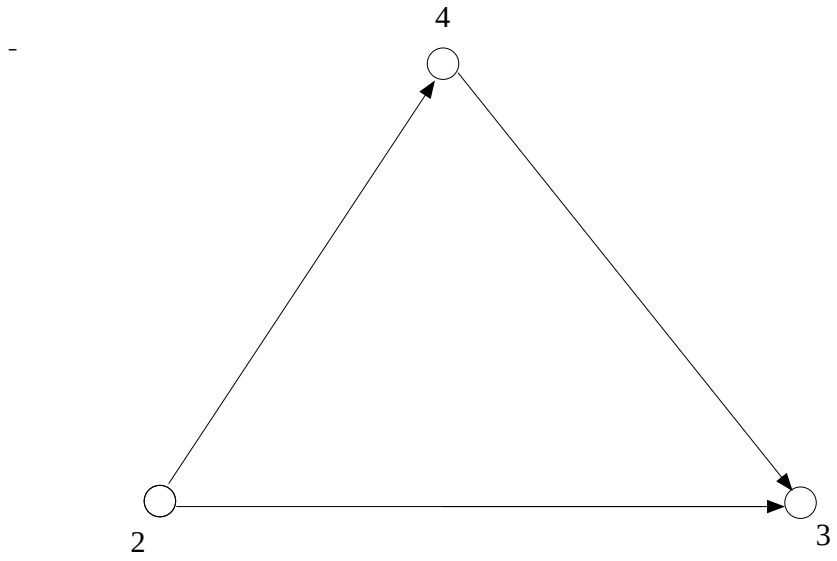

Figura 3.1: Grafo che rappresenta il triangolo di vincoli corrispondente all'elemento della coda (2,3,4).

solo all'operatore di composizione di vincoli. Infatti è necessario definire cosa significa che due vincoli temporali con preferenze sono uguali.

Definizione 22 *Dati due vincoli temporali semplici con preferenze,*$T_1 = \langle I_1, f_1 \rangle$ *e* $T_2 = \langle I_2, f_2 \rangle$*, essi sono uguali se* $I_1 = I_2$ *e* $f_{1_{|I_1}} = f_{2_{|I_2}}$.

Non basta dunque vedere la definizione della funzione dei due vincoli, nel caso ve ne sia una, per dire se sono uguali o meno, ma è necessario guardare la restrizione delle due funzioni all'intervallo in questione. Possiamo duque concludere che T_{ij} verrà cambiato solo se esso non coincide esattamente con la sua intersezione con la composizione dei due vincoli. Si noti come non sia sufficiente la condizione:

$$T_{ij} \neq_S T_{ik} \otimes_S T_{kj}.$$

Ad esempio se T_{ij} ha come intervallo $[2, 3]$ con funzione di preferenza la funzione costante che vale 0.2 e $T_{ik} \otimes_S T_{kj}$ ha come intervallo $[1, 4]$ con preferenza 0.1 per 1 e 4 e 0.2 altrove, chiaramente la disuguaglianza sopra è soddisfatta ma se intersechiamo i due vincoli otteniamo un vincolo esattamente uguale a T_{ij}, e dunque un tale aggiornamento di T_{ij} non porterebbe nessuna modifica. Questo controllo è importante in quanto decide se aggiungere o meno alla coda gli elementi corrispondenti ai triangoli contenenti T_{ij}.

Se di fatto viene apportato qualche cambiamento a T_{ij} con il rilassamento, se cioè viene ristretto l'intervallo o se cambia anche un solo valore di preferenza o entrambe, gli effetti del cambiamento devono essere propagati a tutti i vincoli e questo viene fatto appunto aggiornando la coda come detto. Le

condizioni di terminazione di questo algoritmo sono state descritte nel capitolo 1, paragrafo 1.2. Se supponiamo che il semianello sia S_{FCSP}, si ha che l'operatore moltiplicativo, *min*, è idempotente e si ha la chiusura di *max* e *min* rispetto a tutti gli insiemi finiti che contengono un sottoinsieme di $[0, 1]$, possiamo affermare che PC-2 termina, la strategia è irrilevante e il problema ottenuto è equivalente a quello dato [4]. Si osservi che due STPPs sono equivalenti se hanno lo stesso insieme di soluzioni e alle soluzioni sono associati gli stessi valori di preferenza. Ad esempio, due STPPs, con le stesse variabili che ammettono la stessa soluzione ma che le associano un valore di preferenza diverso non sono equivalenti.

Algoritmo STPP_ PC-2

1. $Q \longleftarrow \{(i,j,j) | (i < j) \text{ e } (k \neq i,j)\}$
2. **if** $Q \neq$ **do**
3. estrarre un elemento (i,k,j) da Q
4. **if** $T_{ij} \neq_S T_{ij} \oplus_S (Tik \otimes_S T_{kj})$ **then**
5. $T_{ij} \longleftarrow T_{ij} \oplus_S (Tik \otimes_S T_{kj})$
6. **if** $T_{ij} =$ **then** exit (inconsistenza)
7. $Q \longleftarrow Q \cup RELATED\_PATHS((i,k,j))$
8. **end_if**
9. **end_ while**

Figura 3.2: Algoritmo STPP_ PC-2.

Guardando l'algoritmo si vede come esso termini solo nel caso avvengono due circostanze: o un vincolo diventa inconsistente, il suo intervallo diventa vuoto, oppure viene raggiunta la stabilità, ovvero la coda si svuota. La funzione RELATED_PATHS((i,k,j)) aggiorna la coda aggiungendo i triangoli contenenti il vincolo T_{ij}.

Avendo esposto come funziona PC-2 su un STPP possiamo ora trarre delle conclusioni su quali modifiche questo algoritmo possa portare ad un vincolo temporale semplice con preferenze [1].

- Durante l'esecuzione di PC_2 l'intervallo di un vincolo può solo stringersi e mai allargarsi. Questo segue essenzialmente quello che accade per gli STP senza preferenze (talvolta detti vincoli *hard*).
- Se si analizza l'operazione di rilassamento si vede che per ogni valore che rimane nell'intervallo come preferenza si prende il minimo tra quella che aveva originariamente e quella nel vincolo ottenuto componendo. Questo è dovuto, come abbiamo già osservato nell'esempio sopra, al fatto che all'intersezione di vincoli $soft$ corrisponde a livello delle funzione l'operatore moltiplicativo, che è min. Possiamo dunque concludere che dopo STPP_PC-2 i livelli di preferenza dei vincoli saranno in generale più bassi, cioè le funzioni vengono schiacciate.

[1] Per semplicità ci riferiremo al semianello S_{FCSP} ma quanto detto vale in generale per un qualunque semianello con operatore moltiplicativo idempotente.

Il secondo punto messo in evidenza fa notare che la stabilità nel nostro caso, verrà raggiunta quando tutte le funzioni saranno scese tutte allo stesso livello. Questo livello sarà $\leq M$ dove M è il minimo tra i massimi delle funzioni. È dunque possibile enunciare il seguente teorema.

Teorema 24 *Dato un STPP consistente sui cammini con funzioni di preferenza orizzontali, tutte le sue funzioni di preferenza hanno lo stesso massimo (valore ottimo di preferenza).*

Dim. Per assurdo, si supponga che esista un vincolo $T = \langle I, f \rangle$ che abbia il valore ottimo di preferenza più alto di quello degli altri vincoli. Si considerino, ora, due degli altri vincoli $T_1 = \langle I_1, f_1 \rangle$ e $T_2 = \langle I_2, f_2 \rangle$ i quali formano un triangolo con T. Si consideri anche un istante $a \in I$ tale che $f(a)$ sia massimo.

Poiché il problema è consistente sui cammini , si ha che :

$$f(a) = sup(inf(f_1(a_1), f_2(a_2)))$$

dove *sup* è esteso a tutte le a_1, a_2 tali che $a_1 + a_2 = a$.

Rimpiazzando ciascuna delle $f_i(a_i))$ con il massimo delle funzioni f_i, $max(f_i)$, per la monotonia di sup e inf, si ha che:

$$f(a) = sup(inf(f_1(a_1), (f_2(a_2)))) \leq sup(inf(max(f_1), max(f_2)))$$

$$= inf(max(f_1), max(f_2)).$$

Ora, per definizione di inf, si ha che:

$$inf(max(f_1), max(f_2)) \leq max(f_1)$$

e

$$inf(max(f_1), max(f_2)) \leq max(f_2).$$

Dunque dalle precedenti equazioni si ha $f(a) \leq max(f_1)$ e $f(a) \leq max(f_2)$.

Questo è assurdo poichè $f(a)$ per ipotesi era maggiore di tutti gli altri massimi. **c.v.d.**

3.2 Una classe trattabile di STPPs

Ci proponiamo ora di mostrare come sia possibile ottenere una soluzione si un STPP, utilizzando STPP_PC-2, con una complessità polinomiale e senza fare backtracking. Il capitolo successivo descriverà come un risolutore basato su questi risultati sia stato effettivamente implementato. Ricordiamo brevemente che una soluzione di un STPP è un assegnamento a tutte le variabili

consistente con tutti i vincoli (in senso hard). Quello che ha in più è un valore di preferenza che viene computato dai valori di preferenza locali.

Facciamo un esempio. Supponiamo di avere un STPP con tre variabili, identificate da v_0, v_1, v_2, con tre vincoli T_{01}, T_{02}, T_{12}. Supponiamo che l'assegnamento $v_0 = 0$, $v_1 = 3$, $v_2 = 1$ soddisfi tutti i vincoli del problema. Questo assegnamento identifica tre distanze temporali, ovvero un elemento per ciascuno dei tre intervalli dei vincoli del problema: esse sono 3 per il vincolo T_{01}, 1 per il vincolo T_{02}, e -2 per il vincolo T_{12}. Supponiamo che i valori di preferenza associati a queste distanze siano rispettivamente: $f_{01}(3) = 0.5$, $f_{02}(1) = 0.7$, e $f_{12}(-2) = 0.1$. A questa soluzione, s, verrà associato il valore di preferenza $t(s) = min(0.5, 0.7, 0.1) = 0.1$, in quanto ricordiamo che per combinare i valori di preferenza si utilizza l'operatore moltiplicativo del semianello, nel nostro caso min.

Dal teorema precedente e da quanto detto sul meccanismo di assegnamento delle preferenze alle soluzioni si può dedurre che tutte le soluzioni ottime di un STPP hanno un valore di preferenza, che è il massimo delle funzioni dopo l'applicazione di STPP_PC-2.

Teorema 25 *Si consideri un STPP P con funzioni semi-covesse definito su un semianello con operatore moltiplicativo idempotente, ed l'STPP P' ottenuto da P dopo aver applicato STPP_PC-2. Si consideri inoltre l'STP P" ottenuto da P' prendendo solo i sottointervalli con valore di preferenza ottimo. Allora l'insieme delle soluzioni ottime di P coincide con l'insieme delle soluzioni di P".*

Dim. Denotiamo con $Opt(P)$ l'insieme delle soluzioni ottime di P, e con $Sol(P'')$ l'insieme delle soluzioni di P''. Si vuole dimostrare che $Opt(P) = Sol(P'')$.

Prima mostriamo $Opt(P) \subset Sol(P'')$. Assumiamo per assurdo che esista una soluzione ottima $s \in Opt(P)$ che non è soluzione di P''. Dato che non è una soluzione per P'', esiste un vincolo, diciamolo $T_{ij} = \langle I_{ij}, f_{ij} \rangle$ violato dalla soluzione. Questo significa che i valori v_i e v_j che vengono assegnati in s rispettivamente a x_i e x_j sono tali che $(v_j - v_i) \notin I_{ij}$. Per come è definito P'' questo significa che $v_j - v_i$ non appartiene al sottointervallo dei valori ottimi si T_{ij}. Quindi $f_{ij}(v_j - v_i)$ non è ottimo e una qualunque soluzione che contenga questo assegnamento avrà preferenza $\leq f_{ij}(v_j - v_i)$ e dunque non può essere ottima.

Proviamo ora che $Opt(P) \supset Sol(P'')$. Si prenda una qualunque soluzione s di P''. Essa è automaticamente anche una soluzione di P. Poiché essa soddisfa i vincoli assegnando dei valori alle variabili che identificano delle distanze tutte aventi il medesimo valore di preferenza, quello ottimo, essa

non può che avere un livello di preferenza ottimo. Dunque $s \in Opt(P)$. **c.v.d.**

Nel teorema che segue mostriamo come utilizzando le cose dette è possibile trovare una soluzione ottima di un STPP in tempo polinomiale e senza fare backtracking.

Teorema 26 *Trovare una soluzione ottima di un STPP consistente sui cammini con*

- *funzioni di preferenza semi-convesse*
- *un semianello avente un operatore moltiplicativo idempotente*

può essere fatto in tempo polinomiale.

Dim. Partendo da un STPP P con funzioni di preferenza semi-convesse e definito su un semianello munito di un operatore moltiplicativo idempotente, si segua questa sequenza:

1. Si costruisca l'STPP P' ottenuto da P applicando STPP_PC-2. Sappiamo che questa procedura richiede tempo polinomiale e che, essendo P' equivalente a P [2], i due problemi hanno lo stesso insieme di soluzioni ottime.

2. Si costruisca l'STP P'' ottenuto da P' prendendo solo gli intervalli i cui elementi hanno valore di preferenza ottimo. Per il 24, $Opt(P)(= Opt(P')) = Sol(P'')$.

3. Ora basta trovare una soluzione di P''. Questo può essere fatto in tempo polinomiale se l'STP è consistente [8] . La condizione di consistenza è sempre soddisfatta nel nostro caso. Infatti per un STP essere consistente significa avere almeno una soluzione. Anche se P (e quindi anche P') avesse tutte le soluzioni al peggior livello possibile di preferenza, i.e. 0, P'' avrebbe comunque soluzioni, ovviamente con livello di preferenza 0. Se invece P non ha soluzioni non si arriva nemmeno a questo punto della sequenza ma STPP_PC-2 avrebbe individuato l'inconsistenza e avrebbe fatto abortire la procedura.

Poiché ciascuno di questi passi richiede tempo polinomiale, e dopo aver eseguito la sequenza viene in effetti trovata una soluzione ottima di P, possiamo concludere che l'obbiettivo proposto può essere raggiunto in tempo polinomiale. **c.v.d.**

Ricordiamo che le ipotesi fondamentali che devono valere affinchè sia possibile applicare i risultati esposti sono:

[2] grazie all'idempotenza dell'operatore moltiplicativo.

- i vincoli temporali devono essere semplici, cioè possedere un unico intervallo;
- le funzioni di preferenza devono essere semi-convesse;
- l'operatore moltiplicativo deve essere idempotente.

Le due Sezioni seguenti sono dedicate ad esaminare il caso in cui vengano rilassate rispettivamente la prima e la seconda ipotesi.

Per quanto riguarda la terza, il suo ruolo fondamentale nel garantire l'applicabilità degli algoritmi di consistenza locale rende più difficile escluderla, qualche accenno su questo si trova nell'ultimo capitolo.

3.3 TCSPPs semi-convessi

Vogliamo ora mostrare come dai nostri risultati si possa ottenere un metodo per risolvere problemi di vincoli temporali non semplici semi-convessi. Tali problemi sono caratterizzati da vincoli temporali che hanno più di un intervallo e funzioni semi-convesse su ciascuno di questi intervalli. Vogliamo seguire un metodo analogo a quello proposto in cite [8] per i vincoli temporali. Si tratta di costruire tanti STPPs prendendo, in tutti i modi possibili, un intervallo per ogni vincolo. Essendo le funzioni semi-convesse per ipotesi, possiamo applicare i risultati ottenuti a ciascuno di essi.

Teorema 27 *Si consideri un TCSPP P con funzioni di preferenza semi-convesse e gli STPPs, $P_1, \cdots, P_n$ ottenuti da P prendendo un intervallo per ciascun vincolo. Allora:*

$$Opt(P) = best(\bigcup_i (sol(P_i))).$$

Dim. Si consideri ora il secondo punto: $opt(P') = max(\bigcup_i (sol(P_i)))$. Sia s una soluzione di P'. Tale soluzione individua delle distanze su ogni vincolo, e dunque sarà una soluzione di qualche STP tra $P_1, \cdots, P_n$, quello che ha i sottointervalli contenenti quelle distanze, diciamolo P_i. Inoltre, il valore di preferenza associato ad una soluzione è lo stesso in P ed in P_i perchè le funzioni di preferenza sono le stesse. Dunque, se s è ottima in P', lo sarà anche in P_i. In tal caso il valore di preferenza associato a s sarà maggiore o uguale a quelli delle altre soluzioni nei P_i, e dunque $s \in best(\bigcup_i (sol(P_i)))$.

Si consideri ora una soluzione $s \in best(\bigcup_i (sol(P_i)))$. Essa avrà valore di prefernza associato maggiore o uguale a quello di tutte le altre soluzioni dei P_i. Poiché, nè i valori di preferenza, nè l'insieme di soluzioni viene cambiato dalla costruzione dei P_i, s è ottima anche per P. **c.v.d.**.

Quindi un metodo per risolvere TCSPPs con funzioni semi-convesse è quello di costruire tutti i possibili STPPs prendendo un intervallo per ogni vincolo. Poi, applicando STPP_ PC-2, è possibile trovare il livello ottimo di ciascun STPP. A questo punto le soluzioni ottime del TCSPP saranno le soluzioni ottime degli STPPs con livello ottimo di preferenza $best(\bigcup_i(sol(P_i)))$

3.4 Funzioni di preferenza non semi-convesse

In [17] è stato mostrato come avere delle funzioni non semi-convesse porta, una volta che vengano considerati problemi temporali aventi vincoli che corrispondono agli insiemi di valori con preferenza maggiore o uguale ad un certo livello, a problemi temporali non semplici, cioè con più intervalli per ogni vincolo. Tuttavia è chiaro come da un STPP con funzioni non semi-convesse si possano ottenere vari STPPs con funzioni semi-convesse. Infatti se si considera una qualsiasi funzione esiste sempre una partizione in intervalli disgiunti del dominio tale che, la restrizione della funzione a ciascun intervallo sia semi-convessa. In parole povere è sempre possibile spezzare il grafico di una qualunque funzione in parti semi-convesse.

Dato un STPP verrà dunque trovata questa partizione per ogni vincolo. Prendere un intervallo per ogni vincolo si otterrà un STPP con funzioni semi-convesse. Ovviamente si otterranno tanti STPPs semi-convessi quante sono le possibili combinazioni degli intervalli dei vincoli. A ciascuno di questi STPPs che soddisfano tutte le ipotesi richieste, si possono dunque applicare i nostri risultati.

Teorema 28 *Si consideri un STPP con funzioni di preferenza non semi-convesse. Si consideri inoltre il TCSPP P' ottenuto da P spezzando gli intervalli dei vincoli di P in sottointervalli aventi funzioni semi-convesse. Allora:*

$$Opt(P) = Opt(P');$$

Dim. Prendendo un intervallo per ogni vincolo, come in P, oppure prendere vari intervalli adiacenti disgiunti tali che la loro unione sia lo stesso vincolo, come in P', è lo stesso per quanto concerne i valori di preferenza associati alle soluzioni. **c.v.d.**

Quindi un metodo per risolvere STPPs con funzioni di preferenza non semi-convesse è quello di ricavare un TCSPP spezzando gli intervalli dell'STPP in modo tale che le funzioni sopra ciascun sottointervallo risultino semi-convesse. A questo punto si applica quanto detto alla sezione precedente, cioè si costruiscono gli STPPs semi-convessi dal TCSPP e si cerca il livello ottimo di preferenza di ciascuno di essi. Le soluzioni ottime dell'STPP

non semi-convesso saranno le soluzioni ottime degli STPPs semi-convessi con livello ottimo di preferenza massimo $best(\bigcup_i(sol(P_i)))$.

3.5 Risultati sulla complessità

Diamo ora dei risultati sulla complessità che comporta risolvere STPPs con funzioni semi-convesse e non. Abbiamo dimostrato nel 26 che si può trovare una soluzione ottima di un STPP con funzioni semi-convesse in tempo polinomiale ma non abbiamo specificato la complessità.

Teorema 29 *Trovare una soluzione ottima di un STPP con funzioni di preferenza semi-convesse ha una complessità di $O(n^3 \times r \times l)$ dove r è la dimensione massima per un intervallo dei vincoli, n è il numero di variabili, e l è il numero totale di livelli di preferenza.*

Dim. Trovare una soluzione ottima significa applicare un algoritmo di consistenza locale, ad esempio STPP_PC-2, e poi trovare una soluzione dell'STP che si ricava nel modo illustrato precedentemente.

STPP_PC-2 deve considerare n^2 triangoli. Per ciascuno di essi, nel caso pessimo, un solo valore di preferenza tra gli r, viene cambiato di un solo livello tra gli l livelli, dunque possiamo avere al più $O(r \times l)$ passi per ciascun triangolo. Inoltre al più n triangoli vengono aggiunti alla coda dopo ciascun aggiornamento. Da questo otteniamo $O(n^3 \times r \times l)$.

Dopo che STPP_PC-2 è terminato non rimane che trovare una soluzione ottima. Questo ha lo stesso costo che trovare una soluzione dell'STP ottenuto considerando gli intervalli con livello ottimo di preferenza. In [8] è stato dimostrato che questo può essere fatto senza backtracking in n passi. Ad ogni passo bisogna trovare un valore per la variabile successiva che sia compatibile con i vincoli tra essa e le variabili a cui è già stato assegnato un valore. Se ci sono d valori possibili non resta che controllare se sono compatibili con al più $n-1$ variabili precedenti. I controlli, dunque, sono $O(n \times d)$, e il costo complessivo di questa fase è $O(n^2 \times d)$.

Anche se l'insieme di valori possibili per ciascuna variabile, che ha cardinalità d, fosse l'insieme di tutti i valori possibili, in realtà non è necessaria una ricerca esaustiva di questo insieme per trovare un valore compatibile. Infatti basta solo considerare i valori che rientrano negli intervalli che si ottengono propagando l'informazione degli assegnamenti precedenti, quindi al più $O(r \times n)$. Dunque la stima $O(n^2 \times d)$ va aggiornata a $O(n^3 \times d)$, ma poichè domina la complessità di STPP_PC-2 possiamo concludere che essa coincide con la complessità dell'intero problema.

Una misura più realistica della complessità può essere ottenuta contando le operazioni aritmetiche. Per ciascun passo di rilassamento sono, infatti, ne-

cessarie $O(r^3)$ operazioni. Infatti, quando si rilassa un triangolo di vincoli, ciascun vincolo ha al più r elementi. Per ciascun elemento del vincolo su cui, si opera il rilassamento, deve essere controllata la preferenza di $O(r^2)$ decomposizioni di questo elemento. Possiamo, quindi, concludere che la complessità totale, secondo questa nuova misura, sarà, $O(n^3 \times r^4 \times l)$. **c.v.d.**

Calcoliamo ora la complessità di un problema ottenuto rilassando la prima delle ipotesi, ovvero un problema temporale non semplice, con funzioni di preferenza semi-convesse.

Teorema 30 *Trovare una soluzione ottima di un TCSPP con funzioni di preferenza semi-convesse ha una complessità $O(k^e \times n^3 \times r \times l)$, dove k è il massimo numero di intervalli per ogni vincolo ed e è il numero di vincoli con funzioni non semi-convesse.*

Dim. Al più verranno generati k^e STPPs, dove ciascun STP seleziona un intervallo per ciascun vincolo. Risolvere ciascuno di essi comporta $O(n^3 \times r \times l)$, come dice il teorema precedente. Poi si devono eliminare le soluzioni che non sono ottime, in quanto non tutte le soluzioni ottime degli STPPs derivati sono soluzioni ottime anche del TCSPP. Questa operazione richiede $O(k^e)$ che è dominato dalla complessità sopra. Dunque otteniamo $O(k^e \times n^3 \times r \times l)$. **c.v.d.**

Diamo ora la stima della complessità per risolvere un STPP con funzioni non semi-convesse. Come ciò venga fatto è stato descritto nel paragrafo precedente.

Teorema 31 *Trovare una soluzione per un STPP ha una complessità pari a $O(g^e \times n^3 \times r \times l)$, dove g è il massimo numero di minimi di una funzione ed e è il numero di vincoli con funzioni non semi-convesse.*

Dim. Per trovare la complessità cercata basta aggiungere a quella trovata nel teorema precedente il costo delle decomposizioni in intervalli con funzioni di preferenza semi-convesse. Questo costo è di $O(e \times r)$ poichè per ciascun intervallo, e ve ne sono al più e, è necessario effettuare una scansione lineare che comporta al più r passi e cercare i minimi. Essendo e $O(n^2)$ e $r \leq r^2$ si ottiene la stessa complessità del Teorema precedente eccetto che il numero di intervalli k qui è rappresentato dal numero dei minimi g, che coincide con il numero massimo di intervalli con funzioni semi-convesse. **c.v.d**

Capitolo 4

Risolutore per vincoli con preferenze

In questo capitolo descriveremo l'implementazione di un risolutore per vincoli temporali semplici con preferenze. Risolvere problemi di questo tipo trova sicuramente riscontro nel campo dello scheduling. Per modellare molti problemi reali, infatti, è utile essere in grado di esprimere delle preferenze sulle distanze temporali che intercorrono tra gli eventi.

Tuttavia, molte volte sfugge come ottenere da tutte le informazioni locali l'ottimo globale. Tra i problemi che il risolutore affronta ci sono quelli in cui si vuole trovare una delle migliori pianificazioni di un certo numero di eventi avendo a disposizione solo delle preferenze locali sulle distanze e sulle durate di tali eventi.

Il risolutore, dunque, da informazioni locali estrae e fornisce informazioni di tipo globale. È importante sottolineare questo in quanto, nel capitolo successivo, verrà mostrato come in effetti si possa fare anche il contrario, cioè come dalle preferenze date alle soluzioni si può indurre la forma delle funzioni locali.

4.1 L'algoritmo

L'implementazione del risolutore è stata realizzata basandosi sul risultato del 26. Presentiamo, ora, lo schema dell'algoritmo fondamentale; in seguito verranno trattate le diverse varianti che sono state implementate per usi specifici.

In breve l'algoritmo prima applica la consistenza locale al problema da risolvere. Questo permette di identificare inconsistenze qualora ve ne siano. A questo punto la rete ottenuta è quella minima ed ha i massimi delle funzioni di preferenza tutti allo stesso livello. Questo permette all'algoritmo di

Algoritmo STPP_SOLVER

1. **input** STPP P ;
2. STPP P'=STPP_PC-2(P);
3. **if** P' inconsistent **then** exit
4. STP P"=REDUCE_TO_BEST(P');
5. **return** EARLIEST_BEST(P")

Figura 4.1: Algoritmo STPP_SOLVER.

identificare l'STP corrispondente al livello ottimo di preferenza di trovarne una soluzione. Questa sarà una soluzione ottima del problema iniziale.

Fase di input

La fase di input inizializza la struttura di STPP del programma all'STPP che si vuole risolvere. L'input può essere dato in due modi: standard input o input da file. Come prima cosa viene richiesto all'utente quale delle due modalità voglia utilizzare.

Lo standard input richiede che l'utente inserisca mediante tastiera, oltre al numero di variabili del problema, tutti i dati relativi ai vincoli non universali del problema da risolvere. Dovranno dunque, per ciascun vincolo da specificare, essere inseriti: 1) gli interi che identificano le variabili, 2) gli estremi dell'intervallo, 3) la funzione di preferenza. Quest'ultima informazione può essere fornita in due modi diversi: 3.1) dando le preferenze punto a punto, cioè inserendo le preferenze per ciascun intero appartenente all'intervallo, oppure 3.2) presupponendo che la funzione sia una parabola, indicando i tre parametri a, b, c che la caratterizzano.

Cogliamo l'occasione per sottolineare che le parabole giocheranno un ruolo molto importante in questa tesi, soprattutto per quanto riguarda il risolutore e il modulo di apprendimento automatico. Più che di parabole in genere ci interesseranno le **funzioni quadratiche semi-convesse** della forma

$$y = ax^2 + bx + c \text{ con } a \leq 0$$

La scelta che verrà talvolta fatta di considerare questa classe di funzioni come funzioni di preferenza è stata fatta per i seguenti motivi. Innanzitutto è facile controllarne la semi-convessità, infatti essa dipende solo dal parametro a, che come abbiamo indicato nella definizione dovrà essere sempre minore o uguale a 0. Inoltre sono funzioni di classe C^∞, cioè continue e derivabili

infinite volte. Questa classe è abbastanza flessibile dal punto di vista della rappresentazione di preferenze. Essa infatti include le rette, e dunque si presta ad esprimere preferenze che cambiano linearmente. Questo significa che anche la rappresentazione dei vincoli classici con STPPs può esser fatta con parabole [1]. Esse si adattano inoltre a rappresentare circostanze in cui si preferiscono valori vicino ad un determinato istante, caso molto frequente nelle applicazioni pratiche.

Se viene scelta la modalità di input da file, il programma legge le informazioni sui vincoli da un file in cui sono state precedentemente caricate. Il file dovrà contenere il numero di variabili del problema e in sequenza gli interi identificativi, gli estremi dell'intervallo e la funzione di preferenza. Anche nella modalità di input da file è possibile dare le preferenze nei due modi indicati per l'input standard. Basterà infatti far precedere i dati da una etichetta che permette al programma di capire se la funzione per quel vincolo è data tramite parametri o punto a punto.

Lo standard input ha pressochè una funzione esemplificativa, nel senso che è utile quando si vuole capire come funziona il programma, quali sono i dati che caratterizzano un vincolo, in che sequenza vengono richiesti. Esso è utile quando si vogliono fare dei piccoli esempi, cioè esempi in cui o il numero stesso delle variabili è piccolo, o il grafo del problema è estremamente sparso.

L'input da file è sicuramente più versatile in quanto permette di gestire problemi grandi a piacere, mantiene le informazioni sul problema iniziale sempre disponibili ed è il tipo di input usato quando si risolvono problemi proposti da un generatore random (di cui parleremo in seguito).

Terminata questa fase, in qualunque delle due modalità il problema da risolvere è stato caricato ed è pronto per essere elaborato dall'algoritmo di consistenza sui cammini.

STPP_PC-2

Questa fase impone sul problema iniziale P la consistenza sui cammini, applicandovi l'algoritmo STPP_PC-2, descritto all'inizio del capitolo precedente. Come prima cosa viene inizializzata la coda inserendovi tutti gli n^2 triangoli possibili di modo che ciascuno di essi venga visitato almeno una volta. Procede dunque, secondo l'ordine della coda, a rilassare i triangoli di vincoli aggiungendo, in caso di effettivo aggiornamento, i triangoli che potrebbero essere influenzati. Come si è già detto a questo punto possono accadere 2 cose:

[1] Ricordiamo che la rappresentazione di vincoli hard in vincoli soft consiste nell'aggiungere all'intervallo del vincolo la funzione costante a 1. Tale funzione è una quadratica semi-convessa con $a = 0$, $b = 0$ e $c = 1$.

1. Viene identificata una inconsistenza, cioè l'intervallo di un vincolo si riduce a $\emptyset$. Se avviene questo l'algoritmo termina e viene segnalata l'inconsistenza. Si osservi che l'inconsistenza è puramente legata alla parte *hard* dei vincoli e non può mai essere causata dalle funzioni.

2. L'algoritmo termina e restituisce un problema equivalente a quello dato ma consistente sui cammini. Questo significa che i vincoli avranno potuto subire i seguenti cambiamenti: (i) la restrizione dell'intervallo,(ii) l'abbassamento dei livelli di preferenza.

   Per quanto riguarda il restringimento degli intervalli per quanto detto in [8], ponendoci ad un livello puramente di vincoli *hard*, possiamo affermare che la rete è minima. Questo significa che negli intervalli sono rimasti solo valori che compaiono almeno in una soluzione[2]. Questa informazione ci sarà utile in una delle successive varianti del risolutore.

   Sulla forma delle funzioni di preferenza, è utile ribadire che a questo punto esse avranno tutte lo stesso massimo e che in generale quelle che erano parabole avranno perso la loro rappresentabilità in forma quadratica.

Alla fine di questa fase, supponendo che si verifichi la seconda delle ipotesi sopra indicate, ci troviamo con un STPP P' consistente sui cammini. Seguendo le orme della dimostrazione del 26, non rimane ora che estrarre il problema di vincoli temporali semplici corrispondente al valore massimo di preferenza.

REDUCE_TO_BEST

Questa funzione estrapola dal problema STPP P' il problema STP P". Una scansione di un vincolo qualsiasi permette di identificare il valore ottimo di preferenza. REDUCE_TO_BEST considera ogni vincolo del problema e comincia una doppia scansione da destra e da sinistra dell'intervallo fermando i due cursori non appena trovi il primo elemento con livello ottimo di preferenza. A questo punto il valore dei due cursori diventa il valore degli estremi dell'intervallo del vincolo temporale semplice corrispondente al vincolo soft in esame.

È utile osservare come l'informazione sul resto degli elementi non vada persa con questa scansione. Questo fatto è molto importante per una eventuale versione incrementale del risolutore [3].

[2] Tale soluzione non è detto che sia ottima.

[3] Questo argomento verrà ripreso nell'ambito delle direzioni di ricerca future nell'ultimo capitolo.

Si osservi inoltre come anche P'' sia minimo. Infatti la rete di vincoli che P'' rappresenta è quello che si otterrebbe applicando PC-2 (nella versione originale per STP semplici) all'STP che si ricava da P considerando gli intervalli corrispondenti a valori di preferenza maggiori o uguali al livello ottimo.

Possiamo dunque affermare che alla fine di questa fase si ottiene un STP P'' consistente, come indicato dal 26, e minimo, per quanto detto sopra.

EARLIEST_BEST

A questo punto l'algoritmo fornisce la soluzione che si ottiene prendendo gli estremi sinistri di tutti gli intervalli di P'' [8]. Infatti, come notato sopra, P'' soddisfa le ipotesi richieste dal da questo teorema. Questa è la soluzione che si trova immediatamente; in realtà è possibile trovarne una qualunque altra senza fare backtracking.

Output.

L'output del risolutore è su file. In questo file non viene solo indicata la soluzione trovata con la sua preferenza, la quale è il livello ottimo di preferenza, ma anche, nel caso in cui i problemi non siano enormi, la rete dopo la consistenza sui cammini. In questo modo ci si può rendere conto di quanto grande è il problema. Guardando infatti la dimensione dei vincoli della rete rilassata si può avere un'idea di quante soluzioni (ottime e non) abbia il problema. Questa informazione può dirci qualcosa anche localmente. Ad esempio su quanto fossero ridondanti o meno i vincoli dati e quali siano i vincoli su cui la propagazione della consistenza ha avuto maggior effetto.

Abbiamo così concluso la descrizione della versione standard del risolutore. Illustriamo ora due varianti mirate l'una a gestire problemi piccoli o molto vincolati, l'altra a generare degli insiemi di soluzioni ottime e non che serviranno per il modulo di apprendimento automatico[4].

4.1.1 Risolutore Completo

Questo algoritmo trova tutte le soluzioni ottime di un STPP P'. Poichè può essere un compito molto laborioso, esso è mirato alla soluzione di problemi piccoli, cioè con un numero limitato di variabili e con intervalli non troppo larghi.

Esso si differenzia da STPP_SOLVER solo per le ultime due fasi. Le procedure di input sono le stesse. Dopo l'inizializzazione viene eseguito PC-2 che può avere uno dei due esiti indicati sopra. Nel caso di un problema consistente si applica REDUCE_TO_BEST al problema consistente sui cammini

[4]Il modulo di apprendimento automatico verrà descritto nel capitolo successivo.

Algoritmo STPP_COMPLETE_SOLVER

1. **input** STPP P ;
2. STPP P'=STPP_PC-2(P);
3. **if** P' inconsistent **then** exit
4. STP P"=REDUCE_TO_BEST(P');
5. **return** ALL_BEST(P")

Figura 4.2: Algoritmo STPP_COMPLETE_SOLVER.

e si ottiene l'STP minimo e consistente P''. STPP_COMPLETE_SOLVER, invece, trova tutte le soluzioni ottime, sfruttando la minimalità di P''.

ALL_BEST

Questa procedura cerca tutte le soluzioni di P''. Essa procede facendo una ricerca $depth - first$, non facendo mai backtracking durante la costruzione di una soluzione. Infatti essendo P'' minimo esso è anche decomponibile [8]. Questo significa che qualsiasi assegnamento parziale consistente può essere esteso ad una soluzione completa.

Nel primo capitolo si è già detto che negli STPPs viene introdotta una variabile , detta "inizio del mondo", rispetto alla quale vengono fatte tutte le misure temporali. Per semplicità in tutti i nostri problemi le verrà assegnato il valore 0. ALL_BEST guarda dunque un qualsiasi vincolo in P'' tra la variabile "inizio del mondo" ed un'altra. L'intervallo di tale vincolo contiene solo valori che compaiono in almeno una delle soluzioni e dunque un qualsiasi elemento, di tale intervallo, darà origine ad un assegnamento che potrà essere esteso ad una soluzione. Questo significa che non dovrà mai essere ritrattato. Il passo successivo è di scegliere un'altra variabile e di cercare, nel vincolo tra questa e una delle due precedentemente instanziate, un valore che sia compatibile con tutti gli assegnamenti precedenti. Questo valore, sempre per la decomponibilità, sicuramente esiste e dunque non è necessario nessun backtracking per trovare una soluzione. Una volta completata un prima soluzione esso continua la ricerca, iterando la procedura qui descritta, fino a trovarle tutte.

Output

L'output di questa variante è sempre su file. Esso consiste nella rete rilassata e nella lista completa delle soluzioni ottime.

L'utilità di questo algoritmo è quella di offrire la possibilità di scegliere tra le soluzioni ottime. Non è difficile pensare, infatti, a delle preferenze che non si riescono a codificare come funzioni di preferenza locali ma che si possono individuare una volta che si abbia una soluzione sotto mano. Inoltre può accadere che cambino alcune circostanze da quando si era impostato il problema e che questo renda alcune soluzioni ottime preferibili rispetto ad altre.

4.1.2 Risolutore globale random

Algoritmo STPP_RANDOM_SOLVER

1. **input** STPP P ;
2. STPP P'=STPP_PC-2(P);
3. **if** P' inconsistent **then** exit
4. STP P"=REDUCE_TO_STP(P');
5. **return** RANDOM_SOLUTIONS(P",P')

Figura 4.3: Algoritmo STPP_RANDOM_SOLVER.

Questo programma è stato disegnato specificatamente per essere utilizzato in alcuni esperimenti del modulo di apprendimento automatico [5]. Il task di STPP_ COMPLETE_ SOLVER è quello di creare due insiemi disgiunti di soluzioni ottime e non di un STPP P indicandone le preferenze.

Una volta che sia stato eseguito STPP_PC-2, si considera direttamente l'STP P" ottenuto considerando gli intervalli di P. Successivamente si generano, con un approccio random, due insiemi di soluzioni le cui dimensioni vengono specificate dall'utente.

REDUCE_TO_STP

Questa funzione praticamente riduce a livello *hard* i vincoli di P". Cioè costruisce un STP avente come intervalli gli insiemi di valori che corrispondono al livello ottimo di preferenza. Questo non significa che si perda l'informazione sulle funzioni di preferenza, che viene invece utilizzata per trovare il livello di preferenza per le soluzioni.

RANDOM_SOLUTIONS

[5] Si veda il Capitolo 5.

Questo programma genera soluzioni del problema P'' con un approccio random. Non viene più, infatti, adottata alcuna politica di ricerca ma si procede casualmente.

In dettaglio: come prima viene assegnato 0 alla variabile "inizio del mondo", poi viene considerato un vincolo tra essa e un'altra variabile e viene *scelto a caso* un elemento dell'intervallo corrispondente al vincolo. Viene poi presa in considerazione un'altra variabile ed un vincolo tra essa e una delle precedentemente instanziate. Di nuovo vengono fatti dei tentativi *scegliendo a caso* un elemento in questo intervallo e fermandosi quando se ne trova uno che è compatibile con tutti gli assegnamenti precedenti. La decomponibiltà garantisce di non dover mai tornare indietro durante la costruzione di una soluzione.

Una volta generato un assegnamento globale consistente esso viene passato ad una funzione che calcola il livello globale di preferenza ricavando da P' i livelli locali individuati. Questa funzione ottiene il valore di preferenza globale di una soluzione s, chiamiamolo $pref(s)$, componendo quelle locali secondo quanto indicato nel capitolo sui vincoli temporali con preferenze. Supponiamo, infatti, che alla variabile X_i la soluzione assegni il valore v_i. Allora, si avrà che

$$pref(s) = \times\{f_{ij}(v_j - v_i)|(v_i, v_j) = s\downarrow_{X_i,X_j}\}$$

dove $i, j = \{1, \cdots, n\}$ e $i < j$. Visto che il semianello che stiamo considerando è S_{FCSP}, e in esso l'operatore moltiplicativo è min, la formula diventa:

$$pref(s) = min\{f_{ij}(v_j - v_i)|(v_i, v_j) = s\downarrow_{X_i,X_j}\}.$$

Questo procedimento viene fatto fino a quando non si sono creati due insiemi disgiunti[6] contenenti tante soluzioni quante sono state richieste nella fase di input.

Output

L'output di questo programma è costituito da tre file. Due contengono le soluzioni e sono quelli di cui abbiamo parlato sopra. Nel terzo viene indicata la versione *hard* di P. Questo file servirà come input al modulo di apprendimento automatico.

[6]Gli insiemi devono essere disgiunti in quanto uno servirà per allenare il modulo di apprendimento (training set) e l'altro per verificare i risultati (test set).

4.2 Scelte implementative

4.2.1 Il linguaggio di programmazione: C++

La scelta di C++ come linguaggio di programmazione è stata fatta per le seguenti motivazioni:

1. È un linguaggio object-oriented. Il fatto di poter definire degli oggetti è molto utile nel nostro caso. In particolare sono stati definiti tre oggetti: l'oggetto vincolo temporale semplice, l'oggetto vincolo temporale con preferenze, e l'oggetto triangolo di vincoli.

2. C++, inoltre, è un linguaggio molto flessibile ed espressivo. Esso permette di gestire facilmente delle strutture dati, come una coda. Questo è stato utile nel programma in quanto, nell'implementazione di STPP_PC-2, è stata utilizzata proprio una coda.

3. In questo linguaggio è particolarmente semplice la gestione della memoria dinamica. L'allocazione di memoria dinamica è stata utilizzata per la rappresentazione, tramite una matrice, degli STPPs.

4. Questo progetto è stato realizzato per essere incorporato con un planner già esistente. Si tratta del sistema EUROPA progettato dal centro di ricerche NASA Ames, Moffett Field, CA, USA. Essendo l'implementazione di questo sistema prevalentemente in C++ si è ritenuto che l'utilizzo dello stesso linguaggio potesse facilitare la fase di embedding di questo progetto.

4.2.2 Rappresentazione di vincoli e di problemi di vincoli

Come è stato già accennato, sono stato definiti tre oggetti. Descriviamo ora la loro definizione come classe C++.

- Un *vincolo temporale semplice* è stato rappresentato da un array di due interi, destinato a contenere il nome delle variabili vincolate, e da due interi rappresentanti, rispettivamente, l'estremo destro e sinistro dell'intervallo . La scelta di ammettere solo vincoli con estremi dell'intervallo interi è stata fatta sulla base di due circostanze. Prima di tutto il legame tra questo fatto e la complessità dell'algoritmo di consistenza sui cammini [8], e in secondo luogo la necessità di volersi uniformare al codice di EUROPA. EUROPA infatti impiega già un risolutore per vincoli temporali semplici senza preferenze il quale, appunto, ammette solo intervalli con estremi interi.

Si potrebbe pensare che questo risolutore possa soppiantare completamente quello per gli STP citato sopra. Infatti, nel nostro framework un STP è semplicemente un STPP in cui tutte le funzioni di preferenza sono la costante uguale a 1. In realtà il costo computazionale del nostro algoritmo è molto più pesante nel caso di un tale problema se lo si confronta con uno che non tiene conto delle preferenze. Quindi, pur essendo vero che STPP_PC-2, può gestire anche problemi semplici senza preferenze, non sembra conveniente dal punto di vista computazionale sostituirlo al risolutore di EUROPA. Esso è destinato ad affiancare tale risolutore e a gestire problemi con funzioni di preferenza non banali.

- Un *vincolo temporale semplice con preferenze* ha una rappresentazione molto più complessa. La definizione della classe corrispondente a questo oggetto consiste in: un array di due interi che conterrà gli interi identificativi per le variabili; un array di interi destinato a contenere i valori dell'intervallo; un array di reali per i valori di preferenza e tre reali che rappresentano i tre parametri nel caso in cui la funzione di preferenza venga espressa come una parabola.

  Si rende ora necessario spiegare perchè sia stata scelta questa rappresentazione discretizzata dell'intervallo e della funzione di preferenza. Il motivo è implicito nel funzionamento di STPP_PC-2. Il primo punto dove una rappresentazione continua creerebbe problemi è il passo di rilassamento, principalmente per come sono state definite le operazioni di intersezione e composizione di vincoli. Infatti, esse sono state definite, a livello delle funzioni, in modo puntuale. In altre parole noi sappiamo sempre, dato un determinato elemento dell'intervallo, quale sarà la preferenza ad esso associata, ma non si sa nulla di quale sia la rappresentazione parametrica globale della funzione che si ottiene. Ad esmpio, partendo da un vincolo avente come funzione di preferenza una parabola semi-convessa [17], dopo STPP_PC-2, sul medesimo vincolo ci potrà essere una funzione, sempre semi-convessa [17], ma con una forma molto diversa da quella di una parabola. Potrebbe trattarsi di una funzione a scalini, di una parabola troncata ad un certo livello, molte volte non si tratterà neanche di una funzione continua. In breve, non è possibile mantenere una rappresentazione parametrica della funzione.

  A questo problema a livello macroscopico, se ne affianca un altro più specifico riguardante l'operazione di composizione di vincoli. Ricordiamo che ad un elemento dell'intervallo del vincolo composto, viene associata come preferenza il massimo dell'insieme contenente i minimi delle preferenze delle decomposizioni. Nel continuo le decomposizioni in cui si può spezzare un dato valore sono infinite. Si potrebbe proporre

di fissare una certa precisione. Supponiamo ad esempio di usare fino a 4 cifre dopo la virgola. Ma in questo caso una banale moltiplicazione per 10000 porta a qualcosa che è rappresentabile con gli interi, e quindi anche dal nostro sistema.

Un'altra obbiezione può essere fatta sull'uso degli array nella definizione dei vincoli con preferenze. In effetti, è possibile definire in C++ una classe di array con grandezza variabile. Il motivo per cui questo non è stato fatto in questa prima versione dell'implementazione è il seguente. Questa ricerca è iniziata non tanto con lo scopo di fornire un risolutore efficientissimo per risolvere STPPs quanto per fare una primo tentativo di apprendimento automatico su questo tipo problemi. L'implementazione di un risolutore si è, dunque, resa necessaria sia per creare degli insiemi di esempi su cui allenare il modulo, in un certo tipo di esperimenti, sia per risolvere i problemi appresi dal modulo e riuscire a darne una valutazione. Tutto questo verrà trattato dettagliatamente in seguito, per ora basti sapere che vi è la consapevolezza della possibilità di una forte ottimizzazione, ma essa è stata posposta per permettere di ottenere di risultati di altro tipo [7].

- Un *problema di vincoli temporali semplici oppure con preferenze* viene rappresentato come una matrice di oggetti che definiscono il tipo di vincolo, classico o con preferenze. La memoria per queste matrici viene allocata dinamicamente. La rappresentazione matriciale di un STPP o di un STP appare ovvia se si pensa ai grafi che corrispondono a questi problemi. La rappresentazione degli STP mediante grafi è descritta in [8]. Per gli STPPs la struttura del grafo è identica. Ai nodi corrispondono le variabili e ai vincoli gli archi. Gli archi vengono etichettati con l'intervallo a cui viene sovrapposta la funzione. Nella matrice, dunque, l'elemento corrispondente alla colonna i e alla riga j contiene un vincolo tra le variabili i e j.

  Facciamo ora un'osservazione sui vincoli. Un vincolo tra la variabile i e la variabile j induce un vincolo "reciproco" tra la variabile j e la variabile i. Gli estremi dell'intervallo reciproco sono legati a quelli del vincolo originale come segue: se il vincolo T ha intervallo $I = [a, b]$, allora l'intervallo del vincolo reciproco $r(T)$ sarà $r(I) = [-b, -a]$. Per quanto riguarda la funzione di preferenza se T ha funzione di preferenza f, allora rf sarà la funzione simmetrica di f rispetto all'asse delle y, cioè $rf(-x) = f(x)$.

[7] In realtà nell'ultimo capitolo verrà descritto un nuovo risolutore molto più veloce che è in fase di sviluppo. Esso, tuttavia, utilizza un approccio completamente diverso e non richiede l'utilizzo di array a dimensione variabile.

Sembra dunque che basti definire gli $n(n-1)/2$ elementi della matrice triangolare superiore escludendo quelli della diagonale, in quanto nel nostro sistema non esistono vincoli fra una variabile e se stessa. In realtà STPP_PC-2 impone di fare una scelta. Infatti non si ha nessuna garanzia sul verso con cui un vincolo apparirà nei triangoli che vengono sottoposti al rilassamento. Allora o si tengono in memoria solo i vincoli necessari e poi, ogni volta che il verso è quello opposto si calcola il vincolo reciproco, oppure durante la fase di input ogni volta che viene caricato un vincolo automaticamente viene caricato anche il suo reciproco. Sperimentalmente si è verificato che in molti casi il secondo approccio è migliore. Infatti STPP_PC-2, se si trova di fronte un problema particolarmente tedioso da risolvere, può visitare uno stesso triangolo più e più volte. In pratica il primo metodo, anche se eviterebbe di definire alcuni vincoli reciproci, porterebbe a ridefinirne altri molte volte.

Vogliamo discutere brevemente la definizione dei vincoli di default. Tali vincoli dovranno rappresentare il fatto che tra due variabili non è stato specificato nessun vincolo. È importante definirli in quanto sono i vincoli a cui il costruttore della classe inizializza un nuovo oggetto. La scelta è tra dei vincoli fittizi contenenti solo l'informazione che non sono ancora stati inizializzati oppure dei vincoli universali. La prima possibiltà è plausibile se la densità dei vincoli definiti è abbastanza alta. Più precisamente deve essere tale che dai vincoli specificati si possono trarre le informazioni su tutti i vincoli non specificati. Se la densità però non è sufficientemente alta alcuni vincoli possono rimanere indefiniti e questo costituisce un problema per STPP_PC-2. L'alternativa è quella di definire come vincoli di default i vincoli universali. L'implementazione di questa scelta non è però del tutto banale. Infatti nel nostro sistema non è possibile costruire un vincolo il cui intervallo sia tutta la retta reale, perchè esso deve essere contenuto in un array. D'altra parte il vincolo anche se non può essere infinito deve essere grande abbastanza da non comportare l'esclusione di nessun valore, nei passi di rilassamento. Più precisamente, supponiamo di aver definito m vincoli, e denotiamo con $a_1, \cdots, a_m$ gli estremi inferiori degli intervalli e con $b_1, \cdots, b_m$ gli estremi superiori. Allora gli estremi dell'intervallo del vincolo di default, $I_d = [a_d, b_d]$, dovranno soddisfare $a_d \leq min\{2(a_i + a_j) | i \neq j\}$ e $b_d \geq max\{2(b_i + b_j) | i \neq j\}$. Questo può essere calcolato solo dopo che i vincoli siano noti. Per quanto riguarda la funzione di preferenza sarà la costante uguale a 1. In realtà, si cerca di raggruppare i problemi in classi caratterizzate dalla dimensione dei vincoli di default. L'inizializzazione dunque potrà rimanere la stessa per tutti problemi di una stessa classe.

Questo approccio verrà seguito in particolare per problemi generati con il generatore random[8].

Queste osservazioni possono suggerire anche un'altra possibile ottimizzazione che si può introdurre in STPP_PC-2 nel caso in cui il problema da risolvere abbia una densità abbastanza grande. L'idea è di ordinare i triangoli nella coda in modo che compaiano prima quelli con due lati fortemente vincolati ed il terzo di default o, comunque, più largo. Questo ordinamento accelererebbe di molto la propagazione dell'informazione ai vincoli più larghi e diminuirebbe i passi di rilassamento.

4.2.3 Rappresentazione dei triangoli di vincoli e della coda di triangoli

Un *triangolo di vincoli* viene rappresentato da una classe nella cui definizione compaiono, semplicemente, tre interi I, K, J. Il primo e l'ultimo rappresentano le variabili del vincolo che verrà rilassato, mentre il terzo corrisponde alla variabile che completa al triangolo. Può essere utile avere una rappresentazione geometrica. In breve, il triangolo $(1, 3, 2)$ rappresenta il un grafo orientato completo tra la prima, la seconda e la terza variabile. La *coda di triangoli* è stata definita nel modo usuale, [6], tramite un puntatore alla testa della coda.

Anche la ricerca $depth-first$, utilizzata in STPP_COMPLETE_SOLVER, è implementata seguendo lo schema classico ricorsivo [6].

4.2.4 Struttura del programma

Vorremmo ora sottolineare come il programma sia stato implementato cercando di suddividere più possibile le varie procedure in funzioni specifiche, evitando dunque la modalità in-line. Questo ha reso il programma molto più flessibile. Tale approccio ha facilitato innanzitutto l'implementazione delle due varianti a partire dal risolutore originale, ma ha anche permesso di utilizzare funzioni globali di preferenza diverse dallo schema $max - min$. Funzioni di questo tipo sono state utilizzate per gli esperimenti condotti durante il periodo passato a NASA Ames, che verranno descritti nell'ultimo capitolo.

[8] Al generatore è dedicata la sezione 5.2.

Capitolo 5

Risultati sperimentali del risolutore

In questo capitolo presentiamo i risultati sperimentali del risolutore. Prima di tutto mostreremo attraverso un esempio come il risolutore modifica una rete di vincoli durante il processo che porta a trovare una, o tutte le soluzioni ottime.

Descriveremo quindi le caratteristiche e l'implementazione del generatore random di STPPs. I problemi da esso generati saranno oggetto degli esperimenti descritti nell'ultima sezione.

5.1 Esempio di applicazione del risolutore

Illustriamo ora un esempio di un problema a cui si può applicare il risolutore.

Il primo passo è quello di **definire il problema**.

Prendiamo in considerazione MARS ROVER K9, il robot progettato dal centro NASA Ames per arrivare su Marte e fare una serie di esperimenti, inviandone i risultati a terra. È plausibile che esso debba effettuare le seguenti due operazioni: fotografare una roccia, ed estrarne un campione. Indicheremo queste attività con gli acronimi IR (Image Retrieving) e SR (Sample Retrieving). Ciascuna attività rappresenta un evento che deve essere pianificato entro un certo orizzonte di tempo.

Il nostro orizzonte, cioè il termine entro cui terminare le attività, sarà di 24 ore come il giorno terrestre. Da esperimenti condotti sulla terra è possibile avere una stima di quanto tempo occorrerà per compiere ciascuna attività. Una durata ragionevole per l'IR può essere compresa tra 1 e 10 ore. Operazioni banali, come il puntamento dell'obiettivo, la messa a fuoco dell'immagine, il controllo della stabilità dell'immagine, che a un fotografo terrestre possono sembrare banali, possono rivelarsi estremamente difficili in un ambiente come quello di Marte, a causa della temperatura, del tempo, etc.

Inoltre K9 deve anche prendere delle pause per permettere alla batteria di ricaricarsi. Quest'ultima procedura assume una importanza ancora maggiore quando si considera un SR per il quale possono servire da 5 a 15 ore.

Supponiamo di cominciare a contare il tempo all'alba marziana. Essa costituirà per noi l'ora 0. Una volta fissato questo punto di riferimento, possiamo stabilire dei limiti orari entro cui le attività devono iniziare. IR, ad esempio può cominciare in qualunque momento tra l'alba e 7 ore da essa, mentre SR può avere inizio tra 5 e 10 ore dopo l'alba. Il range di IR permetterà di ottenere foto con diverse esposizioni di luce, quello per SR, di raccogliere materiale a diverse temperature. Essendovi un unico K9 che deve assolvere ad entrambe i compiti, è ragionevole vincolare anche la fine della parte fotografica e l'inizio dell'estrazione. Può essere ragionevole, ad esempio, che SR cominci da quattro ore prima a quattro ore dopo la fine di IR. Con questo abbiamo terminato la fase di definizione del problema.

La seconda fase è di **modellamento del problema**. Essa consiste nel trasformare i dati che si hanno in un problema di vincoli. Mostreremo ora come tutte le informazioni che abbiamo possano essere rappresentate da un sistema di vincoli temporali semplici *senza preferenze*.

Quando si vincolano degli eventi, per ottenere un loro schedulazione, si tratta di specificare due tipi di vincoli: vincoli sulla durata dell'evento e vincoli sulle distanze tra gli eventi. A ciascun evento vengono associate due variabili, corrispondenti rispettivamente all'inizio e alla fine dell'evento. Indicheremo con IR_s ed IR_e le variabili dell'Image Retrieving e con SR_s e SR_e le variabili del Sample Retrieving. I vincoli sulla durata possono essere rappresentati da un vincolo temporale semplice tra IR_s ed IR_e, T_{IR}, e tra SR_s e SR_e, T_{SR}. L'intervallo di T_{IR}, sarà così definito: $I_{IR} = [1, 10]$, mentre quello per di T_{SR} diventa $I_{SR} = [5, 14]$.

Passiamo ora a definire i vincoli sulle distanze. A tal scopo è necessario introdurre la variabile "inizio del mondo" che indicheremo con BW, Beginning of the World. Nel nostro caso essa sarà la variabile corrispondente all'alba marziana e avrà un unico assegnamento ammesso quello a 0. Dalle informazioni sui tempi di inizio delle attività otteniamo due vincoli: uno tra BW e IR_s , con intervallo $[0, 7]$; l'altro tra BW e SR_s con intervallo $[5, 10]$. Non abbiamo ancora sfruttato l'informazione riguardante la posizione reciproca dei due eventi. Essa può essere tradotta da un vincolo tra IR_e e SR_s apponendovi l'intervallo [-4,4]. Tutti gli altri vincoli sono di default e vengono inizializzati all'intervallo $[-12, 12]$. Abbiamo così modellato il problema facendolo rientrare nello schema STP.

La rete definita corrisponde al grafo rappresentato nella figura 5.1.

Tuttavia i vincoli *hard* non sono abbastanza espressivi quando si tratta di rappresentare preferenze sulle varie durate. I vincoli *hard* ci hanno permesso

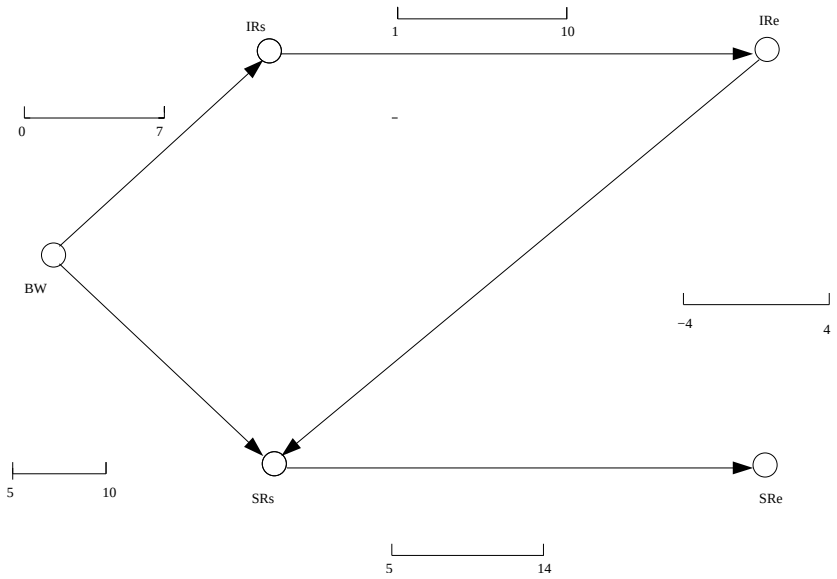

Figura 5.1: Grafo che rappresenta il problema di vincoli temporali semplici senza preferenze.

di dire quali durate o distanze sono ammesse tra le variabili, ma non contemplano la possibilità che una durata sia preferibile, non in modo categorico rispetto ad un'altra. Facciamo alcuni esempi. È ragionevole pensare che sia preferibile fare le foto il più tardi possibile, quando c'è più luce, e che tutta l'attività termini il prima possibile, per risparmiare energia e per evitare di sovraesporre le componenti delicate di K9 agli agenti climatici. Inoltre si può avere una preferenza nei riguardi di un SR che cominci intorno alle 7, perchè il terreno ha la temperatura ideale, e che duri il più possibile per avere un referto più accurato. Si potrebbe anche dare una penalità maggiore ai casi in cui le due attività si sovrappongono.

Si osservi come le cose indicate sopra siano auspicabili, ma non assolutamente necessarie. Questo tipo di informazioni, $soft$, cioè non categoriche, può essere espresso mediante vincoli temporali semplici con preferenze. Si noti come queste informazioni non aggiungono nulla alla struttura del problema, non richiedono nuove variabili o nuovi vincoli. Basta dunque aggiungere delle funzioni che rappresentino quanto espresso fedelmente. Tra le varie ne abbiamo scelte alcune e le abbiamo indicate nella figura 5.2.

Passiamo ora ad applicare il nostro algoritmo.

Come prima cosa viene creato il file di input: prima di impostarlo l'unica cosa da fare è quella di decidere gli interi identificativi delle variabili. Possiamo supporre che siano 0 per BW, 1 per IR_s, 2 per IR_e, 3 per SR_s, e 4 per

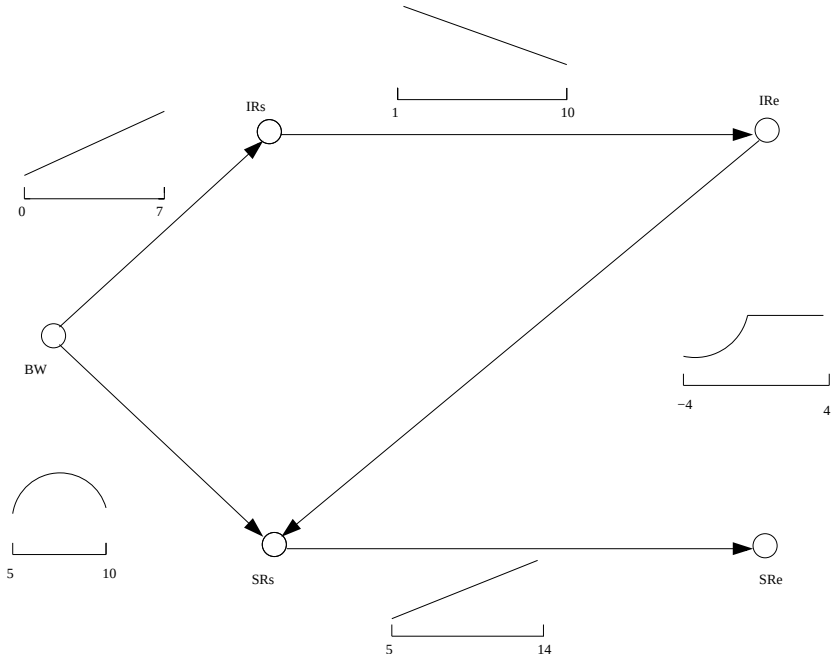

Figura 5.2: Grafo che rappresenta il problema con vincoli temporali semplici con preferenze.

SR_e. Ecco come apparirà il file di input:

5
y
0 1
0 7
p
0 0.1 0.3
y
1 2
1 10
p
0 -0.07 0.93
y
0 3
5 10
p
-0.04 0.61 -1.66
y 2 3

-4 4
d
0.1 0.2 0.6 0.8 1 1 1 1 1
y
3 4
5 14
p
0 0.08 0.2
n

Spieghiamo come si legge il file. Il primo intero è sempre il numero di variabili. L'etichetta y avverte che segue la definizione di un nuovo vincolo, quella n che si è terminato di dare i vincoli. Ad esempio
3 4
5 14
p
0 0.08 0.2
vuol dire che il vincolo tra la variabile 3, SR_s e la variabile 4, SR_e, ha un intervallo di estremi 5 e 14 e ha come funzione di preferenza la parabola , che in questo caso è una retta, di equazione $y = 0x^2 + 0.08X + 0.2$. L'etichetta p precede la descrizione di una funzione di preferenza mediante parametri, l'etichetta d se vengono date le preferenze punto a punto.

A questo punto inizia l'esecuzione di STPP_SOLVER. Nella figura 5.3 appare il problema dopo che è stata imposta la consistenza sui cammini, cioè dopo STPP_PC-2.

Si osservi che gli intervalli dei vincoli tra BW e SR_s, e tra SR_s e SR_e sono stati ristretti. Inoltre tutte le funzioni di preferenza si sono abbassate al livello ottimo di preferenza, in questo caso 0.6 .

Si passa ora alla identificazione del sottoproblema ottimo. Viene, dunque, applicata la funzione REDUCE_TO_BEST al problema sopra raffigurato. Il problema ottenuto è rappresentato dal grafo nella figura 5.4.

Come si può osservare è un problema di vincoli *hard*. Infatti il valore di preferenza è la costante 0.6 su tutti i vincoli. Possiamo già trarre delle conclusioni. Dal vincolo tra BW e SR_s si vede come in una soluzione ottima la fase di recupero di un reperto debba iniziare alla settima ora. Dal vincolo tra SR_s e SR_e si vede che la durata ideale di questo evento è di 5 ore. L'ultima scelta sembra in contraddizione con la preferenza locale che si era espressa per questo vincolo. Tuttavia questi risultati tengono conto di tutte le preferenze,

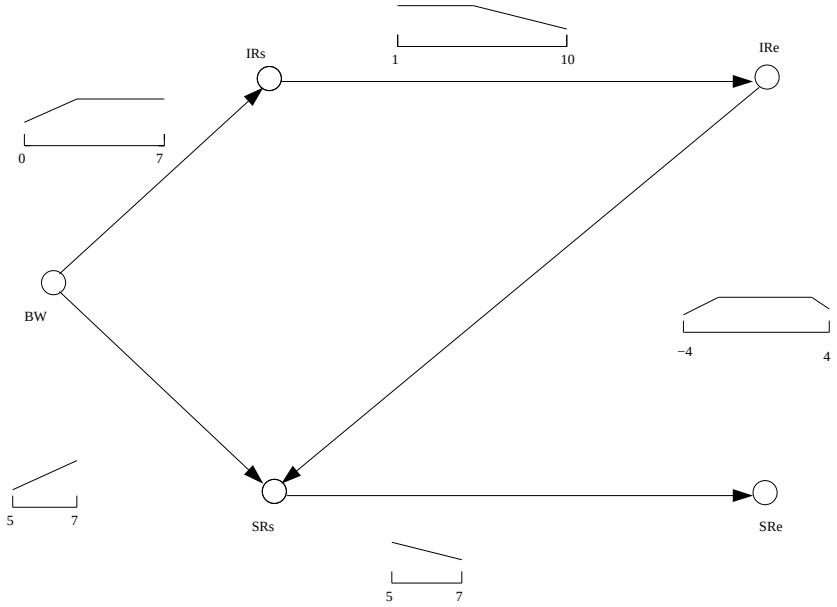

Figura 5.3: STPP dopo STPP_PC-2.

ed evidentemente una durata maggiore avrebbe penalizzato gli altri vincoli più di quanto avrebbe reso migliore la soluzione.

Dal vincolo tra IR_e ed SR_s, appare chiaro che in una soluzione ottima le due attività non si possono sovrapporre, oltre ad un dato limite. La soluzione ottima trovata dal risolutore è quella corrispondente agli inizi più precoci e alle durate più brevi:

BW	IR_s	IR_e	SR_s	SR_e
0	3	4	7	12

Tabella 5.1: Soluzione ottima.

Se invece avessimo utilizzato la variante STPP_COMPLETE_SOLVER avremmo ottenuto, sempre partendo dal grafo della figura 5.1, tutte le soluzioni ottime (Tabella 5.2).

Applicando invece l'algoritmo STTP_RANDOM_SOLVER, richiedendo due insiemi con 5 soluzioni qualsiasi ciascuno, avremmo potuto ottenere due insiemi come i seguenti rappresentati nelle tabelle 5.3 e 5.4:

Come si può notare, i due insiemi contengono soluzioni con valori di preferenza diversi, ovviamente tutti minori o uguali al livello ottimo di preferenza.

BW	IR_s	IR_e	SR_s	SR_e
0	3	4	7	12
0	3	5	7	12
0	3	6	7	12
0	3	7	7	12
0	4	5	7	12
0	4	6	7	12
0	4	7	7	12
0	4	8	7	12
0	5	6	7	12
0	5	7	7	12
0	5	8	7	12
0	5	9	7	12
0	6	7	7	12
0	6	8	7	12
0	6	9	7	12
0	7	8	7	12
0	7	9	7	12

Tabella 5.2: Insieme completo delle soluzioni ottime.

BW	IR_s	IR_e	SR_s	SR_e	Pref
0	6	8	5	11	0.2
0	4	6	6	11	0.56
0	3	4	7	12	0.6
0	0	3	6	12	0.3
0	6	10	7	12	0.2

Tabella 5.3: Insieme di soluzioni costruito da STPP_RANDOM_SOLVER.

BW	IR_s	IR_e	SR_s	SR_e	Pref
0	1	6	7	12	0.4
0	6	11	7	12	0.1
0	2	11	7	12	0.1
0	6	9	6	12	0.2
0	6	8	6	11	0.56

Tabella 5.4: Insieme di soluzioni costruito da STPP_RANDOM_SOLVER.

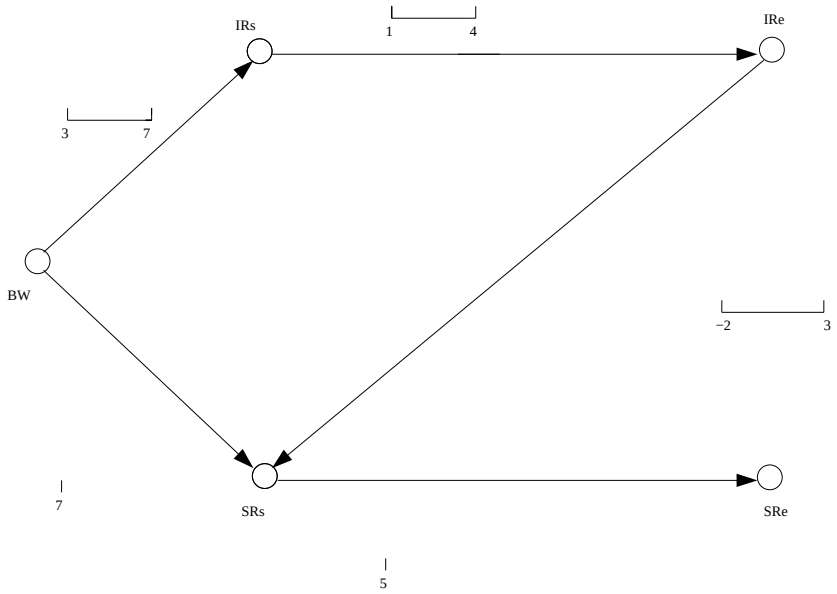

Figura 5.4: STP ottenuto applicando REDUCE_TO_BEST.

5.2 Un generatore random per STPPs

Durante, e soprattutto dopo, la fase di implementazione del risolutore, si sono resi necessari degli esempi su cui valutare il comportamento dell'algoritmo. Oltre ad esempi come quello visto nella sezione precedente è stato implementato un generatore random per poter valutare il comportamento del risolutore su intere classi di problemi.

Per costruirlo, si è cercato di rimanere il più fedeli possibile al generatore random per vincoli temporali senza preferenze in [35]. Questo generatore ha cinque parametri: (1) n, il numero delle variabili, (2) k, il numero di intervalli per ciascun vincolo, (3) $R = [Inf, Sup]$, il range dei vincoli, (4) T_I la $tightness$ cioè la percentuale di R che deve essere coperta dagli intervalli di un vincolo, e (5) P_C la probabilità che un vincolo esista, in pratica la densità dei vincoli definiti.

Nel nostro caso k è sempre uguale a 1 e dunque non è un parametro significativo. Negli esperimenti condotti, in [35], T_I aveva sempre valori molto alti come 90% o 95%. Questo serviva ad evitare che la maggior parte dei problemi fosse inconsistente. I nostri problemi non vengono ben rappresentati da questo parametro, per il seguente motivo. Supponendo di aver fissato R, dire che ogni vincolo deve avere un intervallo che copre il 90% di R, equivale a ottenere intervalli quasi uguali su tutti i vincoli. Questo è dovuto alla rappresentazione discreta dei nostri intervalli. Si pensi ad esempio al caso in cui $R = [0, 100]$, porre $T_I = 90\%$ significa avere solo 10 intervalli diversi a disposizione per tutti i vincoli che si vogliono generare. Inoltre

questi problemi non rappresentano bene i casi reali. Infatti nei, problemi reali, come ad esempio le applicazioni su K9 i vincoli definiti tendono ad essere molto stretti. Tuttavia, assegnando a T_I valori più bassi, rischiamo che la maggior parte dei problemi generati sia inconsistente e dunque non adatta per testare le ultime due fasi degli algoritmi. Un vantaggio di aver tentato quest'ultimo approccio è stato quello di rendersi conto che, in effetti, STPP_SOLVER, e le sue varianti, trovano le inconsistenze. Sulla base di questa esperienza, si è optato per un generatore di problemi consistenti. In pratica, il numero di variabili rimane sempre un parametro. Per ciascuna variabile viene scelto a caso un valore, entro un certo range, ottenendo così un assegnamento globale. Bene, tale assegnamento sarà una soluzione del problema, attorno ad essa verranno costruiti i vincoli.

Nel nostro caso sono inoltre necessari dei parametri per le funzioni di preferenza. Essendo in generale il controllo della semi-convessità abbastanza laborioso, e non perdendo troppo in generalità, è stato deciso di utilizzare solo funzioni quadratiche semi-convesse rappresentabili da un'equazione del tipo: $y = ax^2 + bx + c$. Su ciascun vincolo da definire viene dunque inizializzata una parabola vera e propria, oppure una retta, oppure una retta costante. Nel caso della parabola viene poi introdotta una perturbazione casuale nei tre parametri a, b, c, secondo tre percentuali da specificarsi in input.

Descriviamo ora i parametri del nostro generatore:

1. **n:**, il numero di variabili;
2. **r:** un intero che specifica il range entro cui verranno scelti i valori per il primo assegnamento;
3. **max:** un intero che determina la massima espansione destra e sinistra degli intervalli;
4. **d:** densità, quanti, tra gli n^2 possibili vincoli, devono essere definiti dal generatore;
5. **pa:** percentuale di perturbazione del parametro a;
6. **pb:** percentuale di perturbazione del paramentro b;
7. **pc:** percentuale di perturbazione del parametro c.

Ecco, invece, come procede l'algoritmo
RANDOM_STPP_GENERATOR(n,r,max,D,pa,pb,pc):

1. Osserviamo intanto che n comprende anche la variabile "inizio del mondo" che, come al solito avrà sempre valore 0. Il primo passo consiste nel

scegliere a caso $n-1$ interi in $[0, r]$, e di assegnarli alle variabili nell'ordine di scelta (il primo valore pescato va assegnato alla variabile 1 , il secondo alla 2 etc.). Questo assegnamento costituisce la soluzione dalla quale vogliamo costruire il problema di problemi di vincoli temporali semplici con preferenze.

2. Viene calcolato il numero di vincoli espresso dalla percentuale d, diciamo v. Allora per v volte l'algoritmo compie i seguenti passi.

3. Sceglie due interi in $[0, n-1]$. Essi saranno gli interi identificativi del vincolo che si definisce. Si noti che l'assegnamento iniziale identifica un elemento, chiamiamolo u, che dovrà appartenere all'intervallo di questo vincolo. Si procede, dunque, scegliendo due valori in $[0, max]$, diciamo ed, espansione destra, e es, espansione sinistra. Essi vengono rispettivamente sommati e tolti ad u, di modo che a questo vincolo verrà associato l'intervallo $I = [u - es, u + ed]$.

4. A questo punto l'intervallo è stato fissato, non rimane che definire la funzione di preferenza. A tal fine viene estratto un numero tra 0 e 15 a caso. Se esso è compreso tra 0 e 5, la funzione definita sarà una parabola propria convessa, se è tra 5 e 10, una retta, e se è tra 10 e 15 una retta costante.

   Se deve essere definita una parabola viene fissata sull'intervallo una parabola particolare che chiameremo "parabola standard". Essa passa per i punti $(u - es, 0)$, $(u + ed, 0)$ e $(((u + ed) - (u - es))/2, 1)$. Poi vengono calcolate le quantità, qa, qb e qc, corrispondenti alle percentuali pa, pb, e pc rispetto ai parametri della parabola standard. Cioè:

$$qa = \frac{a(pa)}{100}$$

$$qb = \frac{b(pb)}{100}$$

$$qc = \frac{c(pc)}{100}.$$

   Tre numeri reali vengono estratti a caso rispettivamente in $[-qa, qa]$, $[-qb, qb]$, e $[-qc, qc]$. Essi vengono sommati ai parametri a, b, e c della parabola standard dando origine ad una nuova parabola. A questo punto, bisogna ricordare che il risolutore è modellato per risolvere problemi di vincoli con preferenze basati sul semianello S_{FCSP}. Questo significa che tutte le preferenze devono essere comprese tra 0 ed 1. La parabola

ottenuta a questo punto, potrebbe non soddisfare questa richiesta. Ci sono alcuni casi, infatti, in cui è incompatibile, se assume solo valori negativi nell'intervallo, o troppo permissiva, se assume valori solo maggiori di 1. In questi due casi viene scartata, e si genera un'altra parabola con lo stesso procedimento. Negli altri casi [1] invece, vengono spostate a 0 le preferenze negative, ed abbassate a 1 quelle maggiori di 1.

Se viene estratta una retta, vengono pescati due reali tra 0 e 1, chiamiamoli y_1 e y_2. La retta, che sarà la funzione di preferenza per quel vincolo, passerà per i punti $(u - es, y_1)$ e $(u + ed, y_2)$. Nel caso venga estratta una costante, invece sarà sufficiente pescare un solo valore tra $[0, 1]$, y_1, e fissare la retta costante per quel valore, come funzione di preferenza. Con questo passo abbiamo finito di definire tutto il vincolo. I vincoli di default non vengono specificati dal generatore in quanto sarà il risolutore a definirli.

Facciamo un piccolo esempio. La chiamata con parametri $n = 8$, $r = 50$, $max = 20$, $d = 30\%$, $pa = 20\%$, $pb = 30\%$, e $pc = 40\%$ dà origine ad un problema con $n = 8$ variabili, che avrà una soluzione con valori compresi tra 0 e $r = 50$. Verrà definito il $d = 30\%$ dei 64 vincoli possibili e ciascun intervallo definito avrà al massimo diametro 40 ($max = 20$). Le funzioni di preferenza saranno parabole, rette o rette costanti e, nel caso siano parabole, esse saranno ottenute deformando i parametri a, b, e c della parabola standard rispettivamente del $pa = 20\%$, $pb = 30\%$ e $pc = 40\%$. In questo esempio si può intuire come la nozione di tightness del generatore che abbiamo preso come riferimento venga sostituita implicitamente dai parametri r e max.

5.3 Problemi generati random

In questa sezione verranno presentati i risultati sperimentali del risolutore.

Il procedimento seguito per ciascun esperimento è il seguente. Prima viene generato un STPP tramite il generatore random. Poi l'STPP generato viene scritto in un file. Tale file è passato come input al risolutore che risolve il problema trovando una soluzione ottima. I risultati sperimentali si riferiscono infatti a STPP_SOLVER.

Facciamo alcune osservazioni. Ricordiamo brevemente che i parametri secondo i quali il generatore produce un STPP sono: il numero di variabili n, il dominio per la soluzione di partenza r, l'intero che determina la massima espansione a destra e a sinistra max, la densità d e i parametri di perturbazione delle parabole pa, pb,e pc.

[1] Per altri casi si intendono tutti quelli per cui almeno una porzione dell'intervallo viene mappata in valori tra 0 e 1.

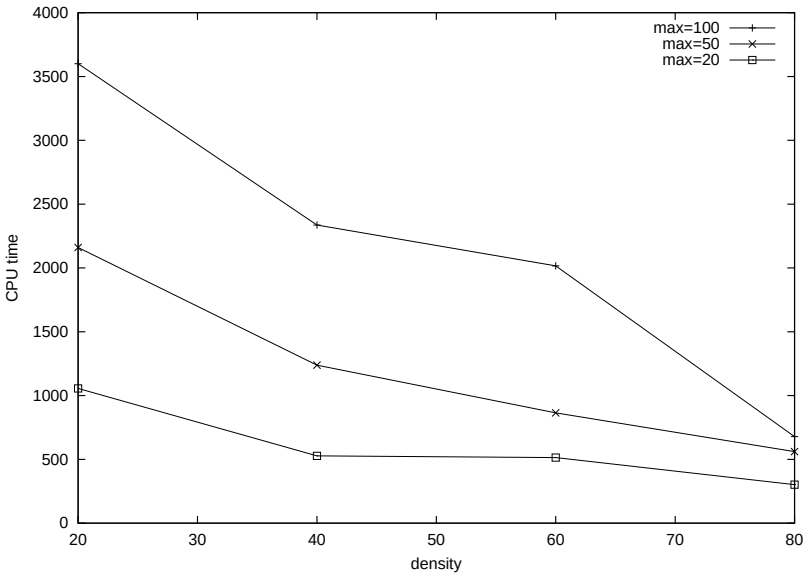

Figura 5.5: Grafico che rappresenta il tempo di soluzione, al variare della desità, per problemi con n=50, r=100, pa=20, pb=20, e pc=30.

Tra questi parametri ne abbiamo scelti 3 da far variare mentre i rimanenti sono stati lasciati fissi. Non abbiamo fatto variare r. Ricordiamo che esso rappresenta il range entro cui vengono scelti i valori da assegnare alle variabili per la soluzione attorno alla quale si costruisce il problema. Infatti il programma non è sensibile al valore effettivo delle variabili. Consideriamo infatti un STPP in cui tutte le soluzioni hanno assegnamenti compresi tra 1 e 100 e consideriamo lo stesso problema con assegnamenti moltiplicati tutti per 100000. Il secondo problema corrisponde ad una traslazione nel futuro del sistema di eventi che il problema rappresenta. Tuttavia è facile vedere come questa traslazione non cambi niente a livello del risolutore [2]. In realtà abbiamo visto alla sezione precedente come r sia in qualche modo connesso alla tightness del problema. Nei nostri esperimenti abbiamo preferito far variare solo un parametro collegato con la tightness : max.

Gli altri parametri che abbiamo mantenuto fissi sono quelli che regolano la distorsione della parabola di partenza. Questi parametri sono collegati alla forma delle parabole e dunque al numero di livelli di preferenza del problema che, come dimostrato al capitolo 3, influiscono sulla complessità del processo risolutivo. Le percentuali che abbiamo scelto tuttavia sono tali da permettere in genere di avere un buon assortimento di valori di preferenza tra 0 e 1. Inoltre la componente random che è stata introdotta nel processo di gene-

[2]Una traslazione provocherebbe qualche problema solo nel caso in cui gli assegnamenti diventassero talmente grandi da non essere rappresentabili dalla macchina.

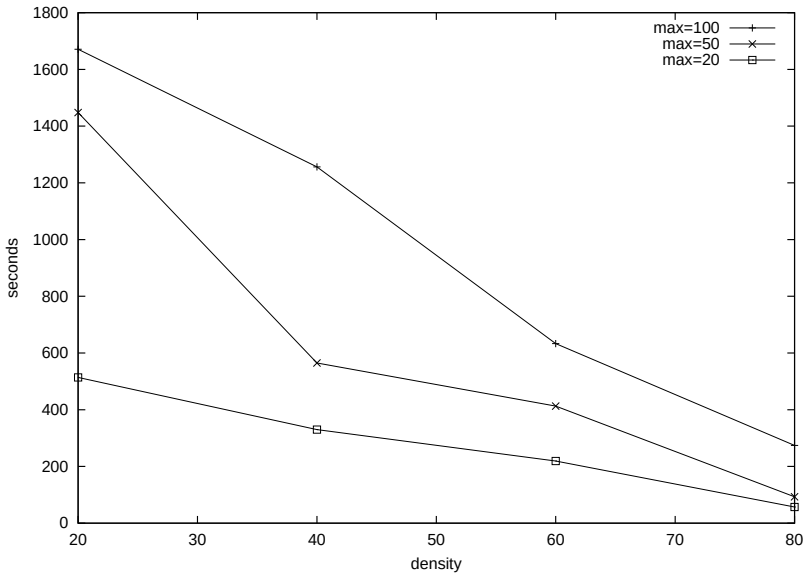

Figura 5.6: Grafico che rappresenta il tempo di soluzione, al variare della desità, per problemi con n=40, r=100, pa=20, pb=20, e pc=30.

razione delle parabole rende difficile una chiara valutazione della dipendenza del numero di livelli di preferenza e le percentuali di perturbazione.

In tutti gli esperimenti, dunque, r è stato posto a 100. Questo significa che il nostro problema avrà una soluzione in cui tutti gli eventi avvengono nelle prime 100 unità di tempo dalla partenza del cronometro. I valori per i parametri pa, pb, e pc sono stati fissati rispettivamente al 20%, 20% e 30%. Ci si potrebbe chiedere perchè una perturbazione più alta della c delle parabole. Il motivo è il seguente: i parametri a e b sono legati al valore del massimo della parabola, mp, dalla seguente relazione:

$$mp = \frac{-b}{2a}.$$

Una percentuale alta di perturbazione di questi parametri porterebbe a generare molte parabole con massimo incompatibile con lo schema fuzzy, parabole che verrebbero tutte scartate allungando il processo di generazione del problema. Il parametro c invece controlla la traslazione orizzontale della parabola e dunque si può sottoporre ad una variazione un po' più forte.

I parametri che abbiamo fatto variare sono stati il numero di variabili, n, la densità dei vincoli definiti, d, e il range di espansione max. Ciascun grafico corrisponde a esperimenti fatti con un determinato numero di variabili: 50, 40, 30, e 20 . Sull'asse x abbiamo invece indicato la densità: 20%, 40%, 60%, e 80%. Sull'asse y sono indicati i secondi impiegati dal risolutore. Tale misura è relativa all'intero processo risolutivo e comprende: l'attesa dell'indicazione

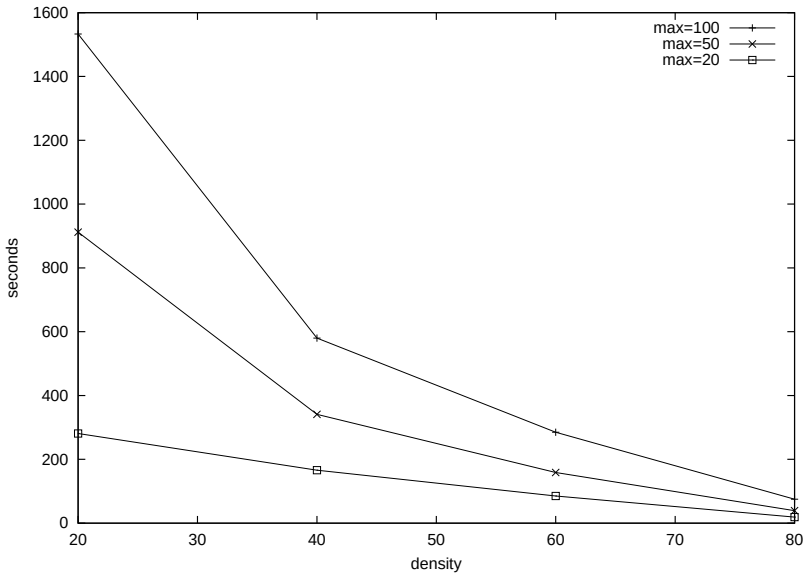

Figura 5.7: Grafico che rappresenta il tempo di soluzione, al variare della densità per problemi con n=30, r=100, pa=20, pb=20, e pc=30.

da parte dell'utente sul tipo di input [3], la lettura del file di input e l'inizializzazione del problema, la vera e propria fase di elaborazione e la stampa dell'output. Ogni grafico è costituito da 3 curve relative a i tre diversi valori di max: 100, 50 e 20. I vincoli di default sono stati posti seguendo quanto detto precedentemente. Essendo r fisso, la loro scelta è dipesa unicamente da max. In conformità con quanto detto sono stati posti a $[-400, 400]$ nei problemi con $max = 100$, a $[-300, 300]$ in quelli con $max = 50$ e a $[-240, 240]$ in quelli con $max = 20$. Ogni punto della curva rappresenta il tempo medio su tre problemi diversi generati dal generatore random con stessi valori per i parametri. A fronte dei risultati ottenuti possiamo trarre alcune conclusioni:

- Il tempo impiegato dal risolutore è direttamente proporzionale al numero di variabili. Questo risulta ovvio se si tiene conto della dipendenza cubica della complessità da questo parametro. Infatti ricordiamo che che STPP_PC-2 considera tutti i vincoli e i vincoli crescono al crescere del numero delle variabili.

- Il tempo impiegato dal risolutore è direttamente proporzionale al massimo grado di espansione dei vincoli, max. Il valore di massima espansione è legato al range che compare nella formula della complessità. In breve, più grande è max, più grandi possono essere gli intervalli dei

[3] Essendo tutti gli esperimenti svolti su problemi generati random, l'input è sempre stato dato da file.

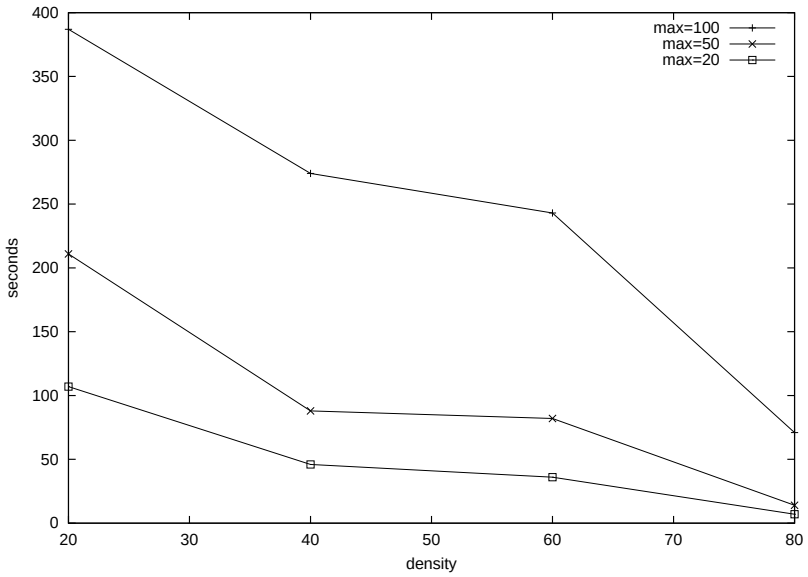

Figura 5.8: Grafico che rappresenta il tempo di soluzione, al variare della densità, per problemi con n=20, r=100, pa=20, pb=20, e pc=30.

vincoli e più tempo impiega l'algoritmo STPP_PC-2. Si può osservare come anche a densità basse questo influisca sui tempi. Questo è dovuto alla diversa dimensione di vincoli di default.

- Il tempo impiegato è inversamente proporzionale alla densità. Infatti più alti sono i valori di d più veloce va il programma. Questo fatto è semplice da spiegare. Più vincoli definisce il generatore, più vincoli avranno intervalli piccoli rispetto a quelli di default e meno tempo sarà necessario per imporre la consistenza locale.

- Più basso è max e meno influisce la densità. La presenza di intervalli molto piccoli accorcia comunque il lavoro di STPP_PC-2 . Infatti quando vengono fatti i rilassamenti c'è una probabilità più alta di troncamenti massicci sugli intervalli più grossi. Questo fatto accorcia il numero di passi necessari per imporre la consistenza locale.

Il risolutore si è rivelato abbastanza lento, soprattutto se si pensa che in futuro prevediamo di gestire problemi NASA con 300 variabili. Sicuramente questo è dovuto alla rappresentazione e alla elaborazione che porta alla consistenza locale entrambe punto a punto. Nella conclusione è descritta una idea in fase di realizzazione per un risolutore più veloce. Tuttavia, al contrario di questo, potrà gestire solo certi tipi di funzione.

Capitolo 6

Apprendimento automatico di STPPs

In questo Capitolo viene descritto il modulo di apprendimento automatico.

La prima parte è mostra come la tecnica di discesa di gradiente sia stata adattata all'apprendimento di STPPs.

Poi viene descritto l'algoritmo vero e proprio, con una spiegazione dettagliata di ogni passo che esso prevede.

Infine vengono specificate le scelte implementative che sono state fatte durante lo sviluppo del modulo.

6.1 Teoria e proprietà

L'applicabilità della teoria dell'apprendimento automatico agli STPPs è stata indagata per ovviare alla evidente difficoltà di determinare le funzioni locali in problemi con molti vincoli o con conoscenze a livello locale confuse o imprecise. Inoltre, può essere più facile dare una preferenza globale alle soluzioni. Il modulo di apprendimento automatico ha lo scopo di indurre da questa preferenza globale le funzioni di preferenza locale. L'algoritmo si fonda sul concetto di apprendimento sulla base di esempi. Infatti fanno parte del processo con cui si realizza l'apprendimento due insiemi. Il primo insieme è detto *training set*, ed indicato con Tr. Esso è costituito da un insieme di soluzioni con il loro valore di preferenza desiderato. Esse costituiscono gli esempi da cui il modulo dovrà "imparare". Il secondo insieme, detto *test set*, Ts , è sempre costituito da delle soluzioni con il loro valore di preferenza. Esso deve essere disgiunto dal primo, in quanto dovrà fornire una stima del successo dell'apprendimento. L'algoritmo dovrà trarre delle conclusioni dagli esempi forniti e applicarle su nuovi esempi mai visti.

In linea generale il procedimento che esegue il modulo di apprendimento è il seguente: ad ogni iterazione esso legge un esempio dal training set, e formu-

la una ipotesi sulla preferenza da assegnare a quella soluzione. Poi calcola di quanto ha sbagliato ed i parametri, secondo i quali il modulo formula le proprie preferenze, vengono cambiati in modo da ridurre l'errore. Quando l'errore commesso scende sotto un certo livello, allora si termina l'apprendimento e si verifica la bontà dell'apprendimento sul test set. L'errore commesso sul test sarà il voto, ma, attenzione, sarà tanto migliore quanto è più piccolo.

La tecnica scelta è quella della discesa di gradiente [1]. Ricordiamo che la discesa di gradiente viene effettuata sulla funzione errore, che stima quanto buona è l'ipotesi correntemente formulata dal processo di apprendimento. La prima cosa da fare, dunque, è scegliere una tale funzione.

La funzione errore che adottiamo è l'**errore quadratico cumulativo**:

$$E_{cum} = \frac{1}{2} \sum_{s \in Tr} (t(s) - h(s))^2 \tag{6.1}$$

In (4.1) $t(s)$ rappresenta la preferenza vera della soluzione s del training set, mentre $h(s)$ è la preferenza ipotizzata dal modulo per la stessa soluzione. Come si vede la somma in questo caso è estesa a tutte le soluzioni del training set.

Ma, come fa il modulo di apprendimento automatico a formulare la sua preferenza per s? Abbiamo già osservato come quello che si vuole imparare siano le funzioni di preferenza locali. Al modulo, dunque dovranno essere dati innanzi tutto i vincoli hard del problema. Sulla base di essi verranno inizializzate delle funzioni tramite le quali verrà formulata l'ipotesi iniziale. Ad ogni passo dell'apprendimento queste funzioni verranno deformate in modo che il loro contributo locale dia origine globalmente a valori simili alle preferenze desiderate [2]. Appare ovvio che le funzioni in questione debbano essere parametrizzate, di modo che l'algoritmo possa deformarle agendo su tali parametri. L'entità delle correzioni, che farà ad ogni passo l'algoritmo, dipenderà direttamente dalla derivata della funzione errore rispetto a quel parametro.

Quanto detto serve a giustificare la scelta che è stata fatta per le funzioni di preferenza. Infatti, nel nostro modulo, esse saranno funzioni quadratiche semi-convesse: parabole convesse, rette etc., della forma:

$$y = ax^2 + bx + c \text{ con } a \leq 0 \tag{6.2}$$

In questo modo, ciascuna funzione può essere descritta dai tre parametri a, b, c. Possiamo dunque concludere che $h(s)$ è la preferenza che viene assegnata alla soluzione dalla rete di vincoli temporali semplici corrente, cioè con le funzioni ad un determinato stadio di deformazione.

[1] Una descrizione di questa tecnica si trova alla Sezione 2.4 .

[2] Le preferenze desiderate sono quelle contenute nel training set.

Indichiamo ora alcune notazioni che useremo: il numero di variabili coinvolte nel nostro problema sarà indicato con n. Dunque avremo n^2 vincoli, di cui solo $\nu = (\frac{(n-1)n}{2}) - n$ sono significativi [3]. Data la soluzione s, indicheremo con s_i la proiezione di s sul vincolo i-simo, e naturalmente i andrà da 1 a ν. In dettaglio, se l'i-simo vincolo è quello tra le variabili X_h e X_k, e s assegna a tali variabili rispettivamente i valori v_h e v_k, allora si ha che $s_i = v_k - v_h$. Ricordiamo la formula che dà la preferenza globale:

$$pref(s) = \times\{f_{hk}(v_k - v_h)|h, k = 1, \cdots, n,\ h < k\}$$

Con le nuove notazioni la formula diventa:

$$h(s) = \prod_{i=1}^{\nu}\{f_i(s_i)\}, \tag{6.3}$$

dove f_i è la funzione di preferenza dell'i-simo vincolo. Teniamo ora conto del fatto che il semianello è S_{FCSP} e che il suo operatore moltiplicativo è min. Si ottiene:

$$h(s) = min_{i=1,\cdots,\nu}\{f_i(s_i)\}. \tag{6.4}$$

Ricordando che le funzioni che utilizziamo sono quadratiche semi-convesse possiamo esprimere f_i nel seguente modo:

$$f_i(s_i) = a_i s_i^2 + b_i s_i + c_i$$

sostituendo nella 6.4, otteniamo:

$$h(s) = min_{i=1,\cdots,\nu}\{a_i s_i^2 + b_i s_i + c_i\}. \tag{6.5}$$

Sostituiamo l'espressione ottenuta per $h(s)$ nell'errore quadratico cumulativo, 6.1:

$$E_{cum} = \frac{1}{2}\sum_{s\in Tr}(t(s) - [min_{i=1,\cdots,\nu}\{a_i s_i^2 + b_i s_i + c_i\}])^2. \tag{6.6}$$

Fermiamoci ora, per fare delle considerazioni su come effettuare e rendere possibile la discesa di gradiente su questo errore. La discesa di gradiente verrà fatta in modo *stocastico*, o *incrementale*.

Come già visto nel capitolo introduttivo[4] questo significa che l'errore verrà calcolato dopo aver considerato un esempio, i.e. una soluzione del training set, ed anche gli aggiornamenti verranno fatti con questo criterio. La sommatoria su tutte le soluzioni del training set va, dunque, tolta e l'errore diventa:

$$E = \frac{1}{2}(t(s) - [min_{i=1,\cdots,\nu}\{a_i s_i^2 + b_i s_i + c_i\}])^2 \tag{6.7}$$

[3] I rimanenti vincoli sono reciproci, come abbiamo osservato parlando del risolutore, e quindi ridondanti per la computazione della preferenza globale.

[4] Ci riferiamo al paragrafo sull'apprendimento automatico. In esso sono trattati i pro e i contro di questa tecnica di discesa di gradiente.

Il nostro scopo è quello di derivare la funzione errore rispetto ai parametri del problema e poi di modificare gli stessi in modo tale da far scendere l'errore, quindi nella direzione opposta rispetto a quella a cui punta il gradiente. Si osservi come nella 6.7, l'errore è già espresso in funzione dei parametri del nostro problema che, ricordiamo, sono $\{a_1, b_1, c_1, \cdots, a_\nu, b_\nu, c_\nu\}$. In altre parole le dipendenze funzionali dell'errore nella 6.7 è esprimibile come:

$$E = E(a_1, b_1, c_1, \cdots, a_\nu, b_\nu, c_\nu).$$

Ha dunque senso calcolare le derivate $\frac{\partial E}{\partial a_i}$, $\frac{\partial E}{\partial b_i}$, e $\frac{\partial E}{\partial c_i}$. Tuttavia vi è un problema: la discontinuità della funzione *min*. Per ovviare a ciò utilizziamo l'approssimazione continua di *min* descritta in [15], e ripresa nel nostro capitolo introduttivo Sezione 2.4 . Essa è

$$min_{i=1,\cdots,\nu}(\alpha_i) \simeq min_\nu^\beta(\alpha_i) = -\frac{1}{\beta}\ln(\frac{1}{\nu}\sum_{i=1}^{\nu} \mathrm{e}^{-\beta\alpha_i}).$$

Ricordiamo che $\beta \geq 0$ è un parametro che dice quanto l'approssimazione sia buona. Il suo ruolo, qui, verrà discusso in seguito. Nel nostro caso $\alpha_i = a_i s_i^2 + b_i s_i + c_i$ e sostituendo nella 6.7 otteniamo:

$$E = \frac{1}{2}(t(s) - [-\frac{1}{\beta}\ln(\frac{1}{\nu}\sum_{i=1}^{\nu} \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)})])^2. \tag{6.8}$$

Annullandosi i due segni negativi si ottiene:

$$E = \frac{1}{2}(t(s) + \frac{1}{\beta}\ln(\frac{1}{\nu}\sum_{i=1}^{\nu} \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)})])^2. \tag{6.9}$$

Una volta calcolato l'errore l'algoritmo dovrà aggiornare i parametri nel seguente modo:

$$\tilde{a_i} = a_i + \Delta a_i \text{ con } \Delta a_i = -\eta\frac{\partial E}{\partial a_i} \tag{6.10}$$

$$\tilde{b_i} = a_i + \Delta b_i \text{ con } \Delta b_i = -\eta\frac{\partial E}{\partial b_i} \tag{6.11}$$

$$\tilde{c_i} = c_i + \Delta c_i \text{ con } \Delta c_i = -\eta\frac{\partial E}{\partial c_i} \tag{6.12}$$

I valori a sinistra, contrassegnati con una tilde, sono i valori dopo l'assegnamento; quelli a destra sono i valori prima dell'assegnamento. Il parametro η, che compare in queste formule è detto passo di discesa (o apprendimento) e regola quanto deve pesare ogni aggiornamento. Calcoliamo $\frac{\partial E}{\partial a_i}$:

$$\frac{\partial E}{\partial a_i} = (t(s) - h(s))(-(\frac{\partial h(s)}{\partial a_i})). \tag{6.13}$$

Come si può notare dalla 6.5 è solo $h(s)$, infatti, a dipendere da $\{a_1, b_1, c_1, \cdots, a_\nu, b_\nu, c_\nu\}$ mentre $t(s)$ è totalmente indipendente da questi parametri. Ora, $h(s)$ espressa con la versione continua della funzione min è:

$$h(s) = -\frac{1}{\beta}\ln(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)})$$

e quindi :

$$414)\frac{\partial h(s)}{\partial a_i} = -\frac{1}{\beta}\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)})}\frac{\partial}{\partial a_i}(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)}) \tag{6.14}$$

dove:

$$\frac{\partial}{\partial a_i}(\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)}) = \frac{\partial}{\partial a_i}(\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2)}\ \mathrm{e}^{-\beta(b_j s_j)}\ \mathrm{e}^{-\beta(c_j)}) \tag{6.15}$$

Usando la linearità dell'operatore di derivata si ottiene:

$$\frac{\partial}{\partial a_i}(\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2)}\ \mathrm{e}^{-\beta(b_j s_j)}\ \mathrm{e}^{-\beta(c_j)}) = (\sum_{j=1}^{\nu}(\frac{\partial}{\partial a_i} \mathrm{e}^{-\beta(a_j s_j^2)})\ \mathrm{e}^{-\beta(b_j s_j)}\ \mathrm{e}^{-\beta(c_j)}) \tag{6.16}$$

Si osservi ora che $(\frac{\partial}{\partial a_i} \mathrm{e}^{-\beta(a_j s_j^2)}) \neq 0$ solo quando $i = j$ ed, in tal caso, si ha :

$$\sum_{j=1}^{\nu}(\frac{\partial}{\partial a_i} \mathrm{e}^{-\beta(a_j s_j^2)})\ \mathrm{e}^{-\beta(b_j s_j)}\ \mathrm{e}^{-\beta(c_j)} = (-\beta s_i^2\ \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)}) \tag{6.17}$$

Questo ci permette di concludere che:

$$\frac{\partial}{\partial a_i}(\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)}) = (-\beta s_i^2\ \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)}) = \frac{-\beta}{\nu}(s_i^2\ \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)}) \tag{6.18}$$

Sostituendo nella 6.14414), otteniamo:

$$\frac{\partial h(s)}{\partial a_i} = \frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)})}(s_i^2\ \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)}) \tag{6.19}$$

da cui:

$$\frac{\partial E}{\partial a_i} = (t(s) - h(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)})}(-s_i^2\ \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)})) \tag{6.20}$$

Per ottenere le derivate rispetto agli altri parametri il calcolo è lo stesso fino alla 6.17, che nel caso le derivate vengano fatte rispetto a b_i e c_i diventa:

$$\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2)}\left(\frac{\partial}{\partial b_i}\,\mathrm{e}^{-\beta(b_j s_j)}\right)\mathrm{e}^{-\beta(c_j)} = \left(-\beta s_i\,\mathrm{e}^{-\beta(a_i s_i^2+b_i s_i+c_i)}\right) \tag{6.21}$$

$$\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2)}\;\mathrm{e}^{-\beta(b_j s_j)}\left(\frac{\partial}{\partial c_i}\,\mathrm{e}^{-\beta(c_j)}\right) = \left(-\beta\,\mathrm{e}^{-\beta(a_i s_i^2+b_i s_i+c_i)}\right) \tag{6.22}$$

Possiamo dunque concludere che:

$$\frac{\partial E}{\partial a_i} = (t(s)-h(s))\left(\frac{1}{\left(\frac{1}{\nu}\sum_{j=1}^{\nu}\,\mathrm{e}^{-\beta(a_j s_j^2+b_j s_j+c_j)}\right)}\left(-s_i^2\,\mathrm{e}^{-\beta(a_i s_i^2+b_i s_i+c_i)}\right)\right) \tag{6.23}$$

$$\frac{\partial E}{\partial b_i} = (t(s)-h(s))\left(\frac{1}{\left(\frac{1}{\nu}\sum_{j=1}^{\nu}\,\mathrm{e}^{-\beta(a_j s_j^2+b_j s_j+c_j)}\right)}\left(-s_i\,\mathrm{e}^{-\beta(a_i s_i^2+b_i s_i+c_i)}\right)\right) \tag{6.24}$$

$$\frac{\partial E}{\partial c_i} = (t(s)-h(s))\left(\frac{1}{\left(\frac{1}{\nu}\sum_{j=1}^{\nu}\,\mathrm{e}^{-\beta(a_j s_j^2+b_j s_j+c_j)}\right)}\left(-\,\mathrm{e}^{-\beta(a_i s_i^2+b_i s_i+c_i)}\right)\right) \tag{6.25}$$

Ricapitolando gli aggiornamenti per i tre parametri saranno:

$$\tilde{a_i} = a_i - \eta\left[(t(s)-h(s))\left(\frac{1}{\left(\frac{1}{\nu}\sum_{j=1}^{\nu}\,\mathrm{e}^{-\beta(a_j s_j^2+b_j s_j+c_j)}\right)}\left(-s_i^2\,\mathrm{e}^{-\beta(a_i s_i^2+b_i s_i+c_i)}\right)\right)\right] \tag{6.26}$$

$$\tilde{b_i} = b_i - \eta\left[(t(s)-h(s))\left(\frac{1}{\left(\frac{1}{\nu}\sum_{j=1}^{\nu}\,\mathrm{e}^{-\beta(a_j s_j^2+b_j s_j+c_j)}\right)}\left(-s_i\,\mathrm{e}^{-\beta(a_i s_i^2+b_i s_i+c_i)}\right)\right)\right] \tag{6.27}$$

$$\tilde{c_i} = c_i - \eta\left[(t(s)-h(s))\left(\frac{1}{\left(\frac{1}{\nu}\sum_{j=1}^{\nu}\,\mathrm{e}^{-\beta(a_j s_j^2+b_j s_j+c_j)}\right)}\left(-\,\mathrm{e}^{-\beta(a_i s_i^2+b_i s_i+c_i)}\right)\right)\right] \tag{6.28}$$

6.2 Algoritmo di Apprendimento

Diamo prima una descrizione informale di quello che fa l'algoritmo. Innanzi tutto deve inizializzare l'STPP da cui partirà l'apprendimento. Questo viene fatto leggendo da un file i vincoli hard del problema da affrontare [5].

[5] Ricordiamo che i vincoli hard si suppone siano sempre noti.

Fatto questo esso inizializza le funzioni di preferenza. L'inizializzazione avviene sempre prendendo come funzioni delle parabole, anche se, per costruire l'STPP di base [6], ne verranno utilizzate due classi in particolare, come verrà descritto tra poco.

Una volta che sia stato inizializzato il problema, comincia la scansione delle soluzioni nel training set. Per ogni soluzione viene calcolato l'errore quadratico non cumulativo, che indicheremo con E, in quanto lo scopo è di fare una discesa di gradiente stocastica. Vengono poi calcolati gli aggiornamenti per ogni parametro di ogni vincolo, propagando l'esperienza ottenuta dall'esame di quella soluzione.

È utile, ora, indicare come il modulo di apprendimento automatico gestisce il fatto che le funzioni debbanno essere (1) semi-convesse, (2) avere l'immagine contenuta in $[0,1]$ per rientrare nello schema fuzzy. In quanto verrà detto è utile ricordare che sia η che β sono sempre positivi.

(1) Osserviamo che gli unici parametri coinvolti nel garantire la semi-convessità sono quelli di tipo a. È noto infatti come una parabola, $y = ax^2 + bx + c$, sia convessa se $a \leq 0$. L'algoritmo provvede a mantenere la convessità delle parabole durante tutto il processo, forzando i parametri a a mantenere valori minori o uguali a 0. Infatti prima di assegnare il nuovo valore, calcolato come indicato dalla 6.26, viene fatto un controllo sul suo segno. Se esso risulta essere strettamente positivo, il nuovo valore per a sarà zero.

Questo metodo, che corrisponde ad un metodo proiettivo, si è rivelato in pratica efficace. Il fatto che sia abbastanza coercitivo diventa evidente se si osserva la 6.26:

$$\tilde{a_i} = a_i - \eta[(t(s) - h(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)})}(-s_i^2 \ \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)}))]$$

Supponiamo che essa rappresenti il primo assegnamento che farebbe diventare $\tilde{a_i}$ positivo. Possiamo dunque supporre che a_i sia negativo. Affinché il membro di destra diventi positivo dovrà essere:

$$-\eta[(t(s) - h(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)})}(-s_i^2 \ \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)}))] \geq 0 \quad (6.29)$$

ovvero:

$$\eta[(t(s) - h(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)})}(s_i^2 \ \mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)}))] \geq 0 \quad (6.30)$$

e questo avviene solo se tutti i fattori sono maggiori di 0. Questo significa che l'errore commesso $(t(s) - h(s))$ è positivo e dunque, l'errore è stato commesso per difetto, $t(s) \geq h(s)$. Non solo, l'errore commesso è sufficiente a

[6] Con STPP di base intendiamo quello da cui parte l'apprendimento, ovvero quello caricato quando parte la prima iterazione del programma.

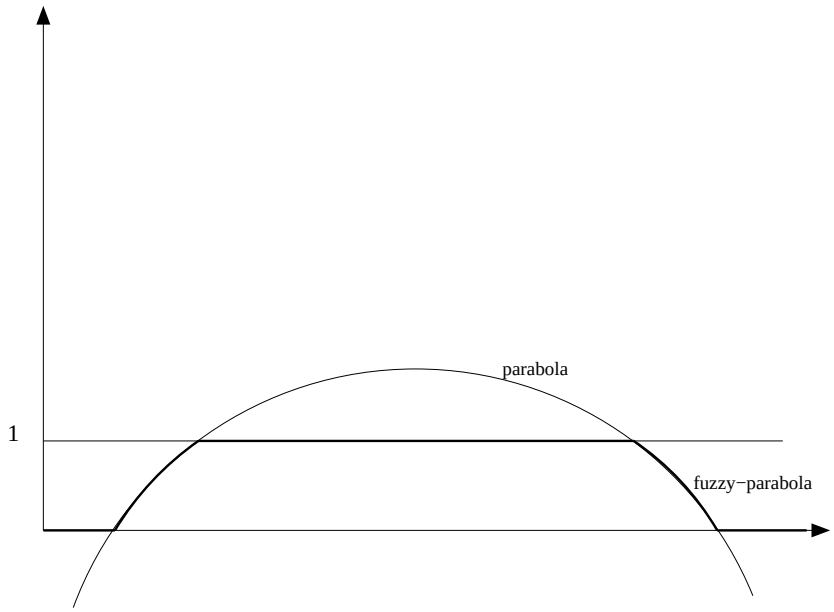

Figura 6.1: In figura è rappresentata una parabola e in neretto la fuzzy-parabola che le corrisponde.

far diventare il secondo addendo della 6.26 più grande, in valore assoluto del valore di a_i. Bene, imporre $\tilde{a_i} = 0$, equivale a considerare l'errore un po' più piccolo sufficiente, cioè solo a rendere il secondo addendo pari ad a_i e non a superarlo.

(2) Far rientrare le funzioni nello schema fuzzy, durante tutto l'apprendimento è una questione più delicata. Definiamo prima l'idea di $fuzzy - parabola$, o semplicemente *parabola troncata*. Essa è una funzione semi-convessa ricavata da una parabola semi-convessa. In pratica avrà gli stessi valori della parabola, se essi sono compresi tra 0 e 1, mentre tutt le immagini maggiori di 1 verranno abbassate a 1 e tutte le immagini minori di 0 verranno alzate a 0.

Questo tipo di funzioni ha il pregio di rientrare nello schema fuzzy ma il difetto di essere discontinuo. Per l'implementazione del modulo sono state studiate tre opzioni:

- utilizzare fuzzy-parabole e le loro derivate. Riconsideriamo la seguente derivata:

$$\frac{\partial}{\partial a_i}(\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)}).$$

  Per tutti gli s_j tali che $a_j s_j^2 + b_j s_j + c_j \geq 1$ o $a_j s_j^2 + b_j s_j + c_j \leq 0$ tale derivata si annulla e non solo quando viene fatta rispetto ad a_i ma anche rispetto a b_i e c_i. Un ulteriore problema è quello della discontinuità

delle fuzzy-parabole negli estremi dei troncamenti. Alla luce di queste considerazioni è questa opzione è stata scartata.

- Un prototipo del modulo di apprendimento è stato implementato sostituendo alle parabole le fuzzy-parabole negli aggiornamenti calcolati con le derivate delle parabole stesse. I risultati sono stati soddisfacenti su alcuni problemi generati random. Ci si è comunque chiesti cosa comportassero questi troncamenti, a 1 e a 0 rispettivamente, a livello della discesa di gradiente.

  Prima di entrare nel dettaglio ridefiniamo alcune notazioni. Indicheremo le parabole $ax^2 + bx + c$ con $f(x)$ e quindi $a_i x_i^2 + b_i x_i + c_i$ sarà $f_i(x_i)$.

  1. Consideriamo il primo tipo di troncamento: $f(x) \geq 1$ viene trasformato in $f(x) = 1$. Prendiamo in considerazione la 6.23, usando le nuove notazioni:

     $$\frac{\partial E}{\partial a_i} = (t(s) - h(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta f_j(s_j)})}(-s_i^2 \ \mathrm{e}^{-\beta f_i(s_i)}))$$

     Osserviamo che da :

     $$f_k(s_k) > 1 \text{ e } \beta \geq 0$$

     si ottiene che:

     $$e^{-\beta f_k(s_k)} \leq e^{-\beta}$$

     e:

     $$s_k^2 e^{-\beta f_k(s_k)} \leq s_k^2 e^{-\beta}$$

     da cui:

     $$-s_k^2 e^{-\beta f_k(s_k)} \geq -s_k^2 e^{-\beta}$$

     Quindi se noi consideriamo la quantità:

     $$(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta f_j(s_j)})}(-s_i^2 \ \mathrm{e}^{-\beta f_i(s_i)}))$$

     essa diventerebbe:

     $$(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta})}(-s_i^2 \ \mathrm{e}^{-\beta}))$$

     e quindi si avrebbe una diminuzione del numeratore, ed un accrescimento del denominatore, che porterebbe ad una complessiva diminuzione del fattore, non mutandone però il segno.

Consideriamo ora il primo fattore $(t(s)-h(s))$. Intanto ricordiamo che per ipotesi le preferenze nel training set sono corrette e dunque, comprese tra 0 e 1. Osserviamo ora come questo troncamento comporti una modifica di $h(s)$ solo nel caso in cui $h(s)$ sia originariamente maggiore di 1. Questo significa che quando sono state considerate tutte le proiezioni della soluzione sui vincoli e i loro rispettivi valori di preferenza, si è ottenuto, prendendo il minimo, un valore maggiore di 1. Quindi tutte queste proiezioni venivano mappate dalle rispettive funzioni in valori maggiori di 1. Questo implica, essendo $t(s) \leq 1$ che si aveva $(t(s) - h(s) \leq 0$. Il troncamento comporta che tutti valori associati alle proiezioni vengano portati a 1 e dunque si ha $h_t(s) = 1$ dove con $h_t(s)$ indichiamo l'ipotesi formulata con le parabole troncate. Da questo si conclude che $(t(s) - h(s))$ può solo calare in valore assoluto operando un troncamento ma non cambiare di segno.

Nel caso in cui $h(s)$ fosse stato già minore o uguale a 1, poichè questo significa che almeno una delle proiezioni veniva mappata dalla rispettiva funzione in un valore minore di 1, il troncamento a 1 non ha alcun effetto. Questo valore di preferenza non viene infatti toccato dal troncamento e rimane comunque minore anche di tutti i valori troncati. Da ciò si può concludere che in questo caso $(t(s)-h(s))$ non viene influenzato dall'operazione in questione nè in modulo nè in segno.

Consideriamo ora, cosa accade alle derivate dei parametri di tipo b. La formula a cui rifarsi è la 6.24, espressa con le nuove notazioni:

$$\frac{\partial E}{\partial b_i} = (t(s) - h(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta f_j(s_j)})}(-s_i \ \mathrm{e}^{-\beta f_i(s_i)}))$$

Si osservi che sempre da $f(x) > 1$ e $\beta \geq 0$ si ottiene:

$$| - s_i e^{-\beta f_i(s_i)}| \leq | - s_i e^{-\beta}|$$

Dunque in:

$$(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta f_j(s_j)})}(-s_i \ \mathrm{e}^{-\beta f_i(s_i)})$$

otteniamo, troncando, un numeratore più grande che potrebbe portare ad un aumento del modulo del secondo fattore. Quanto detto sul primo fattore nel caso dei parametri a, rimane valido.

Consideriamo ora i parametri di tipo c e la loro formula, 6.25 con le nuove notazioni:

$$\frac{\partial E}{\partial c_i} = (t(s) - h(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta f_j(s_j)})}(- \ \mathrm{e}^{-\beta f_i(s_i)}))$$

questa volta, da $f(x) > 1$ e $\beta \geq 0$ otteniamo:

$$-e^{-\beta f(x)} \geq -e^{-\beta}$$

Quindi il numeratore di

$$(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta f_j(s_j)})}(-\mathrm{e}^{-\beta f_i(s_i)}))$$

diventa più piccolo, facendo calare il secondo fattore. Anche in questo caso rimane valido quanto detto su $(t(s)-h(s))$, per i parametri a e b.

Possiamo riassumere la distorsione sulla discesa di gradiente provocata dal troncamento ad 1 dei valori di preferenza maggiori di 1, con i seguenti risultati sulle derivate rispetto ai tre tipi di parametri:

- Il modulo della derivata rispetto ai parametri a diventa minore;
- Il modulo della derivata rispetto ai parametri b diventa maggiore;
- Il modulo della derivata rispetto ai parametri c diventa minore;
- Nessuna derivata viene cambiata di segno;

A livello di aggiornamenti questo comporterebbe:

- I parametri a subirebbero un aggiornamento minore del dovuto;
- I parametri b subirebbero un aggiornamento maggiore del dovuto;
- I parametri c subirebbero un aggiornamento minore del dovuto;
- Gli aggiornamenti rimarrebbero, tuttavia, consistenti con la direzione di discesa di gradiente.

2. Il secondo tipo di modifica, caratteristica di una fuzzy-parabola, è l'innalzamento dei valori minori di 0 a 0. Consideriamone le conseguenze a livello delle derivate $\frac{\partial E}{\partial a}$, $\frac{\partial E}{\partial b}$, e $\frac{\partial E}{\partial c}$. Riconsideriamo la seguente espressione:

$$\frac{\partial E}{\partial a_i} = (t(s) - h(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta f_j(s_j)})}(-(s_i^2\ \mathrm{e}^{-\beta f_i(s_i)})))$$

Da $f(x) < 0$ e $\beta \geq 0$ ricaviamo che:

$$e^{-\beta f(s_i)} > e^0 = 1$$

da cui si ottiene:

$$-s_i^2 e^{-\beta f(s_i)} \leq -s_i^2.$$

Questo significa che in:

$$(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta f_j(s_j)})}(-s_i^2 \ \mathrm{e}^{-\beta f_i(s_i)}))$$

il numeratore cresce mentre il denominatore decresce, rendendo questo fattore più grande in modulo. Vediamo cosa si può dire del primo fattore $(t(s) - h(s))$. Quanto verrà detto varrà anche per le derivate fatte rispetto gli altri parametri.

Osserviamo che $h(s)$ viene modificato dall'innalzamento dei valori minori di 0 a 0 solo se esso era minore di 0. In questo caso, almeno una proiezione della soluzione veniva mappata dalla funzione ad un valore minore di 0. L'innalzamento comporterà che $h(s)$ diventi 0. Se ciò si verifica, tuttavia, significa che l'errore commesso dal modulo di apprendimento era per difetto. Infatti, $t(s)$ è sempre compreso tra 0 e 1. Dunque possiamo concludere che $(t(s) - h(s))$ può solo diminuire in modulo ma non cambiare di segno.

Vediamo cosa accade per la derivata rispetto ai parametri di tipo b. la formula ricordiamo è:

$$\frac{\partial E}{\partial b_i} = (t(s) - h(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta f_j(s_j)})}(-s_i \ \mathrm{e}^{-\beta f_i(s_i)}))$$

Da $f(x) < 0$ e $\beta \geq 0$ si ricava che:

$$|-s_i e^{-\beta f_i(s_i)}| \geq |-x|$$

il numeratore di:

$$(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta f_j(s_j)})}(-s_i \ \mathrm{e}^{-\beta f_i(s_i)}))$$

decresce portando ad un'eventuale diminuzione del modulo di questo fattore.

Per quanto riguarda $\frac{\partial E}{\partial c}$, la formula a cui si fa riferimento è:

$$\frac{\partial E}{\partial c_i} = (t(s) - h(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta f_j(s_j)})}(-1 \ \mathrm{e}^{-\beta f_i(s_i)}))$$

Osservando che $f(x) < 0$ e $\beta \geq 0$ implica:

$$-e^{-\beta f(x)} < -1$$

si ottiene in

$$\left(\frac{1}{\left(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta f_j(s_j)}\right)}\left(-\ \mathrm{e}^{-\beta f_i(s_i)}\right)\right)$$

un numeratore più grande in modulo e quindi un fattore più grande.

Possiamo dunque concludere che la distorsione provocata dall'innalzamento dei valori negativi a 0 è dovuta alle seguenti conseguenze sulle derivate dell'errore (quadratico non cumulativo) rispetto ai tre parametri:

- Il modulo della derivata rispetto ai parametri a diventa maggiore;
- Il modulo della derivata rispetto ai parametri b diventa minore;
- Il modulo della derivata rispetto ai parametri c diventa maggiore;
- Nessuna derivata viene cambiata di segno;

Questo si ripercuote a livello degli aggiornamenti nel seguente modo:

- I parametri a subirebbero un aggiornamento maggiore del dovuto;
- I parametri b subirebbero un aggiornamento minore del dovuto;
- I parametri c subirebbero un aggiornamento maggiore del dovuto;
- Gli aggiornamenti rimarrebbero, tuttavia, consistenti con la direzione di discesa di gradiente.

Si può dunque, concludere che usare fuzzy-parabole durante tutta la fase di apprendimento incide in modo non trascurabile sulla discesa di gradiente, e quindi anche questa opzione è stata scartata

- Rimane tuttavia il problema di produrre un STPP nello schema fuzzy. Per questo è stata adottata la seguente politica: durante l'apprendimento le parabole vengono lasciate "libere", cioè non vengono troncate. Vengono però calcolati due gruppi di errori uno considerando le parabole e l'altro considerando le fuzzy-parabole. Verranno dunque calcolati l' errore quadratico cumulativo, E_c, l'errore assoluto massimo E_{max} e l'errore medio assoluto E_{med} con le parabole non modificate.

  Poi, questi errori verranno ricalcolati, e verranno indicati con E_c^f, E_{max}^f ed E_{med}^f, considerando le fuzzy-parabole. Il criterio di fine dell'apprendimento sarà calcolato proprio in base a E_{med}^f, nel modo che descriveremo tra breve.

Una volta terminato l'apprendimento verrà dato come STPP finale quello con fuzzy-parabole.

Un ultimo argomento va chiarito, prima di proseguire la descrizione informale dell'algoritmo, esso riguarda il modo in cui l'algoritmo calcola $h(s)$. Abbiamo visto come una versione continua della funzione min sia necessaria per poter derivare l'errore rispetto ai parametri. Nella Sezione 2.4 è stato descritta l'approssimazione continua della funzione min. Tuttavia, il carattere approssimativo della funzione che noi sostituiamo a min, si propaga inevitabilmente alle derivate rispetto ai parametri. Abbiamo, dunque, due possibilità per il calcolo di $h(s)$, che osserviamo non comporta alcuna operazione di derivata. Una è utilizzare la versione continua, approssimata della funzione min, l'altra è quella di usare il vero min, quello discontinuo. A favore della prima è che sembrerebbe più corretto utilizzare lo stesso "tipo" di min ovunque nelle equazioni 6.23, 6.24 e 6.25. Invece, riflettendo si vede come utilizzare il min non approssimato nel calcolo di $h(s)$ può in qualche modo correggere l'effetto sulle derivate dovuto all'approssimazione fatta sulla parte a cui andava effettivamente applicato l'operatore di derivata. In altre parole guardiamo ad esempio l'equazione:

$$\frac{\partial E}{\partial c_i} = (t(s) - h(s))(\frac{1}{(\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta f_j(s_j)})}(-\mathrm{e}^{-\beta f_i(s_i)}))$$

al secondo membro abbiamo due fattori. Nel secondo fattore è presente l'approssimazione di min, nel primo sta a noi decidere. Se utilizziamo l'approssimazione l'errore $(t(s) - h(s))$ viene ulteriormente falsato dal fatto che non prendiamo proprio il min ma una sua approssimazione. Invece prendendo il min discontinuo otteniamo che l'errore $(t(s) - h(s))$ rappresenta effettivamente l'errore commesso nella valutazione di s, mentre il secondo fattore viene attenuato dalla presenza dell'approssimazione. La nostra scelta è stata dunque, quella di impiegare per il calcolo di $h(s)$ la versione discontinua della funzione min in quanto, in tal modo, è possibile controbilanciare in parte l'effetto dell'approssimazione negli aggiornamenti.

Ritornando all'algoritmo, a questo punto l'STPP di base è stato caricato e l'apprendimento può cominciare. Vengono analizzate una alla volta le soluzioni nel training set, ad ogni campionamento dal training set corrisponde un aggiornamento dei parametri delle funzioni. Dopo che tutti gli esempi del training set sono stati esaminati, vengono calcolati E_c, E_{max} ,E_{med}, E_c^f, E_{max}^f ,E_{med}^f. Inoltre viene calcolato a quale percentuale del miglioramento su E_{med}^f precedente corrisponde l'ultimo aggiornamento di E_{med}^f, $PLI_{E_{med}}^f$ [7]. Se per un certo numero di iterazioni, $maxit$, non si ottiene un certo miglioramento,

[7] $PLI_{E_{med}}^f$ dall'inglese percentage of last improvement.

tale cioè da superare una certa soglia, $Thres$, ci si ferma. A questo punto le parabole nel problema finale prodotto dal modulo vengono troncate e si procede alla verifica sul test set. All' STPP ottenuto vengono proposti tutti gli esempi nel test set e vengono calcolati gli errori su questo insieme (E_c^{ts}, E_{max}^{ts}, E_{med}^{ts}).

Algoritmo STPP_LEARNING_MODULE

1. **initialize** STPP P ;
2. **do**
3. **for each** $s \in Tr$;
4. **compute** E;
5. UPDATE(P, E, η, β);
6. **end of for;**
7. **compute** E_c, E_{max}, E_{med}, E_c^f, E_{max}^f, E_{med}^f, $PLI_{E_{med}}^f$ on training set;
8. **if** $(PLI_{E_{med}}^f < Thres)$ stopcount++;
9. **if** $(PLI_{E_{med}}^f \geq Thres)$ stopcount=0;
10. **while**(stopcount<maxint);
11. **compute** E_c^{ts}, E_{max}^{ts}, E_{med}^{ts} on test set;
12. **output**;

Figura 6.2: Algoritmo STPP_LEARNING_MODULE.

Descriviamo ora in dettaglio ciascun punto dell'algoritmo:

initialize STPP P

Questa fase prepara l'STPP da cui verrà cominciato l'apprendimento. Possiamo dire che fornisce tutte le informazioni che verranno poi date per scontate. Queste informazioni, o non verranno mai ritrattate, come nel caso dei vincoli hard, o verranno modificate dal processo di apprendimento.

I vincoli hard vengono letti da un file e servono per inizializzare gli intervalli dei vincoli sui quali abbiamo informazioni. Per tutti gli altri vincoli, gli intervalli saranno quelli di default descritti nel capitolo del risolutore. Essi, cioè, saranno abbastanza grandi da non influenzare in nessun modo gli altri vincoli. Osserviamo come questo non sia importante per

STPP_LEARNING_MODULE in sè, in quanto questo algoritmo non modifica mai gli intervalli, ma lo può diventare se in un secondo tempo si vorrà risolvere il problema appreso.

Per completare l'inizializzazione è necessario decidere che funzioni di preferenza usare per ciascun vincolo. Sono state adottate ancora una volta funzioni quadratiche semi-convesse. Si ricorda come questa scelta, sia stata fatta in base alla grande varietà di situazioni che esse riescono a rappresentare e anche per la loro parametrizzazione relativamente semplice. Nel corso degli esperimenti sono stati adottate due tipi di inizializzazioni a parabole. Una è quella a cosidette parabole standard. Per ciascun vincolo viene inizializzata la parabola passante per gli estremi e per il punto con ascissa il punto mediano dell'intervallo e come ordinata 1. Questa inizializzazione si è rivelata soddisfacente sono negli esperimenti toy, cioè fatti su problemi generati dal generatore random. Il motivo è sicuramente legato in parte alla somiglianza tra il processo di inizializzazione a parabole che il generatore utilizza su alcuni vincoli e quello adottato dal modulo di apprendimento.

Un secondo tipo di parabole utilizzate sono state le rette costanti ad 1. Osserviamo come questo sia il sistema più veloce e generale. Infatti adottarlo equivale a far partire l'apprendimento solo da vincoli hard, che , come abbiamo già osservato, nel nostro schema sono rappresentati da intervalli con funzioni di preferenza uguale alla costante 1. Sorprendentemente questo metodo ha dato buoni risultati nei problemi più difficili, cioè con più variabili, e quindi più vincoli e con problemi reali, non toy, in cui si conosce solo la funzione di preferenza globale.

Al termine di questa fase il problema di base da cui partire è caricato e può avere inizio la fase di apprendimento vera e propria.

Nella fase di input viene richiesto all'utente, tramite lo standard input, di inserire il numero di esempi contenuti nel training set e nel test set. Ovviamente questi dati serviranno per assicurare che la scansione dei due insiemi sia completa. Infatti il numero di esempi su cui viene allenato il modulo non è costante. Questa scelta, viene fatta a monte, cioè quando vengono generati i due insiemi, con STPP_RANDOM_SOLVER, e dipenderà da alcune caratteristiche del problema. In, realtà spesso si sono fatti esperimenti su uno stesso problema con training e test set di dimensioni diverse.

A questo punto si entra nel ciclo *while* fondamentale che termina solo quando si ritiene di aver imparato sufficientemente dagli esempi nel training set. In questo ciclo ci sono due fasi: un ciclo *for* che esegue la scansione del training set aggiornando ad ogni soluzione incontrata i parametri delle funzioni , e una fase di verifica del livello di apprendimento raggiunto dopo

una passata del training.

1. *Fase di scansione e aggiornamento.* Essa comprende le righe 3, 4, 5, 6 dell'algoritmo. Per permettere alle soluzioni del training set di essere considerate più e più volte, esse vengono caricate in un array multidimensionale. A ciascuna iterazione del for, una soluzione, chiamiamola s, verrà copiata in un array, avente tanti elementi quante sono le variabili più uno per la preferenza, $t(s)$. A questo punto, una funzione, simile a quella usata da STPP_RANDOM_SOLVER per dare le preferenze, computa $h(s)$, la preferenza data a s dall'STPP corrente. Il motivo che ha permesso di riutilizzare la funzione già implementata per il risolutore è il fatto di aver scelto la versione discontinua di min per il calcolo $h(s)$, scelta fatta per i motivi indicati sopra. Viene dunque calcolato l'errore. Osserviamo che la riga 4 riporta un passo che in realtà non viene effettuato. Infatti non ha senso calcolare esattamente l'errore quadratico non cumulativo, che ricordiamo è:

$$E = \frac{1}{2}(t(s) - h(s))^2$$

ma solo la parte che compare nella sua derivata rispetto ai parametri delle funzioni, cioè, come si vede dalla formule, 6.23, 6.24, e 6.25:

$$(t(s) - h(s)).$$

In parole povere a questo punto STPP_LEARNING_MODULE, fa semplicemente la differenza tra la preferenza corretta di quella soluzione, che compare nel training set, e quella ipotizzata per la medesima dall'STPP corrente. Con questo conto in realtà è già cominciata la fase di calcolo degli aggiornamenti.

UPDATE(P,E,η,β) nell'implementazione non è una sola funzione ma un insieme di funzioni che calcolano, ciascuna, una parte di quelle in cui vengono smembrate le 6.26, 6.27, e 6.28. Una prima funzione calcola:

$$\frac{\mathrm{e}^{-\beta(a_i s_i^2 + b_i s_i + c_i)}}{\frac{1}{\nu}\sum_{j=1}^{\nu} \mathrm{e}^{-\beta(a_j s_j^2 + b_j s_j + c_j)}}$$

che come si può osservare è comune a tutte e tre le equazioni. Poi tre funzioni specifiche per i tre tipi di parametri completano il calcolo. Fatto questo vengono portati a termine gli assegnamenti indicati dalle suddette equazioni, completando la fase di aggiornamento. Sopra è stato illustrato come questi assegnamenti mantengano la semi-convessità. Questo processo viene ripetuto per ciascuna soluzione nel training set, fino a che si sono visitate tutte.

2. *Fase di verifica.* Questa parte riguarda le righe 7, 8, 9 dell'algoritmo. In questa fase viene effettuata un'altra passata completa sul training set e vengono calcolati: l'errore quadratico cumulativo con parabole libere:

$$E_c = \frac{1}{2} \sum_{s \in Tr} (t(s) - h(s))^2,$$

l'errore quadratico cumulativo con fuzzy-parabole:

$$E_c^f = \frac{1}{2} \sum_{s \in Tr} (t(s) - h(s)^f)^2,$$

poi l'errore massimo assoluto con parabole libere:

$$E_{max}^f = max\{|t(s) - h(s)| \ , s \in Tr\},$$

e con fuzzy-parabole:

$$E_{max}^f = max\{|t(s) - h(s)^f| \ , s \in Tr\},$$

l'errore medio assoluto con parabole libere:

$$E_{med} = \frac{1}{|Tr|} \sum_{s \in Tr} |t(s) - h(s)|,$$

l'errore medio assoluto con fuzzyparabole:

$$E_{med}^{f(t)} = \frac{1}{|Tr|} \sum_{s \in Tr} |t(s) - h(s)^f|,$$

In queste formule $h(s)^f$ indica la preferenza calcolata, sempre con una versione diacontinua di min ma con fuzzy-parabole.

Si noti come all'ultimo errore si sia messo l'apice t. Questo significa che esso è l'errore medio all'iterazione, del $while$, t. Infatti, ora l'algoritmo deve calcolare $PLI_{E_{med}}^f$ e per far questo deve aver salvato il miglioramento del passo precedente relativo all'errore medio assoluto fuzzy, chiamiamolo I_{t-1}. Esso sarà stato:

$$I_{t-1} = E_{med}^{f(t-1)} - E_{med}^{f(t-2)}.$$

La prima cosa da fare è calcolare il miglioramento, che nel caso sia negativo è un peggioramento, che è stato ottenuto dall'ultima batteria di aggiornamenti:

$$I_t = E_{med}^{f(t)} - E_{med}^{f(t-1)}$$

$PLI^f_{E_{med}}$ è la percentuale di I_{t-1} che rappresenta I_t:

$$PLI^f_{E_{med}} = \frac{I_t \times 100}{I_{t-1}}$$

Viene, dunque, controllato se questa percentuale è superiore a quella fissata come limite, $Thres$, che può essere, ad esempio 60%, 70%, 80%. Se il miglioramento non è stato sufficiente viene incrementato di 1 il contatore delle iterazioni che hanno portato a miglioramenti esigui o a peggioramenti; se invece l'errore medio assoluto è migliorato sufficientemente allora il contatore viene azzerato. Quando il contatore supera il massimo numero di iterazioni "fallimentari" consecutive tollerate, $maxit$, la fase di apprendimento termina con l'uscita dal ciclo $while$, riga 10.

Valutazione dell'apprendimento tramite il test set

Prima di cominciare la fase di valutazione finale, l'STPP finale viene trasformato con due operazioni:

- vengono troncate le parabole (parabole $\longrightarrow$ fuzzy-parabole);
- ai vincoli viene data la rappresentazione punto a punto.

Il compito di valutazione finale si riferisce al calcolo degli errori sul test set alla riga 11 dell'algoritmo. Una volta che la fase di apprendimento è terminata, si passa a verificare i risultati ottenuti, proponendo all'STPP che è stato generato dal modulo, delle soluzioni mai viste prima.

Per valutare effettivamente la qualità della soluzione appresa, vengono calcolati E^{ts}_c, E^{ts}_{max}, e E^{ts}_{med} sul test set. Osserviamo che , pur avendo omesso l'apice f per questi errori in quanto sono gli unici calcolati sul test set, essi si riferiscono a fuzzy-parabole. E^{ts}_c viene calcolato perché è su di esso che è stata compiuta la discesa di gradiente. Infatti anche se essa è stata effettuata stocasticamente, come è stato osservato nella Sezione 2.4, questo metodo approssima di fatto la discesa di gradiente sull'errore quadratico cumulativo. E^{ts}_{max} indica a quanto ammonta l'errore più grave commesso, cioè la massima differenza, in valore assoluto, tra la preferenza effettiva di un esempio nel test set e quella proposta dal STPP prodotto dal modulo. E^{ts}_{med} è l'errore più rappresentativo. Infatti esso ci dice come è andato in media l'errore sul test set. Viene, infatti, calcolato facendo la media delle differenze tra preferenze effettive e proposte, prese in valore assoluto.

Fase di output

L'output di STPP_LEARNING_MODULE è molto articolato[8]. Esso infatti produce:

1. *un file di log* durante tutto il processo di apprendimento. A questo file, vengono aggiunti, ogni volta che vengono compiute un certo numero di iterazioni del ciclo *while*, gli errori calcolati a quella iterazione. Come abbiamo osservato si tratterà dei valori di E_c, E_{max}, E_{med}, E_c^f, E_{max}^f, E_{med}^f. Verrà inoltre specificato il numero dell'iterazione e il valore del parametro η [9].

2. *Un file contenente l'STPP prodotto dal modulo.*

3. *Un file contenente i risultati sul test set.* Esso mostrerà le soluzioni del test set con le preferenze corrette affiancate da quelle ipotizzate. Gli errori E_c^{ts}, E_{max}^{ts}, E_{med}^{ts} ed il numero totale di iterazioni.

6.3 Scelte implementative

6.3.1 Il linguaggio di programmazione: C++

La scelta del linguaggio di programmazione è stata fatta principalmente per uniformarsi a quanto fatto per il risolutore. Tuttavia, è utile osservare come, nell'implementazione del modulo di apprendimento automatico, non si sia resa particolarmente necessaria la connotazione object-oriented di C++. Una caratteristica, di questo linguaggio, che invece si è rivelata utile è stata la facilità di inserire nel programma dei conti puramente matematici. L'implementazione built-in di alcune funzioni, come l'esponenziale, ha facilitato molto la fase di programmazione dei calcoli, discretamente complicati, che compaiono nella fase di aggiornamento del modulo.

Questo programma inoltre è stato creato per studiare le sue potenzialità all'interno di ambienti software già esistenti [10], ed implementati in C++.

6.3.2 La definizione degli oggetti

Il modulo di apprendimento automatico non ha utilizzato nuovi oggetti rispetto a quelli già definiti dal risolutore. Esso, però, ha mantenuto rappresentazioni diverse degli stessi oggetti rispetto a STPP_SOLVER. Infatti, la rappresentazione delle funzioni di preferenza viene mantenuta parametrica

[8] Per questo non è stata data nessuna ulteriore specifica nella riga 10 della definizione dell'algoritmo.

[9] Rimandiamo tutte le considerazioni sui parametri η e β alla seguente sezione sulle scelte implementative.

[10] Gli ambienti a cui ci riferiamo sono stati sviluppati al centro ricerche NASA Ames.

durante tutto il processo di apprendimento. Chiariamo questo punto: la fase di aggiornamento cambia i parametri a, b, e c di tutte le funzioni di preferenza nel modo descritto nella sezione precedente. Se si volesse ad ogni iterazione mantenere una rappresentazione punto a punto del vincolo, dopo ogni aggiornamento dovrebbero essere ricalcolate le immagini di tutti i punti di tutti gli intervalli. Questo è evidentemente uno spreco, in quanto, l'elaborazione di una soluzione del training set comporta il calcolo dell'immagine di un solo punto di ogni intervallo, il punto identificato dalla soluzione su ciascun vincolo. È evidentemente più conveniente calcolare ogni volta l'immagine di questi punti utilizzando la formula delle parabole, piuttosto che calcolare tutte le immagini ogni volta. Osserviamo che è in via di sviluppo un nuovo risolutore, di cui parleremo nella conclusione, nel quale scompare la rappresentazione punto a punto. Il modulo, dunque, è già pronto per produrre un problema nel formato richiesto dal nuovo risolutore.

6.3.3 Gestione dei parametri del problema

Osserviamo, innanzitutto che come parametri del problema non intendiamo i parametri delle parabole, ma le variabili che influiscono sul comportamento del modulo di apprendimento. Esse sono: il numero di esempi del training set, il numero di esempi nel test set, il passo di discesa η e il coefficiente di approssimazione β. Abbiamo già osservato come i primi due debbano essere forniti dall'utente tramite lo standard input durante la fase di inizializzazione. Consideriamo gli ultimi due e cerchiamo di capire che ruolo hanno nell'apprendimento.

- β. Questo è l'indice di approssimazione di quanto si avvicini la versione continua di min a min "vero". Nella Sezione 2.4 il suo ruolo è già stato chiarito. Ricordiamo che esso è lo stesso parametro che compare nella sigmoide, che come è noto, approssima la funzione a scalino. Dunque, tanto più β è alto tanto più l'approssimazione sarà fedele. Tuttavia più questo parametro è elevato più la discesa di gradiente diventa difficile. La funzione min, come abbiamo più volte ricordato, è una funzione discontinua, quindi più fedele è l'approssimazione più si avvicina ad essere discontinua. La discesa di gradiente avviene nello spazio di dimensione 3ν, cioè tre volte il numero di vincoli. Un innalzamento del parametro β provoca una complicazione a livello della superficie che rappresenta la funzione errore. Introdurre spigoli ed irregolarità aumenta la probabilità di cadere in un minimo locale durante il processo di discesa di gradiente. Nella nostra implementazione abbiamo dunque optato per β non eccessivamente grande di modo da avere una superficie più smussata, anche a scapito della precisione dell'approssimazione. Inoltre,

dipendendo questo parametro essenzialmente dal numero di vincoli, in questa implementazione non ne è stato predisposto l'inserimento da parte dell'utente. L'uso del modulo di apprendimento, infatti, per ora non è certo destinato ad un pubblico molto vasto e questo per due motivi: esso è ancora in fase sperimentale, e inoltre potrebbe essere incluso in un sistema NASA che si occuperebbe direttamente dell'inizializzazione di β e degli altri parametri.

- η. Il passo di discesa ha un'azione molto più diretta, e meno misteriosa rispetto a β, sul processo di discesa di gradiente. Esso infatti regola la magnitudine degli aggiornamenti. Più esso è alto più le modifiche ai parametri parabolici saranno pesanti. Questo comporta un vero e proprio salto nella direzione opposta al gradiente. Nei nostri esperimenti, come si potrà constatare nella sezione seguente, η è dell'ordine di $10^{-5}, 10^{-6}$. I motivi per un valore così piccolo sono fondamentalmente due. Prima di tutto la superficie su cui ci si muove è molto complicata ed è immersa un uno spazio con dimensione abbastanza alta. Valori per η troppo alti, dunque, possono causare un'oscillazione, anche abbastanza accentuata della funzione errore. Il secondo motivo è che i valori che il modulo deve apprendere si trovano in un range limitato, quello fuzzy appunto $[0, 1]$. I valori di preferenza tendono ad essere molto vicini, e nei problemi toy i valori sono anche limitati in numero. L'ideale sarebbe partire con un η basso e di farlo poi crescere una volta che si è imboccata la discesa. Questo tipo di esperimento è stato fatto, non si è riusciti, però, a trovare una formula di variazione per η soddisfacente per più problemi. La scelta alternativa, che è stata adottata è quella di mantenere questo parametro molto basso, impiegando eventualmente più iterazioni. Anche questo parametro, ad oggi, può essere gestito solo dal programmatore o da chi ha accesso al codice. I motivi di questo sono gli stessi che per β. La previsione è quella di includere il modulo in un sistema che riesca automaticamente a monitorare l'andamento dell'apprendimento e a prendere decisioni i tempo reale sul valore di η.

6.3.4 L'output

Vogliamo ancora spendere due parole sull'output del modulo. Come prima cosa si può notare come vengano calcolati errori totalmente inutili ai fini del programma. In realtà essi rientrano in un sistema di monitoraggio dell'andamento dell'apprendimento. Il file di log, di cui si è già accennato, dà informazioni su come procede la discesa di gradiente. Da E_c si può vedere se vi sono oscillazioni molto alte, ad esempio a causa di un passo di discesa troppo elevato, oppure se si è raggiunto un plateau. Dal confronto tra E_{max}

e E_{med} si possono trarre delle conclusioni sul rapporto tra l'errore massimo e l'errore medio. Una caratteristica dei nostri problemi, ad esempio, sarà quella di avere errori massimi abbastanza elevati ed errori medi discretamente bassi. Da questo si può supporre che il modulo tenda ad apprendere migliorando un po' su molte soluzioni e magari peggiorando molto su poche altre. Il confronto tra gli errori calcolati con fuzzy-parabole e con parabole libere permette di capire quanto in realtà il troncamento sia determinante, cioè con che grado di generalità le parabole libere approssimano il problema[11].

Il file che contiene i risultati sul test set è un po' come il verbale di un esame. La lista degli esempi del test set con le preferenze corrette e ipotizzate permette di toccare con mano i risultati dell'apprendimento. Anche solo scorrendo velocemente questa lista è possibile rendersi conto se sono stati commessi errori madornali. L'errore quadratico cumulativo può indicare, talvolta, se è stato colpito un minimo locale. L'errore massimo assoluto e l'errore medio assoluto possono costituire una misura sul grado di confidenza che si può avere sul problema appreso.

Un'ultima osservazione, sull'output, va fatta sulla modalità di output dell'STPP appreso. Potrebbe infatti sembrare in contraddizione con quanto detto sulla rappresentazione degli oggetti. Alla fine il problema appreso viene rappresentato punto a punto e con le parabole troncate. Il motivo del troncamento è già stato chiarito, per quanto riguarda, la rappresentazione punto a punto essa è stata scelta per motivi sperimentali. Infatti in alcuni problemi si è voluto vedere effettivamente che forma assumessero le parabole e quanto, nel caso dei problemi toy, esse si discostassero da quelle originali. La rappresentazione è inoltre completa. Infatti è interessante vedere anche che forma hanno assunto le funzioni su vincoli che nel problema originario erano di default. Osserviamo infine che questa scelta sulla rappresentazione finale non ha nessun nesso con l'applicabilità del risolutore che può gestire anche un input con tutti vincoli dati tramite i parametri delle parabole.

[11] Intendiamo dire che l'analisi di questi valori ci permette di capire se le parabole libere si assestano a valori ragionevoli intorno ai punti che magari compaiono più spesso come proiezioni di soluzioni e sbagliano totalmente su altri o se invece approssimano bene il problema in tutti i punti.

Capitolo 7

Risultati Sperimentali di Apprendimento

Questo capitolo è dedicato agli esperimenti condotti con il modulo di apprendimento. Vengono prima descritte tre classi di problemi a cui è possibile applicare l'algoritmo.

La Sezione 7.2 descrive i risultati ottenuti con problemi generati random. Le ultime due Sezioni, invece, mostrano due esempi di possibili applicazioni nell'ambito della ricerca spaziale.

7.1 Uso del modulo di apprendimento

Il modulo di apprendimento automatico ha a disposizione i vincoli hard del problema e alcune soluzioni con le rispettive preferenze. In pratica esso trasforma un STP in un STPP traendo delle conclusioni da degli esempi che gli vengono forniti. Questo viene fatto mediante una discesa di gradiente stocastica sull'errore. Le informazioni locali indotte dal modulo sono espresse dalla forma delle funzioni di preferenza che troviamo alla fine del processo di apprendimento. Ci si può chiedere su quali basi si è ritenuto che un programma come questo potesse essere utile. Esso è stato disegnato per affrontare due casistiche:

- **problemi in cui sono noti i vincoli hard ed una funzione globale di preferenza.** In questo caso abbiamo un STP ed una funzione globale di preferenza che mappa un assegnamento globale in un certo reale, che sarà appunto la preferenza. Si suppone che l'STP sia troppo grande per essere risolto in un tempo ragionevole, altrimenti basterebbe trovare tutte le soluzioni, applicarvi la funzione e cercare quelle ottime rispetto a tale funzione. A questo punto vengono generati due sottoinsiemi di-

sgiunti di soluzioni che vengono valutate dalla funzione. Si otterranno così il training e il test set. Si possono incontrare due casi:

- *La funzione di preferenza globale è sicuramente rappresentabile dallo schema max-min degli STPPs con semianello* S_{FCSP} [1]. Questo caso si riferisce quasi esclusivamente al primo gruppo di problemi su cui abbiamo testato il modulo, problemi cioè generati dal generatore, e dunque con funzioni di preferenza locali note, usati per testare il comportamento del modulo.
- *La rappresentabilità nello schema max-min non è certa.* Questi sono i problemi che sono stati studiati al centro NASA Ames e che saranno presentati e discussi in dettaglio nel Capitolo 6. Le funzioni di preferenza considerate associano ad ogni soluzione, che in realtà rappresenta una schedulazione, un livello di preferenza dipendente dal consumo di risorse proprio degli eventi in quella schedulazione. L'idea è quella di applicare il modulo per ottenere un STPPs con schema $max - min$ che simuli la funzione di preferenza globale. Se, come in tutti i casi fino ad ora studiati, c'è convergenza ad un STPP, le soluzioni ottime di questo problema saranno soluzioni ottime, o quasi, anche rispetto alla funzione globale. Questo verrà descritto dettagliatamente nel capitolo 6.

- **Problemi in cui la funzione di preferenza non è nota**. Anche questi sono problemi di interesse pratico. Si supponga, per esempio, di voler produrre delle schedulazioni per le attività di un telescopio. Dall'esperienza, intanto sarà molto difficile dare le preferenze locali, ma inoltre in questo campo l'esperto umano non è ancora stato soppiantato completamente da un planner. Se infatti si sottopongono ad un esperto varie pianificazioni degli eventi esso saprà dare un giudizio sulla loro qualità ma questo giudizio sfugge ad una formulazione matematica. Si noti come in questo caso, l'approccio mediante apprendimento automatico sia praticamente l'unica opzione. Infatti anche ammesso di poter risolvere completamente il problema *hard* sarebbe impensabile di sottoporre all'esperto tutte le soluzioni. Per il nostro approccio, invece, l'esperto dovrebbe solo dare alcuni esempi da utilizzare nel training e nel test set ed il modulo, nel caso in cui il meccanismo di giudizio possa essere approssimato ragionevolmente dallo schema $max - min$, fornirebbe un STPP che simula il comportamento dell'esperto.

[1] Osserviamo che questa è l'unica rappresentazione che il modulo può fornire.

7.2 Problemi generati random

In questa sezione riportiamo i risultati di esperimenti condotti su problemi generati dal generatore random. Il procedimento seguito è simile a quello per il risolutore. Prima, attraverso una chiamata al generatore, viene prodotto un STPP. Esso costituisce l'input per STPP_RANDOM_SOLVER, il quale produce il training e il test set. A questo punto la rete di vincoli hard, il training e il test set, vengono forniti al modulo che effettua l'apprendimento.

Ricordiamo quali sono i parametri del generatore random: il numero di variabili n, il dominio entro qui vengono scelti gli assegnamenti per la soluzione iniziale r, l'intero che indica il massimo grado di espansione degli intervalli max, le densità dei vincoli definiti D e i parametri di perturbazione dei coefficienti della parabola pa, pb, e pc.

Tra questi alcuni sono stati mantenuti fissi, mentre altri sono stati fatti variare ottenendo vari tipi di problemi per gli esperimenti.

Abbiamo mantenuto costante il dominio in cui vengono scelti i valori da assegnare alle variabili per la prima soluzione. Il modulo, come il risolutore, non è sensibile alla grandezza degli assegnamenti. È necessario però sottolineare che una traslazione del problema in avanti o indietro nell'asse temporale di uno stesso STPP, quindi un problema con gli intervalli di una certa misura e le parabole con una certa forma, provoca un cambiamento, oltre che degli estremi degli intervalli, anche dei parametri delle parabole. Tuttavia la difficoltà dell'apprendimento non è legata nemmeno ai valori assunti dalle parabole. Infatti la difficoltà di apprendimento di una curva da parte di un sistema di apprendimento automatico non dipende da dove nello spazio sia posta ma dalla sua forma, più o meno regolare, e dal numero di esempi, coppie $(valore, immagine)$, disponibili.

In tutti gli esperimenti è stato mantenuto $r = 40$, significando che il problema generato ha una soluzione in cui tutti gli eventi avvengono nelle prime 40 unità di tempo. Osserviamo come questo non limiti il numero di variabili ad essere inferiore a 40. Infatti i problemi generati dal generatore possono esprimere sistemi di attività in cui molte di esse si svolgono contemporaneamente, cosa che rappresenta meglio la realtà.

Anche per gli esperimenti del modulo sono stati mantenuti fissi i parametri di perturbazione dei coefficienti delle parabole. Essi sono stati fissati in modo tale che il generatore producesse problemi in cui le preferenze associate alle soluzioni avessero un range non banale. Infatti ammettere traslazioni troppo grandi (cioè avere pc alta) significherebbe produrre molti problemi che, una volta risolti, hanno livello ottimo di preferenza a 0. Questo comporta il fatto di avere un solo livello di preferenza per tutti gli esempi da fornire al modulo. I valori scelti per pa, pb hanno seguito la logica considerata per il risolutore e

cioè di mantenerli bassi per velocizzare la generazione del problema. I valori scelti per *pa*, *pb*, e *pc* sono rispettivamente 10%, 10% e 5%.

Passiamo ora ai parametri che sono stati fatti variare.

I numeri di variabili considerati per gli esperimenti è stato $n = 15$, $n = 20$ e $n = 25$. Il numero di variabili è direttamente proporzionale al numero di vincoli e dunque dà una misura del carico computazionale che ciascun aggiornamento comporta. A ciascuno di questi valori corrisponde una tabella diversa.

È stato poi fatto variare il coefficiente di massima espansione degli intervalli max che ha assunto i valori 40, 30, e 20. Esso è stato il parametro discriminante per il numero di esempi del training e del test set. Infatti problemi generati con max più alto hanno potenzialmente più soluzioni. Al modulo sono stati proposti 700 esempi per apprendere tali problemi. La cardinalità del test set è stata, per semplicità, mantenuta sempre uguale a quella del training. In tutti i problemi considerati, con $max = 40$ è stato sempre possibile trovare 1400 soluzioni diverse. Per problemi con $max = 30$ sono stati forniti al modulo 600 esempi e per problemi con $max = 20$, 500 esempi. Nessuno dei problemi generati aveva meno di 1000 soluzioni. L'ultimo parametro da considerare è la densità, D, dei vincoli, che in questi esperimenti ha assunto valori 40%, 60%, e 80%. Come si vede sono valori abbastanza alti. Infatti ricordiamo che queste percentuali indicano il numero di vincoli che verranno definiti dal generatore, cioè quelli che avranno una funzione di preferenza diversa dalla costante 1. Dunque più alta è la densità più si hanno funzioni non banali su cui ha significato apprendere.

Oltre ai parametri del generatore sono stati fissati anche valori per i parametri del modulo: $\beta = 8$ e $\eta = 10^{-7}$. L'inizializzazione delle parabole nel modulo, cioè le funzioni di preferenza poste alla prima iterazione, sono sempre state poste alla costante 1.

Il criterio di stop è stato quello di 100 violazioni consecutive dove, per violazioni, intendiamo passi di aggiornamento in cui il miglioramento è stato inferiore al 70% di quello precedente.

Nelle tabelle 7.1, 7.2, e 7.3 è indicato E^{ts}_{med} cioè l'errore medio assoluto sul test set. Il primo valore che compare è la media di tale errore su tre problemi generati a partire dagli stessi parametri. I valori tra parentesi sono rispettivamente il minimo e il massimo assunto da tale errore sui tre problemi.

A fronte dei risultati ottenuti possiamo osservare che:

- Gli errori sono in generale ragionevoli se si pensa che le preferenze sono numeri compresi tra 0 e 1 con due cifre decimali.

- L'errore è indipendente dal numero di variabili; questo significa che ci possono essere problemi con meno variabili più difficili da apprendere di

	D=40	D=60	D=80
max=20	0.017 (0.013,0.022)	0.007 (0.006,0.008)	0.0077 (0.0075,0.0081)
max=30	0.022 (0.017,0.025)	0.013 (0.01,0.017)	0.015 (0.005,0.028)
max=40	0.016 (0.011,0.019)	0.012 (0.012,0.013)	0.0071 (0.006,0.0079)

Tabella 7.1: Tabella che indica E^{ts}_{med} medio, minimo e massimo relativamente a tre esempi di un STPP con $n = 25$, $r = 40$, $pa = 10$, $pb = 10$, e $pc = 5$.

	D=40	D=60	D=80
max=20	0.032 (0.022,0.043)	0.012 (0.01,0.016)	0.005 (0.004,0.006)
max=30	0.032 (0.021,0.040)	0.018 (0.016,0.021)	0.0074 (0.0073,0.0077)
max=40	0.033 (0.05,0.021)	0.023 (0.025,0.022)	0.016 (0.011,0.021)

Tabella 7.2: Tabella che indica E^{ts}_{med} medio, minimo e massimo relativamente a tre esempi di un STPP con $n = 20$, $r = 40$, $pa = 10$, $pb = 10$, e $pc = 5$.

	D=40	D=60	D=80
max=20	0.018 (0.01,0.028)	0.009 (0.007,0.012)	0.009 (0.007,0.012)
max=30	0.021 (0.018,0.027)	0.014 (0.011,0.017)	0.016 (0.010,0.021)
max=40	0.024 (0.023,0.026)	0.019 (0.019,0.021)	0.0086 (0.007,0.01)

Tabella 7.3: Tabella che indica E^{ts}_{med} medio, minimo e massimo relativamente a tre esempi di un STPP con $n = 15$, $r = 40$, $pa = 10$, $pb = 10$, e $pc = 5$.

altri con più variabili e che il bilanciamento del numero di esperimenti forniti è stato ragionevole.

- Sembra esserci una certa connessione tra la densità e i risultati ottenuti. Infatti più la densità è alta più l'errore è basso. Questo può esser dovuto al fatto che più vincoli sono stati definiti, più vasto è il range di valori di preferenza disponibili. Da ciò risulta un training set in cui le soluzioni assumono valori più diversificati facilitando il modellamento delle funzioni di preferenza locali.

- Facciamo un' ultima considerazione sul numero di iterazioni e sui tempi impiegati. Il numero minimo di iterazioni compiute sono state 357 il numero massimo 3812. Sono numeri ragionevoli. A questo si aggiunga il fatto che il tempo più lungo impiegato complessivamente è stato di 8 minuti e 18 secondi, il più breve di 2 minuti e 31 secondi[2]. I risultati ci appaiono dunque soddisfacenti alla luce del tempo e dello sforzo computazionale spesi.

7.3 Problemi NASA

Durante il periodo trascorso al laboratorio di ricerca NASA Ames, è stato implementato il modulo di apprendimento e sono stati svolti i primi esperimenti. Inoltre sono stati studiati problemi in cui si conoscevano solo i vincoli hard e la funzione globale di preferenza. Quello che è apparso subito ovvio è che quando si hanno problemi di tipo reale, spesso è più facile dare le preferenze direttamente ad una soluzione invece che dare tutte le preferenze locali. Tuttavia nella maggior parte dei casi non è ovvio come tradurre la funzione di preferenza globale in un STPP basato sul semianello fuzzy. D'altra parte questo sarebbe molto utile in quanto il risolutore permetterebbe di trovare una soluzione ottima.

Il problema è il seguente: sono noti dei vincoli hard, quindi un STP, ed una funzione globale di preferenza, chiamiamola F. Si vuole apprendere un STPP il cui comportamento globale, a livello delle soluzioni simuli la F. Più precisamente. Supponiamo di lavorare con n variabili. F associa ad ogni soluzione un valore reale compreso tra 0 e 1. Una soluzione è un assegnamento completo alle variabili consistente con i vincoli, hard, dell'STP. Dunque, F mappa n-uple di interi in reali dell'intervallo [0,1], cioè, indicando come D un sottoinsieme di Z grande abbastanza da includere i domini di tutte le variabili:

[2] Si osservi che il problema che ha compiuto meno iterazioni non è quello che ha impiegato meno tempo in quanto il tempo dipende fortemente dal numero di esempi nel training set.

$$F : D^n \longrightarrow [0,1] \subset \Re$$

$$F(s) = F(x_1, \cdots, x_n) = p \in [0,1]$$

Il nostro scopo è quello di trovare delle funzioni $f_1, \cdots, f_\nu$ [3] così definite:

$$f_i : I_i \longrightarrow [0,1]$$

$$f_i(s_i) = p_i \in [0,1]$$

dove I_i è l'intervallo dell'i-esimo vincolo e s_i la proiezione di s su quel vincolo, tali che un STPP avente gli stessi vincoli a livello hard dell'STP e le $f_1, \cdots, f_\nu$ come funzioni di preferenza, associa, con il solito metodo $max - min$, alle stesse soluzioni valori che approssimano quelli associati dalla F. In altre parole vogliamo trovare $f_1, \cdots, f_\nu$ tali che valga:

$$F(s) = F(x_1, \cdots, x_n) = min(f_1(s_1), \cdots, f_\nu(s_\nu))$$

Il nostro modulo di apprendimento, che è il programma che dovrà trovare queste funzioni, impone però un'ulteriore restrizione: quella della rappresentabilità delle medesime in forma quadratica semi-convessa. In altre parole si cercheranno gli a_h, b_h, e c_h con $h = 1, \cdots, \nu$, tali che sia soddisfatta:

$$F(s) = F(x_1, \cdots, x_n) = min(a_1 s_1^2 + b_1 s_1 + c_1, \cdots, a_\nu s_\nu^2 + b_\nu s_\nu + c_\nu)$$

Sono stati studiati due tipi di funzioni.

1. Dal campo della *schedulazione* sono state considerate due funzioni che sono di interesse per le missioni spaziali in genere. Si tratta di "minimize makespan", e "minimize maximum lateness".

   - **Minimize makespan.** L'obbiettivo da raggiungere è quello di trovare una schedulazione degli eventi con durata complessiva minima, cioè una in cui il tempo trascorso tra l'inizio della prima attività e la fine dell'ultima sia minimo. Le applicazioni di questa funzione sono ovvie: nello spazio il tempo è prezioso e schedulazioni di durata minima sono in molti casi da preferirsi.

     Questa funzione viene modellata nel nostro schema nel seguente modo. Prima di tutto è necessario dividere le variabili che rappresentano l'inizio di eventi dalle variabili che ne rappresentano la fine. Data una soluzione, la funzione makespan vi associa la differenza tra il massimo dei valori assegnati alle variabili di tipo fine e il minimo dei valori assegnati alle variabili di tipo inizio.

[3] Ricordiamo che con ν indichiamo il numero totale di vincoli.

Da makespan dobbiamo ora ottenere una funzione di preferenza che premi, cioè assegni un valore di preferenza più alto [4], a soluzioni di durata complessiva minore e che penalizzi quelle in cui la durata complessiva è più dilatata. Questo viene fatto caso per caso, considerando la struttura del problema dalla quale si può evincere un'idea di quella che può essere la durata minima e massima. Queste informazioni ci permettono anche di normalizzare la preferenza, in modo tale che rientri in $[0, 1]$.

- **Minimize maximum lateness**. L'obbiettivo è quello di ottenere una schedulazione in cui l'ultima attività termini il più presto possibile. In altre parole si cerca un ordine degli eventi tale che la fine dell'ultima attività da eseguire nell'ordine, si posizioni il più presto possibile nell'asse temporale. Le applicazioni sono per lo più rivolte a problemi in cui c'è un orizzonte di tempo che non va assolutamente oltrepassato. Anche questi problemi sono comuni nell'ambito spaziale: basti pensare agli orizzonti entro i quali un agente remoto può cominicare a terra, etc.

  Si osservi come questa funzione è molto simile a makespan. Infatti se si pone che la prima attività cominci sempre all' "inizio del mondo" essa si riduce ad un caso particolare di makespan. Tuttavia se questa circostanza non si verifica, come accade spesso nella realtà in quanto è difficile far cominciare le attività degli enti spaziali esattamente al tempo 0 del cronometro, maximum lateness si rivela meno restrittiva di makespan. Infatti, secondo questo criterio, una buona soluzione termina presto, anche se la durata complessiva dei suoi eventi può essere maggiore di una soluzione che termina più tardi.

  Il modellamento della funzione di preferenza per il nostro schema è ancora più semplice in questo caso. Si parte prendendo il massimo dei valori associati alle variabili di tipo fine poi si passa a sistemare la funzione in modo da penalizzare soluzioni con massimi alti e premiare quelle con massimi più bassi.

  Si possono anche implementare delle varianti sia della prima che della seconda funzione, ad esempio fissando esplicitamente una soglia e dei coefficienti di penalità che crescono linearmente con la violazione, del makespan o della maximum lateness, rispetto alla soglia.

2. Dallo studio del **consumo di risorse** si possono ottenere varie funzioni

[4] Ricordiamo come l'ordine indotto dall'operatore additivo di S_{FCSP}, che è max, sia l'ordine usuale sui reali.

di preferenza globali. Il goal generico è quello di minimizzare il consumo di risorse complessivo associato ad una soluzione s. Come soluzione si intende sempre una schedulazione di eventi che, per la loro esecuzione, richiedono un certo consumo di una data risorsa. Questi problemi sono onnipresenti nella preparazione delle missioni spaziali. Ovviamente la risorsa di cui ci si occupa di più è l'energia, che pur essendo presente in quantitativi enormi nello spazio è spesso difficilmente sfruttabile. Questo tipo di problema dunque riguarda attività che consumano energia solare, acqua, ossigeno, risorse umane etc.

La funzione con cui lavoreremo è molto semplice. Supponiamo di avere un'unica risorsa disponibile a cui devono attingere tutte le attività. Nell'esperimento essa viene ritenuta inesauribile ma costosa. Questo significa che essa sarà sufficiente affinchè in ogni caso tutte le attività possano sempre essere portate a termine, ma sono da preferirsi le schedulazioni in cui il consumo è minimo. Supponiamo che l'attività w_i, alla quale, ricordiamo, corrispondono due variabili, consumi e_i energia per unità di tempo. Ad esempio, un movimento di 10 unità spaziali in direzione sud-est per K9[5] consuma 8 unità di energia per unità di tempo. Supponiamo che per svolgere questa attività vengano impiegate 5 unità di tempo: il consumo totale di energia sarà di 40 unità energetiche. Ci proponiamo, dunque, di studiare il caso in cui la relazione tra il consumo di energia e il tempo è lineare e i coefficienti di proporzionalità variano a seconda del tipo di attività. Tali coefficienti saranno noti, cioè data una soluzione saremo sempre in grado si calcolarne il consumo di risorse.

Per il modellamento partiamo proprio dalla funzione che associa ad ogni soluzione il consumo che essa comporterebbe:

$$f(s) = \sum_{i \in VD} (e_i s_i).$$

In questa formula compare VD che è il sottoinsieme dei vincoli del problema che rappresentano la durata di qualche evento. Infatti, in un problema in cui le variabili sono l'inizio o la fine di un evento, i vincoli possono essere di durata o di distanza tra gli eventi. Un modello più realistico considererebbe anche i vincoli di distanza. Prendiamo ancora una volta ad esempio K9. È ragionevole pensare che ci saranno dei momenti in cui verrà fatto riposare, ma durante i quali deve stare comunque acceso in attesa di nuovi comandi da eseguire. Il fatto di rimanere acceso comporta comunque un consumo di energia, anche se

[5]Ricordiamo che K9 è il robot candidato ad andare su Marte ideato al centro NASA Ames.

basso. Un modello più fedele ne terrebbe conto; tuttavia, essendo nel nostro caso l'energia inesauribile e il suo consumo in stato "dormiente" trascurabile rispetto agli altri, noi non ne terremo conto [6] .

L'identità dei vincoli di durata verrà data dal problema, e viene adottata la convenzione di dare come inizio e fine di un evento le variabili con interi identificativi successivi, quindi si avrà che tutte le variabili di uno stesso tipo (inizio o fine) apparterranno alla stessa classe di parità (pari o dispari).

La funzione verrà normalizzata e "invertita" in modo da associare valori più alti, sempre tra 0 e 1, a soluzioni con consumi complessivi minori.

Descriviamo come sono stati impostati esperimenti di questo tipo.

Si è già osservato che i vincoli hard sono noti, cioè come dato del problema vi è un STP. Esso viene inizialmente trasformato in un STPP, nel modo usuale, cioè apponendo agli intervalli la funzione di preferenza costante a 1.

In questa prima fase viene utilizzato STPP_RANDOM_SOLVER con delle opportune modifiche. Quello che si vuole ottenere sono due insiemi disgiunti di soluzioni con le preferenze associate da una delle funzioni sopra citate. Questi due insiemi dovranno costituire rispettivamente il training e il test set per il modulo di apprendimento automatico. Per ottenere questo, viene apportata a STPP_RANDOM_SOLVER la seguente modifica: viene inibita la funzione che calcola la preferenza tramite gli operatori del semianello fuzzy, e viene sostituita da una implementazione delle funzioni globali di preferenza descritte sopra. Si osservi che alla funzione che calcola questo tipo di preferenza è necessario passare solo il numero di variabili e la soluzione, cioè l'insieme degli assegnamenti, e non serve darle alcuna informazione sulla rete di vincoli. Sottolineamo ulteriormente la differenza tra queste due funzioni: la funzione originale che utilizza lo schema $max - min$ prende una soluzione, va a cercare su ciascun vincolo la proiezione identificata da quella soluzione e il suo valore di preferenza, paragona tutti i valori di preferenza e ne prende il minimo. La funzione di preferenza globale non "standard" viene invece calcolata secondo una formula preimpostata guardando solo i valori che le variabili assumono per quella soluzione, ed eventualmente utilizzando dei coefficienti.

Il motivo per cui abbiamo ritenuto utile enfatizzare questo fatto è che in caso di successo, le due funzioni avranno comportamenti molto simili, cioè assoceranno, compiendo delle operazioni completamente diverse, alle stesse soluzioni valori di preferenza molto simili.

Si osservi che da quanto detto, traspare che ad ognuna delle tre funzioni sopra indicate dovrà corrispondere un'implementazione diversa della funzione. Tuttavia non è difficile pensare a conglobare tutto in due implementazioni,

[6] Si osservi che trascurare il consumo di una attività è come metterne il coefficiente a 0.

una relativa alla classe delle funzioni riguardanti proprietà di scedulazione, e l'altra riguardante la classe dei problemi di risorse. Queste due implementazioni potrebbero essere costruite in modo da avere una discreta potenza di rappresentazione delle funzioni nella rispettiva classe.

Continuiamo a descrivere come avviene l'esperimento. Applicando, dunque, STPP_RANDOM_SOLVER, opportunamente modificato, otteniamo il training e il test set. A questo punto questi due insiemi e la rete di vincoli hard vengono forniti a STPP_LEARNING_MODULE che effettua l'apprendimento. Quello che otteniamo è un STPP con funzioni di preferenza locali non banali.

A questo punto il problema prodotto da STPP viene risolto da STPP_ COMPLETE_ SOLVER che trova tutte le soluzioni ottime. Anche questo programma però viene adattato ulteriormente. Infatti nell'output verrà indicato , a fianco di ciascuna soluzione, il livello ottimo di preferenza che le compete in quanto soluzione ottima dell'STPP appreso dal modulo, e la preferenza calcolata utilizzando la funzione di preferenza globale da cui si era partiti. Viene, a questo punto, indagato il livello di soddisfazione dei seguenti parametri di valutazione:

- quanto vicini, numericamente, sono i valori proposti dalle funzioni;
- quanto buone sono rispetto alla funzione globale di preferenza da cui si era partiti le soluzioni ottime dell'STPP proposto dal modulo.

Questi sono criteri abbastanza generici che non richiedono di conoscere molto sul problema originario. Infatti un'idea del range in cui variano le preferenze globali assegnate dalle funzioni si può avere guardando i valori del training set o anche costruendosi a mano una soluzione con durata molto piccola e una con durata grande, ad esempio. Se invece il problema è piccolo abbastanza da essere risolto completamente avremo a disposizione tutte le soluzioni ottime rispetto alla funzione globale. Questo rende evidentemente inutile l'applicazione del modulo di apprendimento, ma problemi di queste dimensioni sono molto utili per valutare il comportamento dell'STPP proposto dal modulo. Se infatti abbiamo le soluzioni ottime del problema hard con funzione globale di preferenza possiamo considerare come parametri di valutazione:

- quante soluzioni ottime per il problema originale sono state identificate come ottime anche dall'STPP appreso.
- quante soluzioni ottime erano nel training set e quante invece sono state scoperte tramite il modulo.

A questo punto vogliamo trattare la seguente questione: le funzioni locali apposte sui vincoli dal modulo di learning sono o no significative rispetto alla

funzione globale di preferenza? Ad esempio, se parliamo di risorse avrebbe senso che attività con consumo più alto abbiano preferenza più bassa. Bene, sperimentalmente non sempre le funzioni di preferenza hanno rispecchiato a livello locale la funzione globale che si stava studiando. Il motivo di questo, secondo noi, risiede nella diversità, sottolineata sopra, della procedura con cui associano la preferenza le funzioni globali fuzzy, che sono quelle che addotta il modulo di apprendimento, e le funzioni di preferenza che valutano le risorse o il makespan etc. In altre parole, il modulo non "sa" cosa può rappresentare a livello locale la funzione globale di preferenza che lui cerca di imparare. Esso sa come costruire un STPP che valuta con lo schema fuzzy le soluzioni imitando solo globalmente il comportamento di tale funzione. Il modulo di apprendimento imposta, cioè, le funzioni di preferenza locali in modo che, utilizzate con lo schema $max - min$, diano globalmente un risultato simile a quello della funzione globale in input. Se ci si pensa pretendere che le funzioni locali siano rappresentative per l'obbiettivo che si vuole raggiungere globalmente, sarebbe come dire che la funzione globale di preferenza può essere ottenuta come min di funzioni locali che rappresentano la preferenza dallo stesso aspetto. In molti casi il successo del nostro modulo ha dimostrato che tali funzioni esistono ma che la loro interpretazione a livello locale va fatta alla luce di una funzione globale che utilizza gli operatori del semianello S_{FCSP} e non di una che combina in un altro modo gli assegnamenti fatti alle variabili.

7.4 Esempi

Presentiamo ora due esempi. In questi esempi verranno applicate allo stesso problema le funzioni di preferenza per minimizzare il maximum lateness e per minimizzare il consumo di risorse. La rete di vincoli hard sarà la stessa per entrambi i casi ed è rappresentata dalla figura seguente 7.1.

Come si può vedere abbiamo 8 attività, dunque 16 variabili . I vincoli tra le attività sono indicati dalla rete. Osserviamo come essa ci fornisca l'STP da cui partire. Riassumiamo nella tabella 7.4 i dati del problema modellato nel nostro schema.

Nella prima colonna della tabella 7.4 sono indicati i nomi, simbolicamente delle lettere, delle 8 attività. Nella seconda colonna sono indicati gli interi identificativi delle variabili corrispondenti all'inizio degli eventi. Nella terza colonna quelli corrispondenti alla fine degli eventi. Come si può vedere, l'inizio dell'attività C e la variabile "inizio del mondo", che, ricordiamo viene sempre identificata con 0, coincidono. Nella quarta ed ultima colonna sono indicate la durata minima e massima di una attività. Si può notare che mentre alcune delle attività hanno delle durate variabili, ad esempio l'attività D, altre hanno durate fisse, come l'attività G.

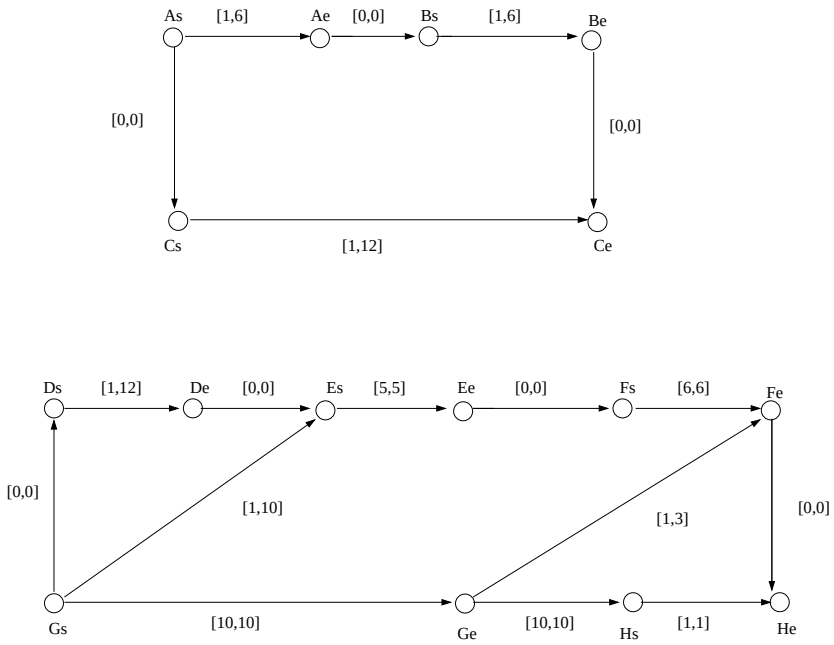

Figura 7.1: Rete di vincoli temporali semplici senza preferenze. Gli intervalli sono indicati con $[a, b]$.

7.4.1 Maximum lateness

Il primo esperimento proposto riguarda la funzione di preferenza globale del problema di minimizzazione di maximum lateness. Prima di tutto definiamo la funzione che associa ad una soluzione l'istante in cui è finita l'ultima attività. Indicheremo, come al solito, con x_h il valore assegnato alla variabile X_h [7].

$$f(s) = max\{x_3, x_5, x_1, x_7, x_9, x_{11}, x_{13}, x_{15}\}$$

Osserviamo che gli indici corrispondono tutti a variabili di fine, e che essi sono tutti dispari. Nel nostro problema $f(s)$ sarà sempre minore di 100, dunque possiamo fare una sommaria normalizzazione al fine di ottenere un valore in [0,1], dividendo per 100:

$$\tilde{f}(s) = \frac{f(s)}{100}.$$

Non rimane che sistemare le cose di modo che soluzioni con maximum lateness minore abbiano preferenza più elevata. Otteniamo così la nostra funzione di preferenza globale F:

$$F(s) = 1 - \tilde{f}(s).$$

[7] Per evitare confusione si possono interpretare gli interi identificativi di una variabile come l'indice h in X_h.

attività	variabili		durate
nome	inizio	fine	max e min durata
A	2	3	1-6
B	4	5	1-6
C	0	1	1-12
D	6	7	1-12
E	8	9	5-5
F	10	11	6-6
G	12	13	10-10
H	14	15	1-1

Tabella 7.4: Descrizione delle attività del problema. Sono indicati: il nome delle attività, le variabili corrispondenti e le durate.

Il problema di vincoli temporali semplici che stiamo considerando è relativamente piccolo. I vincoli di default sono stati posti a $[-24, 24]$ volendo rappresentare il fatto che le attività devono essere tutte svolte nell'arco di un giorno. È stato dunque possibile applicarvi una variante di STPP_ COMPLETE_ SOLVER che ha trovato tutte le soluzioni del problema hard, sfruttando il fatto che tutte le preferenze erano uguali ad 1. Tutte queste soluzioni sono poi state valutate tramite la funzione di preferenza globale.

Si è ottenuto così un insieme di 900 soluzioni con i loro valori di preferenza secondo la funzione maximum lateness. Come detto prima, a questo punto avremmo finito, in quanto basterebbe fare una scansione di questo insieme tenendo tutte le soluzioni ottime. Ma non è questo che si vuole fare, il problema è destinato a servire per testare il comportamento del modulo di apprendimento.

Una volta associate le preferenze a tutte le 900 soluzioni si vede che esse variano nell'intervallo $[0.76, 0.88]$ e tutti i valori con due cifre decimali compresi tra gli estremi sono stati associati ad almeno una soluzione. Si osservi che il range è abbastanza stretto, come accade spesso in questi problemi, e che questo non facilita molto l'apprendimento. Abbiamo applicato la procedura sopra indicata con un training set contenente 200 soluzioni ed un test set contenente 300 soluzioni. Il modulo di apprendimento ha lavorato con i parametri η e β settati rispettivamente a 10^{-7} e 8. Il criterio di stop è stato in questo caso il fatto di avere 100 iterazioni consecutive in cui non vi fosse un miglioramento pari almeno al 70% di quello precedente. Ecco i risultati ottenuti dopo 1545 iterazioni: l'errore massimo assoluto sul test set è stato pari a 0.04 e quello medio 0.01.

Il problema ottenuto è stato risolto. Nella tabella 7.5 è rappresentata

variabili																L pref	O pref
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
0	6	0	4	4	6	0	1	1	6	6	12	0	10	10	12	0.85	0.88
0	6	0	4	4	6	1	2	2	7	7	13	1	11	11	13	0.85	0.87
0	6	0	4	4	6	2	3	3	8	8	14	2	12	12	14	0.85	0.86
0	6	0	4	4	6	3	4	4	9	9	15	3	13	13	15	0.85	0.85
0	6	0	4	4	6	4	5	5	10	10	16	4	14	14	16	0.85	0.84
0	6	0	4	4	6	5	6	6	11	11	17	5	15	15	17	0.85	0.83
0	6	0	4	4	6	6	7	7	12	12	18	6	16	16	18	0.85	0.82
0	6	0	5	5	6	0	1	1	6	6	12	0	10	10	12	0.85	0.88
0	6	0	5	5	6	1	2	2	7	7	13	1	11	11	13	0.85	0.87
0	6	0	5	5	6	2	3	3	8	8	14	2	12	12	14	0.85	0.86
0	6	0	5	5	6	3	4	4	9	9	15	3	13	13	15	0.85	0.85
0	6	0	5	5	6	4	5	5	10	10	16	4	14	14	16	0.85	0.84
0	6	0	5	5	6	5	6	6	11	11	17	5	15	15	17	0.85	0.83
0	6	0	5	5	6	6	7	7	12	12	18	6	16	16	18	0.85	0.82
0	7	0	1	1	7	0	1	1	6	6	12	0	10	10	12	0.85	0.88
0	7	0	1	1	7	1	2	2	7	7	13	1	11	11	13	0.85	0.87
0	7	0	1	1	7	2	3	3	8	8	14	2	12	12	14	0.85	0.86
0	7	0	1	1	7	3	4	4	9	9	15	3	13	13	15	0.85	0.85
0	7	0	1	1	7	4	5	5	10	10	16	4	14	14	16	0.85	0.84
0	7	0	1	1	7	5	6	6	11	11	17	5	15	15	17	0.85	0.83
0	7	0	1	1	7	6	7	7	12	12	18	6	16	16	18	0.85	0.82
0	7	0	2	2	7	0	1	1	6	6	12	0	10	10	12	0.85	0.88

Tabella 7.5: Sottoinsieme dell'insieme delle soluzioni ottime dell'STPP proposto dal modulo.

una parte dell'insieme delle 252 soluzioni ottime del problema appreso con la relativa preferenza data da F a tali soluzioni.

Le prime 16 colonne della tabella 7.5 descrivono la soluzione cioè gli assegnamenti corrispondenti alle variabili i cui interi identificativi sono indicati nella prima riga. La penultima colonna dà la preferenza trovata risolvendo l'STPP proposto dal modulo. Come si vede questa è uguale per tutti in quanto queste sono tutte soluzioni ottime di quel STPP. Nell'ultima colonna è riportata la preferenza globale associata alla soluzione dalla funzione F.

Cerchiamo di verificare fino a che punto sono stati soddisfatti i criteri di valutazione:

- *Quanto vicini, numericamente, sono i valori proposti dalle funzioni.* .

Una prima risposta può essere data guardando velocemente i risultati della tabella. I valori sono abbastanza vicini: l'errore massimo è 0.03 quello minimo è 0. Una valutazione di questo fatto può anche essere fatta alla luce dei risultati sul training set che ci dicono che su 300 soluzioni l'errore medio è stato dello 0.01 e quello massimo dello 0.04.

- *Quanto buone sono le soluzioni proposte dal modulo.* Le soluzioni proposte dal modulo hanno valore di preferenza associato da F maggiore o uguale a 0.82. Questo significa che il risolutore ha proposto un STPP le soluzioni del quale sono abbastanza vicine al livello ottimo di preferenza rispetto all'insieme delle soluzioni originali. Questo è un fatto molto interessante. Infatti, è improbabile che il modulo riesca a convergere ad un STPP che abbia esattamente le stesse soluzioni ottime di quello hard con funzione globale di preferenza. Tuttavia l'apprendimento riesce, come si è visto in questo esempio e come si vedrà nel successivo, a scartare le soluzioni peggiori. In pratica, se noi consideriamo l'STP che viene ottenuto alla fase REDUCE_TO_BEST di STPP_COMPLETE_SOLVER applicato all'STPP proposto dal modulo, possiamo affermare che le sue soluzioni avranno rispetto alla funzione globale di preferenza, in questo caso maximum lateness, una preferenza che non si discosta più che di una certa quantità dal livello ottimo.

Essendo il problema in questione abbastanza piccolo possiamo rispondere alle ulteriori domande sulla performance ottenuta:

- *Quante soluzioni ottime sono state identificate dal modulo.* Il problema hard ha 37 soluzioni ottime con valore di preferenza 0.88. Tutte e 37 sono soluzioni ottime anche dell'STPP proposto dal modulo e hanno come valore di preferenza computato nello schema fuzzy 0.85.

- *Quante soluzioni ottime erano nel training set e quante, invece, sono state scoperte tramite il modulo.* Solo 8 soluzioni ottime erano comprese nel training set quindi il modulo ne ha identificate 29 nuove.

Possiamo dunque concludere che il modulo di apprendimento ha fornito un STPP che simula globalmente abbastanza bene il comportamento della funzione maximum lateness, fornendo delle soluzioni ottime che sono abbastanza vicine ad essere ottime anche dal punto di vista della funzione citata.

7.4.2 Consumo di risorse

Consideriamo ora un esempio di minimizzazione del consumo di risorse. Il problema hard sarà esattamente quello di prima. Per i vincoli, dunque, facciamo riferimento al grafo in figura 7.1. Ricordiamone i dati e aggiungiamo

attività	variabili		durate	consumo risorse
nome	inizio	fine	max e min durata	consumo per 1 unità di tempo
A	2	3	1-6	1
B	4	5	1-6	2
C	0	1	1-12	-
D	6	7	1-12	2
E	8	9	5-5	-
F	10	11	6-6	4
G	12	13	10-10	-
H	14	15	1-1	1

Tabella 7.6: Descrizione delle attività del problema. Sono indicati: il nome delle attività, le variabili corrispondenti, le durate e i consumi.

una colonna indicando il consumo di risorse per unità di tempo per le attività rilevanti da questo punto di vista. Otteniamo così la tabella 7.6.

Descriviamo la funzione globale di preferenza che abbiamo adottato. Prima di tutto indichiamo la funzione che calcola il consumo di risorse di una soluzione:

$$f(s) = \{1(x_3 - x_2) + 2(x_5 - x_4) + 2(x_7 - x_6) + 4(x_{11} - x_{10}) + 1(x_{15} - x_{14})\}.$$

Questa funzione non fa altro che moltiplicare il consumo per l'effettiva durata dell'attività nella soluzione, che si ottiene facendo la differenza tra il valore di fine e quello di inizio. Una prima normalizzazione può essere fatta considerando il caso in cui tutti gli eventi abbiano durata massima [8]. Chiamiamo il consumo di risorse che si avrebbe in quel caso MAX. Allora avremo:

$$\tilde{f}(s) = \frac{f(s)}{MAX}$$

Nel nostro caso, ad esempio, MAX=78. Modificando questa funzione in modo da associare preferenze maggiori a consumi minori, otteniamo la funzione di preferenza globale per questo problema di minimizzazione delle risorse:

$$F(s) = 1 - \tilde{f}(s).$$

Il range di valori associati a tutte le soluzioni del problema, che ricordiamo sono 900, è [0.37,0.6].

Il modulo di apprendimento è stato allenato sempre su 200 esempi del training set e testato su 300 esempi del test set [9]. Per questo esperimento è

[8] Non è detto che esista una soluzione consistente con i vincoli in cui si abbia massima durata per tutte le attività.

[9] Si osservi che solo la cardinalità dei due insiemi coincide con quella del caso precedente, ma le soluzioni negli insiemi sono cambiate.

stato adottato un criterio di stop diverso, ovvero una soglia pari al 0.075 sull'errore quadratico cumulativo non troncato sul training set. Il motivo per cui è stato fatto questo è che il monitoraggio di varie prove dell'apprendimento ha portato a osservare come la discesa di gradiente procedesse a "singhiozzo", fermandosi per molte iterazioni e poi scendendo repentinamente per altrettante. Questa soglia ha forzato il modulo a compiere ben 33945 iterazioni con parametri $\eta = 0.0006$ e parametri $\beta = 8$. Alla fine si è avuto un errore massimo assoluto pari 0.08 ed un errore medio dello 0.02.

Nella tabella 7.7 sono indicate le soluzioni ottime dell'STPP proposto alla fine dal modulo e le preferenze ad esse associate da F.

Questa volta le soluzioni ottime dell'STPP appreso dal modulo sono state indicate tutte. Infatti, lo sforzo di compiere oltre 33000 iterazioni ha permesso di isolare 39 delle 900 soluzioni tutte aventi uno dei tre valori di preferenza migliori rispetto a F. Vediamo cosa si può dire per ciascun criterio di valutazione:

- *Quanto vicini, numericamente, sono i valori proposti dalle funzioni.* Come si può vedere l'errore massimo è pari 0.1 . Il valore ottimo di preferenza dell'STPP è 0.58 e sono state considerate ottime solo soluzioni che rispetto a F hanno valore di preferenza 0.57, 0.58, e 0.6 (ottimo).

- *Quanto buone sono le soluzioni proposte dal modulo.* Le soluzioni proposte sono molto buone rispetto a F. Infatti ricordiamo che il range era $[0.37, 0.6]$, quindi il modulo ha scartato tutte le altre soluzioni a parte quelle con un valore di preferenza tra i tre più alti.

- *Quante soluzioni ottime sono state indentificate dal modulo.* Tutte. Le soluzioni ottime del problema di vincoli hard rispetto alla funzione F corrispondono alle prime 13 righe della tabella e hanno preferenza 0.6.

- *Quante soluzioni ottime erano nel training set e quante, invece, sono state scoperte tramite il modulo.* Solo 4 delle 13 soluzioni ottime rispetto a F erano nel training set, dunque il modulo ha identificato come ottime le rimanenti 9.

Possiamo concludere che in entrambi i casi il modulo è riuscito a isolare se non le soluzioni ottime un insieme contenente le migliori. In entrambi i casi questo insieme conteneva tutte le soluzioni ottime originarie. Da questi risultati si può intuire come si possa utilizzare il modulo per produrre dei sottoproblemi di quello iniziale aventi come soluzioni solo quelle di una certa qualità. Si può anche ipotizzare di poter raffinare tali sottoproblemi applicando più volte l'apprendimento, magari allenandolo su esempi nuovi. Nei nostri piccoli esempi l'esiguità del numero totale delle soluzioni non permette di fare

questo, ma i problemi reali della NASA, su cui si sta cercando di applicare questo metodo hanno fino a 300 variabili e decine di migliaia di soluzioni.

variabili																L. pref	O.pref
0	2	0	1	1	2	0	1	1	6	6	12	0	10	10	12	0.58	0.6
0	2	0	1	1	2	1	2	2	7	7	13	1	11	11	13	0.58	0.6
0	2	0	1	1	2	2	3	3	8	8	14	2	12	12	14	0.58	0.6
0	2	0	1	1	2	3	4	4	9	9	15	3	13	13	15	0.58	0.6
0	2	0	1	1	2	4	5	5	10	10	16	4	14	14	16	0.58	0.6
0	2	0	1	1	2	5	6	6	11	11	17	5	15	15	17	0.58	0.6
0	2	0	1	1	2	6	7	7	12	12	18	6	16	16	18	0.58	0.6
0	2	0	1	1	2	7	8	8	13	13	19	7	17	17	19	0.58	0.6
0	2	0	1	1	2	8	9	9	14	14	20	8	18	18	20	0.58	0.6
0	2	0	1	1	2	9	10	10	15	15	21	9	19	19	21	0.58	0.6
0	2	0	1	1	2	10	11	11	16	16	22	10	20	20	22	0.58	0.6
0	2	0	1	1	2	11	12	12	17	17	23	11	21	21	23	0.58	0.6
0	2	0	1	1	2	12	13	13	18	18	24	12	22	22	24	0.58	0.6
0	3	0	2	2	3	0	1	1	6	6	12	0	10	10	12	0.58	0.58
0	3	0	2	2	3	1	2	2	7	7	13	1	11	11	13	0.58	0.58
0	3	0	2	2	3	2	3	3	8	8	14	2	12	12	14	0.58	0.58
0	3	0	2	2	3	3	4	4	9	9	15	3	13	13	15	0.58	0.58
0	3	0	2	2	3	4	5	5	10	10	16	4	14	14	16	0.58	0.58
0	3	0	2	2	3	5	6	6	11	11	17	5	15	15	17	0.58	0.58
0	3	0	2	2	3	6	7	7	12	12	18	6	16	16	18	0.58	0.58
0	3	0	2	2	3	7	8	8	13	13	19	7	17	17	19	0.58	0.58
0	3	0	2	2	3	8	9	9	14	14	20	8	18	18	20	0.58	0.58
0	3	0	2	2	3	9	10	10	15	15	21	9	19	19	21	0.58	0.58
0	3	0	2	2	3	10	11	11	16	16	22	10	20	20	22	0.58	0.58
0	3	0	2	2	3	11	12	12	17	17	23	11	21	21	23	0.58	0.58
0	3	0	2	2	3	12	13	13	18	18	24	12	22	22	24	0.58	0.58
0	4	0	3	3	4	0	1	1	6	6	12	0	10	10	12	0.58	0.57
0	4	0	3	3	4	1	2	2	7	7	13	1	11	11	13	0.58	0.57
0	4	0	3	3	4	2	3	3	8	8	14	2	12	12	14	0.58	0.57
0	4	0	3	3	4	3	4	4	9	9	15	3	13	13	15	0.58	0.57
0	4	0	3	3	4	4	5	5	10	10	16	4	14	14	16	0.58	0.57
0	4	0	3	3	4	5	6	6	11	11	17	5	15	15	17	0.58	0.57
0	4	0	3	3	4	6	7	7	12	12	18	6	16	16	18	0.58	0.57
0	4	0	3	3	4	7	8	8	13	13	19	7	17	17	19	0.58	0.57
0	4	0	3	3	4	8	9	9	14	14	20	8	18	18	20	0.58	0.57
0	4	0	3	3	4	9	10	10	15	15	21	9	19	19	21	0.58	0.57
0	4	0	3	3	4	10	11	11	16	16	22	10	20	20	22	0.58	0.57
0	4	0	3	3	4	11	12	12	17	17	23	11	21	21	23	0.58	0.57
0	4	0	3	3	4	12	13	13	18	18	24	12	22	22	24	0.58	0.57

Tabella 7.7: Insieme delle soluzioni ottime dell'STPP proposto dal modulo.

Capitolo 8

Conclusioni e direzioni future

In questa tesi è stato proposto un modo per risolvere ed apprendere problemi di vincoli temporali semplici con preferenze. La soluzione di questi problemi è stata sviluppata sia in termini teorici che implementativi. Sono stati inoltre presentati dei risultati sperimentali del risolutore che danno un'idea di cosa comporti risolvere questo tipo di problemi.

Abbiamo poi indagato la possibilità di utilizzare informazioni globali per indurre conoscenze a livello locale da utilizzarsi per trovare una soluzione ottima del problema. A tal fine è stato proposto un modulo di apprendimento che attraverso la tecnica di discesa di gradiente permette di trasformare informazioni sulle soluzioni in preferenze locali tra le variabili. Anche di questo modulo di apprendimento abbiamo presentato alcuni risultati sperimentali.

Riesaminando gli obiettivi che ci eravamo proposti all'inizio di questa ricerca cerchiamo di valutare i risultati ottenuti:

- *Studiare la possibilità di trovare una soluzione di un STPP senza backtracking e in tempo polinomiale.* Nel capitolo 3 abbiamo dimostrato che questo è possibile. Siamo pervenuti a questo risultato analizzando le analogie con gli STPs e gli effetti della consistenza locale sulle funzioni di preferenza.

- *Implementare un risolutore.* Anche questo obiettivo è stato raggiunto. Grazie ai risultati teorici ottenuti abbiamo potuto implementare un risolutore e diverse sue varianti. L'implementazione di un generatore random ha permesso di testare il risolutore e valutarne la performance.

- *Studiare la possibilità e le modalità dell'applicazione di tecniche di apprendimento automatico.* È stata identificata una classe di funzioni di preferenze, le quadratiche convesse, che ha permesso l'impostazione di un modulo di apprendimento automatico. Abbiamo individuato nella tecnica di discesa di gradiente le caratteristiche adatte ad operare

con il nostro tipo di problemi. Abbiamo studiato diverse euristiche con cui applicare questa tecnica e siamo pervenuti all'implementazione di un modulo di apprendimento automatico. Sempre grazie al generatore random è stato possibile testare la qualità del modulo che si è rivelata soddisfacente. Sono state, inoltre, individuate delle applicazioni realistiche, soprattutto nel campo spaziale su cui il modulo ha dato risultati efficaci e promettenti

- *Cercare delle applicazioni reali.* Quanto abbiamo fatto è oggetto di studio e di ulteriori perfezionamenti per applicazioni di tipo spaziale. La più immediata è quella di costruire un tool con modulo e risolutore combinati che fornisca al planner di K9 schedulazioni migliori per la sua futura attività su Marte.

Passiamo ora a descrivere verso quali direzioni si sta muovendo questa ricerca:

- È attualmente in corso, e parzialmente realizzata, l'implementazione di un risolutore più veloce ma meno generale. Il suo funzionamento si basa sulle stesse ipotesi di quello proposto in questa tesi, cioè l'idempotenza dell'operatore moltiplicativo del semianello e la semi-convessità delle funzioni di preferenza. Esso segue sostanzialmente il procedimento suggerito in [17] per la dimostrazione della risolvibilità in tempo polinomiale segli STPPs. Il miglioramento dal punto di vista della performance è ottenuto rinunciando ad una rappresentazione discretizzata sia degli intervalli sia delle funzioni di preferenza. Infatti questo risolutore gestisce vincoli temporali con preferenze rappresentati da una coppia di interi per l'intervallo e da una rappresentazione parametrica delle funzioni. Il procedimento seguito è quello di effettuare una ricerca binaria sul dominio dei livelli di preferenza. Amesso di lavorare con il semianello fuzzy, la ricerca verrà fatta nei reali tra 0 e 1. Se si suppone siano sufficienti 12 cifre dopo la virgola basteranno al più tredici passi della ricerca binaria. Per ogni reale k proposto dalla fase di ricerca, verrà costruito l'STP_k, cioè il sottoproblema temporale semplice avente come intervalli gli insiemi di valori mappati dalle funzioni in preferenze maggiori di k. Verrà poi verificata la consistenza di STP_k. La ricerca permetterà di trovare il massimo k per cui si ottiene un STP consistente. Una volta individuato tale livello, diciamolo *opt*, non rimane che trovare una soluzione di STP_{opt}. Questo risolutore rispetto a quello presentato è molto più veloce e richiede meno memoria. Tuttavia ha il limite di ammettere solo funzioni semi-convesse parametrizzabili. Inoltre la loro forma parametrica deve essere tale da poter sempre trovare i punti di intersezione fra le funzioni e i vari livelli a cui si effettuano i tagli.

- Una volta che sarà finita l'implementazione di questo risolutore si vuole indagare la possibiltà di poter gestire problemi con un'evoluzione incrementale. Questo significherebbe poter gestire problemi in cui il numero delle variabili coinvolte, i vincoli e le funzioni di preferenza mutano nel tempo. Una prima idea è quella di permettere l'aggiunta di variabili e quindi di vincoli. Supponiamo infatti di aver risolto con la tecnica esposta al punto precedente un STPP con certe variabili e supponiamo di voler aggiungere un paio di variabili, che potrebbero rappresentare un nuovo evento da inserire nella schedulazione. A questo punto basterà tagliare le funzioni dei nuovi vincoli al livello ottimo che era stato trovato. Se si otterrà un STP consistente allora le sue soluzioni saranno ottime altrimenti si procederà con una ricerca binaria nelle preferenze tra 0 e quel livello. Questa è solo una prima idea, ma sarà interessante anche vedere come eventualmente scoprire se sostituendo il nuovo evento a qualcuno di quelli già schedulati si può ottenere qualcosa di migliore etc. In futuro, dunque vorremmo porre l'attenzione sull'aspetto dinamico di questi problemi.

- Un argomento che vorremmo approfondire è quello di una possibile *parametrizzazione delle funzioni semi-convesse*. Trovare infatti una formula, o un insieme di formule, che riescano a rappresentare tutte le funzioni di questa classe permetterebbe di considerare nuovi approcci per l'apprendimento automatico e per il risolutore.

- Le due ipotesi fondamentali su cui si basano molti dei risultati da noi ottenuti sono la semi-convessità delle funzioni e l'*idempotenza dell'operatore moltiplicativo* del semianello. In realtà sarebbe interessante capire fino a che punto la proprietà del semianello pesa. Sappiamo che essa è necessaria per garantire l'equivalenza del problema ottenuto da STPP_PC-2, ma non si sa bene che relazione ci sarebbe tra i due problemi, quello di partenza e quello consistente sui cammini, se venisse rilassata questa ipotesi. Inoltre il fatto che solo alcuni semianelli (quello fuzzy è il più usato) soddisfano tale ipotesi è abbastanza restrittivo dal punto di vista pratico. Per esempio nel caso fuzzy, utilizzarlo significa massimizzare il caso pessimo che non è sempre una politica adatta per risolvere i problemi. Consideriamo una soluzione s, abbiamo visto che la sua preferenza viene fissata prendendo il minimo delle preferenze locali associate alle proiezioni che essa individua sui vincoli. Potrebbe capitare che tutte le proiezioni tranne una abbiano livelli molto alti, mentre questa abbia un livello locale pessimo. La soluzione verrebbe quindi considerata pessima anche se in complesso soddisfa con preferenze alte

la maggior parte dei vincoli. Sarà interessante indagare un modo per evitare questi inconvenienti.

- Un'altra applicazione interessante di quanto esposto potrebbe essere nel campo dei vincoli n-ari. Infatti si può interpretare un tale vincolo come una soluzione di un STPP con n variabili, e si può considerare il processo di apprendimento automatico, cioè l'induzione da dei valori globali a valori locali, come una proiezione di vincoli n-ari su vincoli binari. Ci sono risultati teorici su questo argomento riguardanti i vincoli hard di cui sarebbe interessante poter valutare la generalizzazione ai vincoli con preferenze.

- Infine speriamo di poter continuare la finora fruttuosa collaborazione con NASA Ames in modo da poter approfondire le varie possibilità di applicabilità del nostro modello nell'ambito della ricerca spaziale.

Bibliografia

[1] J.F. Allen. Towards a general theory of action and time. Artif.Intell. 23(2)(1984) 123-154.

[2] J.F. Allen. Maintaining knowledge about temporal intervals. Commun. ACM 26 (11) (1983) 832-843.

[3] A. Biso, F. Rossi and A. Sperduti. Experimental Results on Learning Soft Constraints. Proc. KR 2000, Morgan Kaufmann, 2000.

[4] S. Bistarelli, U. Montanari and F. Rossi. Semiring-based Constraint Solving and Optimization. Journal of the ACM, Vol.44, n.2, March 1997.

[5] S. Bistarelli, U. Montanari e F. Rossi. Constraint Solving over Semirings. Proc. IJCAI95. morgan Kaufman, 1995

[6] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. Introduction to Algorithms. MIT press, Cambridge, Ma, 1990.

[7] T.L. Dean and D.V. McDermott. Temporal database management. Artiff. Intell. 32 (1987) 1-55.

[8] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. Artificial Intelligence, Vol. 49, 1991, pp. 61-95.

[9] Rina Dechter. Encyclopedia of Artificial Intelligence, Chapter Constraint Networks. John Wiley&Sons Inc., 2nd edition,1992.

[10] D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraints satisfaction. Proc. IEEE International Conference on Fuzzy Systems, IEEE, 1993.

[11] H. Fargier and J. Lang. Uncertainity in constraint satisfaction problems: a probabilistaic approach. Proc. European Conference on Symbolic and Qualitative Approaches to Reasoning an Uncertainity. Springer-Verlag, LNCS 747, 1993.

[12] E.C. Freuder Synthesizing constraint expressions. Communications of the ACM, 21(11), 1978.

[13] E.C. Freuder backtrack-free and backtrack-bounded search. Kanal and Kumar editors, Search in Artificial Intelligence, Springer-Verlag, 1988.

[14] E.C. Freuder and R.J. Wallace. Partial Constraint Satisfaction. AI Journal, 58, 1992 21-70.

[15] C. Goller. A Connectionist Approach for Learning Search-Control Heuristics for Automated Deduction. Technical University Munich, Computer Science, 1997.

[16] K. Kahn and G.A. Gorry. Mechanizing temporal knowledge. Artif.Intell. 9 (1977) 87-108.

[17] L. Khatib, P. Morris, R. Morris and F. Rossi. Temporal Constraint Reasoning With Preferences. In Proc. IJCAI01. Morgan Kaufman, 2001.

[18] P.B. Ladkin. Metric constraint satisfaction with intervals. Tech. Rept. TR89-038, Intenational Computer Science Institute, Berkeley, CA, (1989).

[19] C.E. Leiserson and J.B. Saxe. A mixed-integer linear programming problem which is efficiently solvable. proceedings 21st Annual Allerton Conference on Communications, Control, and Computing (1983) 204-213.

[20] Y.Z. Liao and C.K. Wang. An algorithm to compact a VLSI compact symbolic layout with mixed constraints. IEEE Trans. Computer-Aided Design of integrated Circuits and Systems 2 (2) (1983) 62-69.

[21] A.K. Mackworth. Consistency in networks of relations. Artif. Intell. 8 (1)(1977) 99-118.

[22] A.K. Mackworth. Constraint satisfaction. In Stuart C. Shapiro. editor, Encyclopedia of AI, volume 1, 285-293. John Wiley&Sons, 1992

[23] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. Artif. Intell. 25 (1)(1985) 65-74.

[24] J. Malik and T.O. Binford. Reasoning in time and space. Proceedings IJCAI-83, Karlsruhe, FRG (1983) 343-345.

[25] K. Marriott and P. Stuckey Programming with Constraints. The MIT press, Cambridge 1998.

[26] D.V. McDermott. A temporal logic for reasoning about processes and plans. Cogn. Sc. 6 (1982) 101-155.

[27] T.M. Mitchell Machine Learning. WCB/McGraw-Hill 1997

[28] U. Montanari. Networks of constraints: fundamental properties and applications to picture processing. Inf. Sci. 7 (1974) 95-132.

[29] U. Montanari e F. Rossi. Constraint relaxation may be perfect. Artificial Intelligence Journal, 48:143-179,1991.

[30] A. Rosenfels, R.A. Hummel and and S.W. Zucker. Scene labelling by relaxations operations. IEEE Transaction on Systems, Man, Cybernetics, 6 (6)(1976)

[31] F. Rossi and A. Sperduti Learning solution preferences in constraint problems. Journal of Experimental and Theoretical Computer Science, 1998. Vol10.

[32] F. Rossi, K.B. Venable, A. Sperduti, L. Khatib, P. Morris, R.Morris. Solving and Learning Soft Temporal Constraints: Experimental Scenario and Examples. Pro. CP2001 Workshop on Soft Constraints (soft'01), Cyprus, 2001.

[33] S. Russell and P. Norvig. Artficial Intelligence: A Modern Approach. Prentice Hall, 1995

[34] Zs Ruttkay. Fuzzy constraint satisfaction. Proc. 3rd International Conference on Fuzzy Systems, 1994.

[35] E. Scwalb, R. Dechter Coping With Disjunctions in Temporal Constraint Satisfaction Poblems. Proc. AAAI'93 127-132

[36] E. Scwalb e L. Vila Temporal Constraints: A Survey. An ICS Technical Report, October 1997.

[37] R. Shostak. Deciding linear inequalities by computing loop residues. J. ACM 28 (4) (1981) 769-779.

[38] R.E. Valdés-Pérez. Spatio-temporal reasoning and linear inequalities. Artificial Intelligence Laboratory, AIM-875, MIT, Cambridge, MA (1986).

[39] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. Proceedings AAAI-86, Philadelphia, PA (1986) 377-382.

Appendice A

Codici

Riportiamo ora i codici dell'implementazione di tutti gli algoritmi.

A.1 Header files

```
/* CLASSE CHE DEFINISCE L'OGETTO VINCOLO
TEPORALE SEMPLICE CON PREFERENZE */


class SoftC{
  public:
int name[2];
   int le;
   int re;
   int x[900];
   float fx[900];
   float a;
   float b;
   float c;
};
```

```
/* CLASSE CHE DEFINISCE L'OGGETTO VINCOLO
TEMPORALE SENZA PREFERENZE */
```

```
class Stp{
 public:
  int sname[2];
  int iends[2];
  int noval;
  Stp(){sname[0]=-1;
  sname[1]=-1;
  iends[0]=-1;
  iends[1]=-1;
  noval=0;};
};




/* CLASSE CHE DEFINISCE L'OGGETTO
TRIANGOLO DI VINCOLI (ELEMENTO DELLA CODA) */


class Path{
   public:
    int I;
    int K;
    int J;
    Path *next;
    Path(){I=-1; K=-1; J=-1; next=NULL;};
    };




/* CLASSE CHE DEFINISCE LA CODA DI TRIANGOLI DI VINCOLI */

class PQueue{
      public:
      Path *head;
      Path *tail;
      };


PQueue make_PQueue(void){
```

```
      PQueue *Q=new PQueue;
      Q->head=Q->tail=NULL;
      return *Q;
      };

Path dePQueue (PQueue & Q){
    Path del;
    if (Q.head==NULL){
       cout<<"PQueue is empty"<<endl;
       return del;
       };
    del=*(Q.head);
    if(Q.head==Q.tail){
       delete Q.head;
       Q.head=Q.tail=NULL;
       }
    else{
         Path *p=Q.head;
         Q.head=Q.head->next;
         delete p;
         };
     return del;
};

void enPQueue(PQueue & Q, int a, int b, int c){
Path *p= new Path;
p->I=a;
p->K=b;
p->J=c;
if(Q.head==NULL) Q.head=Q.tail=p;
else
     Q.tail->next=p;
     Q.tail=p;
};



void print_Path( Path P){
    cout<<"("<<P.I<<","<<P.K<<","<<P.J<<")"<<endl;
   };
```

```
void print_PQueue(PQueue Q){
    Path *temp=Q.head;
    while(temp!=NULL){
       print_Path(*temp);
       temp=temp->next;
       };
    };
```

A.2 Risolutori

```
/* STPP_SOLVER */


SoftC::SoftC(){
  a=0;
  b=0;
  c=0;
  le=0;
  re=0;
  int i;
  name[0]=-1;
  name[1]=-1;
  for(i=0; i<40; i++){
    x[i]=0;
    fx[i]=0;
  };
};

//calcola  y dati a,b,c,x
float parabola (float a, float b, float c, int x){
  float y;
  y=a*(x*x)+b*x+c;
  return y;
};

//gestisce lo standard input
 SoftC build_SoftC(){
  SoftC C ;
```

```
float temp,a,b,c;
int ix, dx, j, h, n;
cout<<"number of the first variable constrained:"<<endl;
cin>>n;
C.name[0]=n;
cout<<"number of the second variable constrained:"<<endl;
cin>>n;
C.name[1]=n;
cout<<"estremo sinistro?\n";
cin>>ix;
cout<<"estremo destro?\n";
cin>>dx;
C.le=0;
C.re=dx-ix;
char ch;
h=ix;
for(j=(C.le); j<=(C.re); j++){
  C.x[j]=h;
  h++;
};
enum iposs {p='p', d='d'};
cout<<"p fo parabolic preference function
d for descrete point to point"<<endl;
cin>>ch;
if(ch==d){

  h=ix;
  for( j=(C.le); j<=(C.re); j++){
    cout<<"x="<<h<<" fx=?\n";
  cin>>temp;
  C.fx[j]=temp;
  h++;
  };
};
if(ch==p){
  cout<<"a?"<<endl;
  cin>>a;
  C.a=a;
  cout<<"b?"<<endl;
  cin>>b;
  C.b=b;
```

```
    cout<<"c?"<<endl;
    cin>>c;
    C.c=c;
    h=ix;
    for(j=C.le; j<=(C.re); j++){
      C.fx[j]=parabola(a,b,c,C.x[j]);
    };
  };
  return C;
 };

//stampa un vincolo temporale con preferenze punto a punto
void print_SoftC (SoftC C){
  ofstream outFile("optsol.out", ios::app);
  int j, h;
  outFile<<C.name[0]<<C.name[1]<<endl;
  h=C.le;
  outFile<<h<<endl;
  outFile<<C.re<<endl;
  for(j=C.le; j<=C.re; j++){
    outFile<<"x="<<C.x[h]<<" fx="<<C.fx[h]<<endl;
    h++;
  };
};

int max(int i, int j){
  if (i>=j) return i;
  else return j;
};
float max(float i, float j){
  if (i>=j) return i;
  else return j;
};
int min(int i, int j){
  if (i<=j) return i;
  else return j;
};
float min(float i, float j){
  if (i<=j) return i;
  else return j;
};
```

```
//questa funzione, usata da compose_SoftC, calcola i valori
//di preferenza per il vincolo composto
float maxmin(int i, SoftC C1, SoftC C2){
  int h, j, cheks, q;
  float mini=0;
  float maxi=0;
  int M=max(C1.re-C1.le,C2.re-C2.le);
  if(i<=M){ h=0;
  j=i;
  };
  if(i>M){ h=i-M;
  j=M;
  };
  cheks=j-h;
  for(q=0; q<=cheks; q++){
    mini=min(C1.fx[h+C1.le], C2.fx[j+C2.le]);
    maxi=max(maxi,mini);
    h++;
    j--;
  };
  return maxi;
};

//calcola il vincolo ottenuto componendo due vincoli
SoftC compose_SoftC (SoftC C1, SoftC C2){
  int i, k;
  SoftC C3;
  C3.name[0]=C1.name[0];
  C3.name[1]=C2.name[1];
  C3.le=0;
  C3.re=(C1.re+C2.re)-(C1.le+C2.le);
  k=C1.x[C1.le]+C2.x[C2.le];
  for(i=0; i<=C3.re; i++){
    C3.x[i]=k;
    C3.fx[i]=maxmin(i, C1, C2);
    k++;
  };
  return C3;
};
```

```
//proietta il vincolo C1 su C2
SoftC project_SoftC (SoftC C1, SoftC C2){
  assert (C1.name[0]==C2.name[0]);
  assert (C1.name[1]==C2.name[1]);
  int i;
  if((C1.x[C1.re]<C2.x[C2.le]) || (C2.x[C2.re]<C1.x[C1.le])){
    cout<<"in here"<<endl;
    C2.le=-1;}
  else {
    while(C2.x[C2.le]<C1.x[C1.le])
      {C2.le++;};
    while(C2.x[C2.le]>C1.x[C1.le])
      {C1.le++;};
    while(C2.x[C2.re]>C1.x[C1.re])
      {C2.re--;};
    while(C2.x[C2.re]<C1.x[C1.re])
      {C1.re--;};
    int j=C1.le ;
    for(i=C2.le; i<=C2.re; i++){
      C2.fx[i]=min(C2.fx[i], C1.fx[j]);
      j++; };
  };
  return C2;
};

//rilassa il triangolo di vincoli IJ, IK, IJ
SoftC relax_triangle (SoftC IJ, SoftC IK, SoftC KJ){
  SoftC Comp;
  Comp=compose_SoftC(IK,KJ);
  if(IJ.name[0]==-1) IJ=Comp;
  else IJ=project_SoftC(Comp,IJ);
  return IJ;
};

//per ogni vincolo IJ calcola il vincolo JI
SoftC specular_SoftC(SoftC C){
  SoftC mirror;
  mirror.name[0]=C.name[1];
  mirror.name[1]=C.name[0];
  mirror.re=C.re-C.le;
  int h=(C.x[C.re]);
```

```
  int q=C.re;
  for(int j=mirror.le; j<=mirror.re; j++){
    mirror.x[j]=-h;
    mirror.fx[j]=C.fx[q];
    q--;
    h--;
  };
  return mirror;
};

//controlla se due vincoli sono uguali o meno
bool notequal_SoftC (SoftC C1, SoftC C2){
  bool b=true;
  if(C1.name[0]!=C2.name[0]) return b;
  if(C1.name[1]!=C2.name[1]) return b;
  if(C1.x[C1.le]!=C2.x[C2.le]) return b;
  if(C1.x[C1.re]!=C2.x[C2.re]) return b;
  int n=C1.re-C1.le;
  for (int i=0; i<=n; i++){
    if(C1.fx[C1.le+i]!=C2.fx[C2.le+i]) return b ;
  };
  b=false;
  return b;
};

//tquesta funzione e' usata da PC2.
//Gestisce la coda
void related_Paths(PQueue & Q, int m, int i,int k, int j){
  int h;
  for (h=0; h<m; h++){
    if (h!=i && h!=j){
      if (h!=k) enPQueue(Q,i,h,j);
      if (i<h) enPQueue(Q,i,j,h);
      if (j<h) enPQueue(Q,j,i,h);
      if (h<j) enPQueue(Q,h,i,j);
      if (h<i) enPQueue(Q,h,j,i);
    };
  };
};

//identifica i sottointervalli con valore ottimo di preferenza
```

```
SoftC restrict_to_max(SoftC C, float M){
  while (C.fx[C.le]<M) { C.le++;};
  while (C.fx[C.re]<M) { C.re--;};
  return C;
};


int main(){                              //INPUT
  int m;
  SoftC * * network;
  network=new SoftC* [100];
  for(int ini=0; ini<100; ini++){
    network[ini]=new SoftC[100];
  };

  char ch, ch1;
  enum possIN {s='s', f='f'};
  cout<<"press s for standard input, f for file input"<<endl;
  cin>>ch1;
  if(ch1==s){                              //standard input
    cout<<"how many variable?"<<endl;
    cin>>m;
    enum poss{y='y', n='n'};
    cout<<"costruire?"<<endl;
    cin>>ch;
    while(ch!=n){
      SoftC C=build_SoftC();
      network[C.name[0]][C.name[1]]=C;
      network[C.name[1]][C.name[0]]=specular_SoftC(C);
      cout<<"Build a constraint? y or n?"<<endl;
      cin>>ch;
   };
 };
  if(ch1==f){                              //file input
    ifstream inFile("tobesol.out", ios::in);
    inFile>>m;
    cout<<"m"<<m<<endl;
    enum poss{y='y', n='n'};
    inFile>>ch;
    while(ch!=n){
      SoftC C ;
```

```
      float temp, a ,b, c, d1, d2, d3;
      int ix, dx, p, h, q;
      inFile>>q;
      C.name[0]=q;
      inFile>>q;
      C.name[1]=q;
      inFile>>ix;
      inFile>>dx;
      C.le=0;
      C.re=dx-ix;
      h=ix;
      for(p=(C.le); p<=(C.re); p++){
C.x[p]=h;
h++;
         };
      char tp;
      enum ifposs{z='z', d='d'}; //z per parabola, d per pti
      inFile>>tp;
      if (tp==d){
        h=ix;
        for( p=(C.le); p<=(C.re); p++){
  inFile>>temp;
  C.fx[p]=temp;
  h++;
};
      };
      if(tp==z){
inFile>>a;
C.a=a;
inFile>>b;
C.b=b;
inFile>>c;
C.c=c;
for(p=(C.le); p<=(C.re); p++){
  float temper;
  temper=parabola(a,b,c,C.x[p]);
  temper=temper*100;
  temper=floor(temper);
  temper=temper/100;
  if (temper>1) temper=1;
  if (temper<0) temper=0;
```

```
  C.fx[p]=temper;
};
      };
      network[C.name[0]][C.name[1]]=C;
      network[C.name[1]][C.name[0]]=specular_SoftC(C);
      inFile>>ch;
    };
  };
  for (int q=0; q<m; q++){  //stampa l'STPP iniziale
    for(int p=1; p<m; p++){
      if (q<p) print_SoftC(network[q][p]);
    };
  };
  cout<<"loading terminated"<<endl;
  PQueue Q= make_PQueue();       //STPP_PC2
  int i, j, k;
  for(i=0; i<m; i++){
    for(j=1; j<m; j++){
      for(k=0; k<m; k++){
if (i<j && k!=i && k!=j){
  enPQueue(Q, i, k, j);
};
      };
    };
  };
  while(Q.head!=NULL){
    Path P;
    P=dePQueue(Q);
    if (network[P.I][P.K].name[0]!=-1 &&
        network[P.K][P.J].name[0]!=-1){
      if(notequal_SoftC(network[P.I][P.J],
         relax_triangle(network[P.I][P.J],
        network[P.I][P.K], network[P.K][P.J]))){
network[P.I][P.J]=relax_triangle(network[P.I][P.J],
        network[P.I][P.K],
        network[P.K][P.J]);
network[P.J][P.I]=specular_SoftC(network[P.I][P.J]);
if (network[P.I][P.J].le==-1) {
  print_SoftC(network[P.I][P.J]);
  cout<<"empty interval! inconsistency"<<endl;
  return 0;};
```

```
related_Paths(Q, m, P.I, P.K, P.J);
      };
    };
  };
  for (int q=0; q<m; q++){
    for(int p=1; p<m; p++){
      if (q<p) print_SoftC(network[q][p]);
    };
  };
                        //costruisce l'STP corrispondente
                        //al livello ottimo di preferenza

  float MAX=0;
  for (i=network[0][1].le; i<=network[0][1].re; i++){
    if (MAX<network[0][1].fx[i]) MAX=network[0][1].fx[i];
  };
  Stp * * minimalnet;
  minimalnet=new Stp* [100];
  for(int minit=0; minit<100; minit++){
  minimalnet[minit]= new Stp[100];
  };
  SoftC OM;
  for(i=0; i<m; i++){
    for(j=1; j<m; j++){
      if(i<j){
minimalnet[i][j].sname[0]=network[i][j].name[0];
minimalnet[i][j].sname[1]=network[i][j].name[1];
OM=restrict_to_max(network[i][j],MAX);
minimalnet[i][j].iends[0]=OM.x[OM.le];
minimalnet[i][j].iends[1]=OM.x[OM.re];
minimalnet[i][j].noval=OM.re-OM.le+1;
minimalnet[j][i].sname[0]=j;
minimalnet[j][i].sname[1]=i;
minimalnet[j][i].iends[0]=-(minimalnet[i][j].iends[1]);
minimalnet[j][i].iends[1]=-(minimalnet[i][j].iends[0]);
minimalnet[j][i].noval=minimalnet[i][j].noval;
      };
    };
  };
  int solution[m]; //calcola e stampa la soluzione backtrack_free
  solution[0]=0;
```

```
  j=1;
  while(j<m){
    solution[j]=minimalnet[0][j].iends[1];
    j++;
  };
  ofstream outfile("optsol.out", ios::app);
  outfile<<"back-track-free computed solution: "<<endl;
  for(k=0; k<m; k++){
    outfile<<" "<<solution[k];
  };
  outfile<<endl;
  for(  i=0; i<100; i++){
    delete network[i];
    delete minimalnet[i];
  };
};




/* STPP_COMPLETE_SOLVER */

SoftC::SoftC(){
  a=0;
  b=0;
  c=0;
  le=0;
  re=0;
  int i;
  name[0]=-1;
  name[1]=-1;
  for(i=0; i<40; i++){
    x[i]=0;
    fx[i]=0;
  };
};

//calcola  y dati a,b,c,x
float parabola (float a, float b, float c, int x){
  float y;
```

```
  y=a*(x*x)+b*x+c;
  return y;
};

//gestisce lo standard input
 SoftC build_SoftC(){
  SoftC C ;
  float temp,a,b,c;
  int ix, dx, j, h, n;
  cout<<"number of the first variable constrained:"<<endl;
  cin>>n;
  C.name[0]=n;
  cout<<"number of the second variable constrained:"<<endl;
  cin>>n;
  C.name[1]=n;
  cout<<"estremo sinistro?\n";
  cin>>ix;
  cout<<"estremo destro?\n";
  cin>>dx;
  C.le=0;
  C.re=dx-ix;
  char ch;
  h=ix;
  for(j=(C.le); j<=(C.re); j++){
    C.x[j]=h;
    h++;
  };
  enum iposs {p='p', d='d'};
  cout<<"p fo parabolic preference function
  d for descrete point to point"<<endl;
  cin>>ch;
  if(ch==d){
    h=ix;
    for( j=(C.le); j<=(C.re); j++){
      cout<<"x="<<h<<" fx=?\n";
    cin>>temp;
    C.fx[j]=temp;
    h++;
    };
  };
  if(ch==p){
```

```
    cout<<"a?"<<endl;
    cin>>a;
    C.a=a;
    cout<<"b?"<<endl;
    cin>>b;
    C.b=b;
    cout<<"c?"<<endl;
    cin>>c;
    C.c=c;
    h=ix;
    for(j=C.le; j<=(C.re); j++){
      C.fx[j]=parabola(a,b,c,C.x[j]);
    };
  };
  return C;
 };

//stampa un vincolo temporale con preferenze punto a punto
void print_SoftC (SoftC C){
  ofstream outFile("optsol.out", ios::app);
  int j, h;
  outFile<<C.name[0]<<C.name[1]<<endl;
  h=C.le;
  outFile<<h<<endl;
  outFile<<C.re<<endl;
  for(j=C.le; j<=C.re; j++){
    outFile<<"x="<<C.x[h]<<" fx="<<C.fx[h]<<endl;
    h++;
  };
};

int max(int i, int j){
  if (i>=j) return i;
  else return j;
};
float max(float i, float j){
  if (i>=j) return i;
  else return j;
};
int min(int i, int j){
  if (i<=j) return i;
```

```
  else return j;
};
float min(float i, float j){
  if (i<=j) return i;
  else return j;
};

//questa funzione, usata da compose_SoftC, calcola i valori
//di preferenza per il vincolo composto
float maxmin(int i, SoftC C1, SoftC C2){
  int h, j, cheks, q;
  float mini=0;
  float maxi=0;
  int M=max(C1.re-C1.le,C2.re-C2.le);
  if(i<=M){ h=0;
  j=i;
  };
  if(i>M){ h=i-M;
  j=M;
  };
  cheks=j-h;
  for(q=0; q<=cheks; q++){
    mini=min(C1.fx[h+C1.le], C2.fx[j+C2.le]);
    maxi=max(maxi,mini);
    h++;
    j--;
  };
  return maxi;
};

//calcola il vincolo ottenuto componendo due vincoli
SoftC compose_SoftC (SoftC C1, SoftC C2){
  int i, k;
  SoftC C3;
  C3.name[0]=C1.name[0];
  C3.name[1]=C2.name[1];
  C3.le=0;
  C3.re=(C1.re+C2.re)-(C1.le+C2.le);
  k=C1.x[C1.le]+C2.x[C2.le];
  for(i=0; i<=C3.re; i++){
    C3.x[i]=k;
```

```
    C3.fx[i]=maxmin(i, C1, C2);
    k++;
  };
  return C3;
};

//proietta il vincolo C1 su C2
SoftC project_SoftC (SoftC C1, SoftC C2){
  assert (C1.name[0]==C2.name[0]);
  assert (C1.name[1]==C2.name[1]);
  int i;
  if((C1.x[C1.re]<C2.x[C2.le]) || (C2.x[C2.re]<C1.x[C1.le])){
    cout<<"in here"<<endl;
    C2.le=-1;}
  else {
    while(C2.x[C2.le]<C1.x[C1.le])
      {C2.le++;};
    while(C2.x[C2.le]>C1.x[C1.le])
      {C1.le++;};
    while(C2.x[C2.re]>C1.x[C1.re])
      {C2.re--;};
    while(C2.x[C2.re]<C1.x[C1.re])
      {C1.re--;};
    int j=C1.le ;
    for(i=C2.le; i<=C2.re; i++){
      C2.fx[i]=min(C2.fx[i], C1.fx[j]);
      j++; };
  };
  return C2;
};

//rilassa il triangolo di vincoli IJ, IK, IJ
SoftC relax_triangle (SoftC IJ, SoftC IK, SoftC KJ){
  SoftC Comp;
  Comp=compose_SoftC(IK,KJ);
  if(IJ.name[0]==-1) IJ=Comp;
  else IJ=project_SoftC(Comp,IJ);
  return IJ;
};

//per ogni vincolo IJ calcola il vincolo JI
```

```
SoftC specular_SoftC(SoftC C){
  SoftC mirror;
  mirror.name[0]=C.name[1];
  mirror.name[1]=C.name[0];
  mirror.re=C.re-C.le;
  int h=(C.x[C.re]);
  int q=C.re;
  for(int j=mirror.le; j<=mirror.re; j++){
    mirror.x[j]=-h;
    mirror.fx[j]=C.fx[q];
    q--;
    h--;
  };
  return mirror;
};

//controlla se due vincoli sono uguali o meno
bool notequal_SoftC (SoftC C1, SoftC C2){
  bool b=true;
  if(C1.name[0]!=C2.name[0]) return b;
  if(C1.name[1]!=C2.name[1]) return b;
  if(C1.x[C1.le]!=C2.x[C2.le]) return b;
  if(C1.x[C1.re]!=C2.x[C2.re]) return b;
  int n=C1.re-C1.le;
  for (int i=0; i<=n; i++){
    if(C1.fx[C1.le+i]!=C2.fx[C2.le+i]) return b ;
  };
  b=false;
  return b;
};

//questa funzione e' usata da STPP_PC2.
//gestisce la coda
void related_Paths(PQueue & Q, int m, int i,int k, int j){
  int h;
  for (h=0; h<m; h++){
    if (h!=i && h!=j){
      if (h!=k) enPQueue(Q,i,h,j);
      if (i<h) enPQueue(Q,i,j,h);
      if (j<h) enPQueue(Q,j,i,h);
      if (h<j) enPQueue(Q,h,i,j);
```

```
      if (h<i) enPQueue(Q,h,j,i);
    };
  };
};

//identifica i sottointervalli con valore ottimo di preferenza
SoftC restrict_to_max(SoftC C, float M){
  while (C.fx[C.le]<M) { C.le++;};
  while (C.fx[C.re]<M) { C.re--;};
  return C;
};

//questa funzione utilizzata per
//la ricerca di tutte le soluzioni ottime
//controlla se il valore scelto
//per la variabili e' compatibile
//con i vincoli precedenti
bool Pr_all_sat(int *S, int var, int nvarval, Stp * * minimalnet){
  int i;
  for(i=0; i<=var-1; i++){
     if((nvarval-S[i])<minimalnet[i][var].iends[0] ||
       (nvarval-S[i])>minimalnet[i][var].iends[1]){
      cout<<"non soddisfa vinc prec"<<endl;
      return false;};
  };
  return true;
};

//funzione che calcola il valore di
//preferenza locale di una soluzione (con max-min)
float drater(int *S, int m, SoftC * * network){
  float rating=1;
  int xi, xj, index;
  for (int i=0; i<m; i++){
    for (int j=1; j<m; j++){
      if(i<j){
xi=S[i];
xj=S[j];
index=((xj-xi)-network[i][j].x[network[i][j].le]+network[i][j].le);
if(rating>network[i][j].fx[index]){
  rating=network[i][j].fx[index];
```

```
};

      };
    };
  };
  return rating;
};

//trova tutte le soluzioni ottime
//(implementazione ricorsiva di depth first search)
void S_builder(int var,int *S,int m,
               Stp * * minimalnet,int & solnum,
                SoftC * * network){
  int h,k;
  fstream sio( "optsol.out", ios::app);
  if(var==m){
    solnum++;
    sio<<solnum;
    for (h=0; h<m; h++){
      sio<<" "<<S[h];
    };
    float rating=drater(S,m,network);
    sio<<" "<<rating;
    sio<<endl;
    }
  else{
    for(k=minimalnet[var-1][var].iends[0];
        k<=minimalnet[var-1][var].iends[1]; k++){
      int nvarval=k+S[var-1];
      if(Pr_all_sat(S,var,nvarval,minimalnet)){
S[var]=nvarval;
S_builder(var+1,S,m, minimalnet,solnum,network);
      };
    };
  };
};


int main(){                              //INPUT
  int m;
  SoftC * * network;
```

```
network=new SoftC* [100];
for(int ini=0; ini<100; ini++){
  network[ini]=new SoftC[100];
};
char ch, ch1;
enum possIN {s='s', f='f'};
cout<<"press s for standard input, f for file input"<<endl;
cin>>ch1;
if(ch1==s){                              //standard input
  cout<<"how many variable?"<<endl;
  cin>>m;
  enum poss{y='y', n='n'};
  cout<<"costruire?"<<endl;
  cin>>ch;
  while(ch!=n){
    SoftC C=build_SoftC();
    network[C.name[0]][C.name[1]]=C;
    network[C.name[1]][C.name[0]]=specular_SoftC(C);
    cout<<"Build a constraint? y or n?"<<endl;
    cin>>ch;
   };
};
if(ch1==f){                               //file input
  ifstream inFile("tobesol.out", ios::in);
  inFile>>m;
  cout<<"m"<<m<<endl;
  enum poss{y='y', n='n'};
  inFile>>ch;
  while(ch!=n){
    SoftC C ;
    float temp, a ,b, c, d1, d2, d3;
    int ix, dx, p, h, q;
    inFile>>q;
    C.name[0]=q;
    inFile>>q;
    C.name[1]=q;
    inFile>>ix;
    inFile>>dx;
    C.le=0;
    C.re=dx-ix;
    h=ix;
```

```
      for(p=(C.le); p<=(C.re); p++){
C.x[p]=h;
h++;
         };
      char tp;
      enum ifposs{z='z', d='d'};   //z per parabola, d per pti
      inFile>>tp;
      if (tp==d){
        h=ix;
        for( p=(C.le); p<=(C.re); p++){
  inFile>>temp;
  C.fx[p]=temp;
  h++;
};
      };
      if(tp==z){
inFile>>a;
C.a=a;
inFile>>b;
C.b=b;
inFile>>c;
C.c=c;
for(p=(C.le); p<=(C.re); p++){
  float temper;
  temper=parabola(a,b,c,C.x[p]);
  temper=temper*100;
  temper=floor(temper);
  temper=temper/100;
  if (temper>1) temper=1;
  if (temper<0) temper=0;
  C.fx[p]=temper;
};
      };
      network[C.name[0]][C.name[1]]=C;
      network[C.name[1]][C.name[0]]=specular_SoftC(C);
      inFile>>ch;
    };
  };
  for (int q=0; q<m; q++){ //stampa l'STPP iniziale
    for(int p=1; p<m; p++){
      if (q<p) print_SoftC(network[q][p]);
```

```
    };
  };
  cout<<"loading terminated"<<endl;
  PQueue Q= make_PQueue();     /STPP_PC2
  int i, j, k;
  for(i=0; i<m; i++){
    for(j=1; j<m; j++){
      for(k=0; k<m; k++){
if (i<j && k!=i && k!=j){
  enPQueue(Q, i, k, j);
};
      };
    };
  };
  while(Q.head!=NULL){
    Path P;
    P=dePQueue(Q);
    if (network[P.I][P.K].name[0]!=-1 &&
        network[P.K][P.J].name[0]!=-1){
      if(notequal_SoftC(network[P.I][P.J],
         relax_triangle(network[P.I][P.J],
         network[P.I][P.K], network[P.K][P.J]))){
network[P.I][P.J]=relax_triangle(network[P.I][P.J],
        network[P.I][P.K], network[P.K][P.J]);
network[P.J][P.I]=specular_SoftC(network[P.I][P.J]);
if (network[P.I][P.J].le==-1) {
  print_SoftC(network[P.I][P.J]);
  cout<<"empty interval! inconsistency"<<endl;
  return 0;};
related_Paths(Q, m, P.I, P.K, P.J);
      };
    };
  };
  for (int q=0; q<m; q++){
    for(int p=1; p<m; p++){
      if (q<p) print_SoftC(network[q][p]);
    };
  };
  cout<<"OUT OF PC"<<endl;
  float MAX=0;             //costruisce l' STP corrispondente
  //al livello ottimo di preferenza
```

```
  for (i=network[0][1].le; i<=network[0][1].re; i++){
    if (MAX<network[0][1].fx[i]) MAX=network[0][1].fx[i];
  };
  Stp * * minimalnet;
  minimalnet=new Stp* [100];
  for(int minit=0; minit<100; minit++){
     minimalnet[minit]= new Stp[100];
  };
  SoftC OM;
  for(i=0; i<m; i++){
    for(j=1; j<m; j++){
      if(i<j){
minimalnet[i][j].sname[0]=network[i][j].name[0];
minimalnet[i][j].sname[1]=network[i][j].name[1];
OM=restrict_to_max(network[i][j],MAX);
minimalnet[i][j].iends[0]=OM.x[OM.le];
minimalnet[i][j].iends[1]=OM.x[OM.re];
minimalnet[i][j].noval=OM.re-OM.le+1;
minimalnet[j][i].sname[0]=j;
minimalnet[j][i].sname[1]=i;
minimalnet[j][i].iends[0]=-(minimalnet[i][j].iends[1]);
minimalnet[j][i].iends[1]=-(minimalnet[i][j].iends[0]);
minimalnet[j][i].noval=minimalnet[i][j].noval;
      };
    };
  };
  int solution[m];
  solution[0]=0;
  j=1;
  while(j<m){
    solution[j]=minimalnet[0][j].iends[1];
    j++;
  };
  ofstream outfile("optsol.out", ios::app);
  outfile<<"back-track-free computed solution: "<<endl;
  for(k=0; k<m; k++){
    outfile<<" "<<solution[k];
  };
  outfile<<endl;
  int solnum=0; //calcola e stampa tutte le soluzioni ottime
  for (i=minimalnet[0][1].iends[0];
```

```
       i<=minimalnet[0][1].iends[1]; i++)
    {
    int S[m];
    S[0]=0;
    S[1]=i;
    S_builder(2,S,m,minimalnet,solnum, network);
    };
  for(i=0; i<100; i++){
    delete network[i];
    delete minimalnet[i];
   };
   };
};



/* STPP_RANDOM_SOLVER */

#include<iostream.h>
#include <sys/types.h>
#include <time.h>
#include <stdlib.h>
#include <fstream.h>
#include <math.h>
#include <assert.h>
#include "SoftC.h"
#include "PQueue.h"
#include "Stp.h"

#define a15 452807053
#define mask 2147483647
#define MAXINT 2147483647
#define RANDF(range,seed)range*(float)(seed-MAXINT/2)/MAXINT
#define RANDF2(range,seed)(int) rint(range*(float)seed/MAXINT)

//generatore di numeri random
int clock_seed(){
  int seed;
  unsigned int st;
  time_t times;
  times= time(0);
```

```
  st=(unsigned int) times / 2;
  seed=(int) st;
  return( ((seed%2) == 0) ? seed+1 : seed );
};
int rnd7 (int IX)
{
  IX=IX * a15;
  return( (IX<0)? IX & mask : IX);
};

int clock_rand( double R){
  int j;
  j=clock_seed();
  j=rnd7(j);
  float a = RANDF(R,j);
  int g= (rint(a));
  return g;
};


//costruttore
SoftC::SoftC(){
  a=0;
  b=0;
  c=0;
  le=0;
  re=0;
  int i;
  name[0]=-1;
  name[1]=-1;
  for(i=0; i<40; i++){
    x[i]=0;
    fx[i]=0;
  };
};

//calcola y dati a,b,c, e x
float parabola (float a, float b, float c, int x){
  float y;
  y=a*(x*x)+b*x+c;
  return y;
```

```
};

//calcola y con al parabola troncata
float fuzzy_parabola (float a, float b, float c, int x){
  float y;
  y=a*(x*x)+b*x+c;
  if(y>1) y=1;
  if (y<0) y=0;
  y=y*100;
  y=floor(y);
  y=y/100;
  return y;
};

//gestisce lo standard input
 SoftC build_SoftC(){
  SoftC C ;
  float temp,a,b,c;
  int ix, dx, j, h, n;
  cout<<"number of the first variable constrained:"<<endl;
  cin>>n;
  C.name[0]=n;
  cout<<"number of the second variable constrained:"<<endl;
  cin>>n;
  C.name[1]=n;
  cout<<"estremo sinistro?\n";
  cin>>ix;
  cout<<"estremo destro?\n";
  cin>>dx;
  C.le=0;
  C.re=dx-ix;
  char ch;
  h=ix;
  for(j=(C.le); j<=(C.re); j++){
    C.x[j]=h;
    h++;
  };
  enum iposs {p='p', d='d'};
  cout<<"p fo parabolic preference function
  d for descrete point to point"<<endl;
```

```
  cin>>ch;
  if(ch==d){
    h=ix;
    for( j=(C.le); j<=(C.re); j++){
      cout<<"x="<<h<<" fx=?\n";
      cin>>temp;
      C.fx[j]=temp;
      h++;
    };
  };
  if(ch==p){
    cout<<"a?"<<endl;
    cin>>a;
    C.a=a;
    cout<<"b?"<<endl;
    cin>>b;
    C.b=b;
    cout<<"c?"<<endl;
    cin>>c;
    C.c=c;
    h=ix;
    for(j=C.le; j<=(C.re); j++){
      C.fx[j]=parabola(a,b,c,C.x[j]);
    };
  };
  return C;
 };

void print_SoftC (SoftC C){
  ofstream outFile("leatrunc.out", ios::app);
  int j, h;
  outFile<<C.name[0]<<C.name[1]<<endl;
  h=C.le;
  outFile<<h<<endl;
  outFile<<C.re<<endl;
  for(j=C.le; j<=C.re; j++){
    outFile<<"x="<<C.x[h]<<" fx="<<C.fx[h]<<endl;
    h++;
  };
};
```

```
int max(int i, int j){
  if (i>=j) return i;
  else return j;
};
float max(float i, float j){
  if (i>=j) return i;
  else return j;
};
int min(int i, int j){
  if (i<=j) return i;
  else return j;
};
float min(float i, float j){
  if (i<=j) return i;
  else return j;
};

//funzione che calcola il modulo di un reale
float modulo(float g){
  if(g>=0) return g;
  else return -g;
};

//questa funzione, usata da compose_SoftC, calcola i valori
//di preferenza per il vincolo composto
float maxmin(int i, SoftC C1, SoftC C2){
  int h, j, cheks, q;
  float mini=0;
  float maxi=0;
  int M=max(C1.re-C1.le,C2.re-C2.le);
  if(i<=M){ h=0;
  j=i;
  };
  if(i>M){ h=i-M;
  j=M;
  };
  cheks=j-h;
  for(q=0; q<=cheks; q++){
    mini=min(C1.fx[h+C1.le], C2.fx[j+C2.le]);
    maxi=max(maxi,mini);
    h++;
```

```
    j--;
  };
  return maxi;
};

//calcola il vincolo ottenuto componendo due vincoli
SoftC compose_SoftC (SoftC C1, SoftC C2){
  int i, k;
  SoftC C3;
  C3.name[0]=C1.name[0];
  C3.name[1]=C2.name[1];
  C3.le=0;
  C3.re=(C1.re+C2.re)-(C1.le+C2.le);
  k=C1.x[C1.le]+C2.x[C2.le];
  for(i=0; i<=C3.re; i++){
    C3.x[i]=k;
    C3.fx[i]=maxmin(i, C1, C2);
    k++;
  };
  return C3;
};

//proietta il vincolo C1 su C2
SoftC project_SoftC (SoftC C1, SoftC C2){
  assert (C1.name[0]==C2.name[0]);
  assert (C1.name[1]==C2.name[1]);
  int i;
  if((C1.x[C1.re]<C2.x[C2.le]) ||
 (C2.x[C2.re]<C1.x[C1.le])){C2.le=-1;}
  else {
    while(C2.x[C2.le]<C1.x[C1.le])
      {C2.le++;};
    while(C2.x[C2.le]>C1.x[C1.le])
      {C1.le++;};
    while(C2.x[C2.re]>C1.x[C1.re])
      {C2.re--;};
    while(C2.x[C2.re]<C1.x[C1.re])
      {C1.re--;};
    int j=C1.le ;
    for(i=C2.le; i<=C2.re; i++){
      C2.fx[i]=min(C2.fx[i], C1.fx[j]);
```

```
      j++; };
  };
  return C2;
};

//rilassa il triangolo di vincoli IJ, IK, IJ
SoftC relax_triangle (SoftC IJ, SoftC IK, SoftC KJ){
  SoftC Comp;
  Comp=compose_SoftC(IK,KJ);
  if(IJ.name[0]==-1) IJ=Comp;
  else IJ=project_SoftC(Comp,IJ);
  return IJ;
};

//per ogni vincolo IJ calcola il vincolo JI
SoftC specular_SoftC(SoftC C){
  SoftC mirror;
  mirror.name[0]=C.name[1];
  mirror.name[1]=C.name[0];
  mirror.re=C.re-C.le;
  int h=(C.x[C.re]);
  int q=C.re;
  for(int j=mirror.le; j<=mirror.re; j++){
    mirror.x[j]=-h;
    mirror.fx[j]=C.fx[q];
    q--;
    h--;
  };
  return mirror;
 };

//controlla se due vincoli sono uguali o meno
bool notequal_SoftC (SoftC C1, SoftC C2){
  bool b=true;
   if(C1.name[0]!=C2.name[0]) return b;
   if(C1.name[1]!=C2.name[1]) return b;
   if(C1.x[C1.le]!=C2.x[C2.le]) return b;
   if(C1.x[C1.re]!=C2.x[C2.re]) return b;
   int n=C1.re-C1.le;
   for (int i=0; i<=n; i++){
     if(C1.fx[C1.le+i]!=C2.fx[C2.le+i]) return b ;
```

```
   };
   b=false;
   return b;
};

//questa funzione e' usata da STPP_PC2.
//gestisce la coda
void related_Paths(SoftC net[40][40], PQueue & Q,
 int m, int i,int k, int j){
  int h;
  for (h=0; h<m; h++){
    if (h!=i && h!=j){
      if (h!=k) enPQueue(Q,i,h,j);
      if (i<h) enPQueue(Q,i,j,h);
      if (j<h) enPQueue(Q,j,i,h);
      if (h<j) enPQueue(Q,h,i,j);
      if (h<i) enPQueue(Q,h,j,i);
    };
  };
};

//identifica i sottointervalli
//con valore ottimo di preferenza
SoftC restrict_to_max(SoftC C, float M){
  while (C.fx[C.le]<M) { C.le++;};
  while (C.fx[C.re]<M) { C.re--;};
  return C;
};

//questa funzione utilizzata per
//la ricerca di tutte le soluzioni ottime
//controlla se il valore
//scelto per la variabili e' compatibile
//con i vincoli precedenti
bool Pr_all_sat(int *S, int var,
int nvarval, Stp minimalnet[40][40]){
  int i;
  for(i=0; i<=var-1; i++){
    if((nvarval-S[i])<minimalnet[i][var].iends[0] ||
    (nvarval-S[i])>minimalnet[i][var].iends[1]){
      return false;};
```

```
  };
  return true;
};

//questa funzione \‘{e} quella
//utilizzata per gli esperimenti NASA
//essa volta per volta conterra’
//la funzione di preferenza globale
float userdrater(int *S, int m){
  float rating; //esempio di consumo di risorse
  float temp1=(S[3]-S[2])+(S[15]-S[14]);
  float temp2=2*(S[5]-S[4])+2*(S[7]-S[6]);
  float temp3=4*(S[11]-S[10]);
  rating=(temp1+temp2+temp3)/78;
  rating=1-rating;
  rating=rating*1.5;
  rating=rating*100;
  rating=floor(rating);
  rating=rating/100;
  rating=rating*2;
  return rating;
};

//funzione che calcola il valore di preferenza locale
//di una soluzione (con max-min)
float drater(int *S, int m, SoftC network [40][40]){
  float rating=1;
  int xi, xj, index;
  for (int i=0; i<m; i++){
    for (int j=1; j<m; j++){
      if(i<j){
xi=S[i];
xj=S[j];
index=((xj-xi)-network[i][j].x[network[i][j].le]
          +network[i][j].le);
         if(rating>network[i][j].fx[index]){
   rating=network[i][j].fx[index];
 };
      };
    };
   };
```

```
  return rating;
};

//controlla se due soluzioni coincidono
bool match (int * S, int * T, int m){
  for (int i=0; i<m; i++){
    if (S[i]!=T[i]) return false;
  };
  return true;
};

//controlla se la soluzione proposta e' gia' nel training set
bool notapp(int *S, float trai[][41], int nex, int m){
  int e;
  for(e=0; e<nex; e++){
    int T[m];
    for (int i=0; i<m; i++){
      T[i]=trai[e][i];
    };
    if (match(S,T,m)) return false;
  };
  return true;
};


int main(){
 float oldrating=0;
 float thres=0.5;
 float tempthres=0;
 int m;
 SoftC network[40][40];
 char ch, ch1;
 enum possIN {s='s', f='f'};
 cout<<"press s for standard input, f for file input"<<endl;
 cin>>ch1;
 if(ch1==s){
   cout<<"how many variable?"<<endl;
   cin>>m;
   enum poss{y='y', n='n'};
   cout<<"costruire?"<<endl;
   cin>>ch;
```

```
   while(ch!=n){
      SoftC C=build_SoftC();
      network[C.name[0]][C.name[1]]=C;
      network[C.name[1]][C.name[0]]=specular_SoftC(C);
      cout<<"Build a constraint? y or n?"<<endl;
      cin>>ch;
     };
   };
 if(ch1==f){
   ifstream inFile("tobesol.out", ios::in);
   inFile>>m;
   enum poss{y='y', n='n'};
   inFile>>ch;
   while(ch!=n){
        SoftC C ;
        float temp, a ,b, c, d1 , d2 , d3;
        int ix, dx, p, h, q;
        inFile>>q;
        C.name[0]=q;
        inFile>>q;
        C.name[1]=q;
        inFile>>ix;
        inFile>>dx;
        C.le=0;
        C.re=dx-ix;
        h=ix;
        for(p=(C.le); p<=(C.re); p++){
           C.x[p]=h;
           h++;
         };
        char tp;
        enum ifposs{z='z', d='d'};//z per parabola, d per pti
        inFile>>tp;
        if (tp==d){
        h=ix;
        for( p=(C.le); p<=(C.re); p++){
           inFile>>temp;
           C.fx[p]=temp;
           h++;
           };
};
```

```
if(tp==z){
          inFile>>a;
          C.a=a;
          inFile>>b;
          C.b=b;
          inFile>>c;
          C.c=c;
          for(p=(C.le); p<=(C.re); p++){
    C.fx[p]=fuzzy_parabola(a,b,c,C.x[p]);
  };
};
        network[C.name[0]][C.name[1]]=C;
        network[C.name[1]][C.name[0]]=specular_SoftC(C);
       inFile>>ch;
     };
   };

 for (int q=0; q<m; q++){   //rete non rilassata
   for(int p=1; p<m; p++){
        if (q<p) print_SoftC(network[q][p]);
            };
        };
//stampa sul file tnet.out i vincoli hard
//per il modulo di apprendimento
int i, j, k;
fstream outnet("tnet.out", ios::app);
for(i=0; i<m; i++){
   for(j=1; j<m; j++){
     if(i<j){
       outnet<<'y'<<endl;
     outnet<<network[i][j].name[0]<<" "
     <<network[i][j].name[1]<<endl;
     outnet<<network[i][j].x[network[i][j].le] <<" "<<
     network[i][j].x[network[i][j].re]<<endl;
     };
};
};
 outnet<<'n'<<endl;   //STPP_PC2
PQueue Q= make_PQueue();
for(i=0; i<m; i++){
   for(j=1; j<m; j++){
```

```
       for(k=0; k<m; k++){
           if (i<j && k!=i && k!=j){
                   enPQueue(Q, i, k, j);
                              };
                          };
                      };
                   };
print_PQueue(Q);
while(Q.head!=NULL){
Path P;
P=dePQueue(Q);
if (network[P.I][P.K].name[0]!=-1 &&
network[P.K][P.J].name[0]!=-1){
if(notequal_SoftC(network[P.I][P.J],
relax_triangle(network[P.I][P.J],
network[P.I][P.K], network[P.K][P.J]))){
network[P.I][P.J]=relax_triangle(network[P.I][P.J],
network[P.I][P.K], network[P.K][P.J]);
network[P.J][P.I]=specular_SoftC(network[P.I][P.J]);
if (network[P.I][P.J].le==-1)
{ cout<<"empty interval! inconsistency"<<endl;
                                return 0;};
related_Paths(network, Q, m, P.I, P.K, P.J);
   };
 };
};
 cout<<network[0][1].fx[0]<<endl;
for (int q=0; q<m; q++){
   for(int p=1; p<m; p++){
        if (q<p) print_SoftC(network[q][p]);
            };
        }; //QUI INIZA A GENERARE TRAINING SET
 fstream insol("trset.out", ios::app);
 int nex;              //nex=numero esempi del training set
 cout<<"how many examples would you like to generate
 for the training set?"<<endl;
 cin>>nex;
 Stp minimalnet[40][40];
 SoftC OM;
 for(i=0; i<m; i++){
   for(j=1; j<m; j++){
```

```
    if(i<j){
      minimalnet[i][j].sname[0]=network[i][j].name[0];
      minimalnet[i][j].sname[1]=network[i][j].name[1];
      minimalnet[i][j].iends[0]=network[i][j].x[network[i][j].le];
      minimalnet[i][j].iends[1]=network[i][j].x[network[i][j].re];
      minimalnet[i][j].noval=network[i][j].re-network[i][j].le+1;
      minimalnet[j][i].sname[0]=j;
      minimalnet[j][i].sname[1]=i;
      minimalnet[j][i].iends[0]=-(minimalnet[i][j].iends[1]);
      minimalnet[j][i].iends[1]=-(minimalnet[i][j].iends[0]);
      minimalnet[j][i].noval=minimalnet[i][j].noval;
    };
  };
};
fstream intest("tstset.out", ios::app);
insol<<m<<endl; //SCRIVE QUANTE VARIABILI SUL TRAINING
float trai[nex][41]; // ARRAY CHE CONTIENE IL TRAINING
int e=0; //contatore esempi
int jcl=clock_seed();
while(e<nex){
 int S[m];
 int nvarval;
 S[0]=0;
 for(i=0; i<m-1; i++){
   double f;
   f=network[i][i+1].re-(network[i][i+1].le);
   do{
     int t;
     jcl=rnd7(jcl);
     t=RANDF2(f,jcl);
     t=t+network[i][i+1].le;
     nvarval=network[i][i+1].x[t]+S[i];
   }
   while(not(Pr_all_sat(S, i+1, nvarval, minimalnet)));
   S[i+1]=nvarval;
 };
 if(notapp(S,trai,e-1,m)){
   insol<<0;
   trai[e][0]=0;
   for(int mu=1; mu<m; mu++){
     trai[e][mu]=S[mu];
```

```
      insol<<" "<<S[mu];
    };
 float rating=userdrater(S,m);
 trai[e][m]=rating;
 insol<<" "<<rating<<endl;
 e++;
 };
 };
 int tstnex; //QUI COMINCIA A GENERARE IL TEST SET
 cout<<"how many examples
would you like to generate for the test set"<<endl;
 cin>>tstnex;
 float test[tstnex][41];
 e=0;
 while(e<tstnex){
   int S[m];
   int nvarval;
   S[0]=0;
   for(i=0; i<m-1; i++){
     double f;
     f=network[i][i+1].re-(network[i][i+1].le);
     int count=0;
     do{
       int t;
       jcl=rnd7(jcl);
       t=RANDF2(f,jcl);
       t=t+network[i][i+1].le;
       nvarval=network[i][i+1].x[t]+S[i];
       count++;
}
     while(not(Pr_all_sat(S, i+1, nvarval, minimalnet)));
     S[i+1]=nvarval
   };
   if(notapp(S,test,e-1,m) && notapp(S,trai,nex,m)){
     float rating=userdrater(S,m);
     test[e][0]=0;
     test[e][m]=rating;
     intest<<0;
     for(i=1; i<m; i++){
       test[e][i]=S[i];
       intest<<" "<<S[i];
```

```
   };
     intest<<" "<<rating<<endl;
     e++;
 };
 };
};
```

A.3 Generatore Random

```
/* STPP_RANDOM_GENERATOR */

#include<iostream.h>
#include <sys/types.h>
#include <time.h>
#include <stdlib.h>
#include <fstream.h>
#include <math.h>
#include <assert.h>
#include "SoftC.h"
#include "PQueue.h"
#include "Stp.h"

#define a15 452807053
#define mask 2147483647
#define MAXINT 2147483647
#define RANDF(range,seed) range*(float) (seed-MAXINT/2)/MAXINT
#define RANDF2(range,seed) (int) rint(range*(float) seed/MAXINT)

//generatore di numeri random
int clock_seed(){
  int seed;
  unsigned int st;
  time_t times;
  times= time(0);
  st=(unsigned int) times / 2;
  seed=(int) st;
  return( ((seed%2) == 0) ? seed+1 : seed );
};
```

```
int rnd7 (int IX)
{
IX=IX * a15;
return( (IX<0)? IX & mask : IX);
};

float parabola (float a, float b, float c, float x){
  float y;
  y=a*(x*x)+b*x+c;
  return y;
};

//calcola y dati a,b,c, e x
SoftC::SoftC(){
  a=0;
  b=0;
  c=1;
  le=0;
  re=120;
  int i;
  name[0]=-1;
  name[1]=-1;
   int j=-60;
   for(i=0; i<=120; i++){
     x[i]=j;
     fx[i]=1;
     j++;
   };
};

//stampa un vincolo temporale con preferenze
void print_SoftC (SoftC C){
 ofstream outFile("trunc.out", ios::app);
  int j, h;
  outFile<<C.name[0]<<C.name[1]<<endl;
  outFile<<"a:"<<C.a<<" b:"<<C.b<<" c:"<<C.c<<endl;
  h=C.le;
  outFile<<h<<endl;
  outFile<<C.re<<endl;
  for(j=C.le; j<=C.re; j++){
    outFile<<"x="<<C.x[h]<<" fx="<<C.fx[h]<<endl;
```

```
    h++;
  };
};

//stampa in forma ridotta
//un vincolo temporale con preferenze
void pfi_SoftC(SoftC C){
  fstream ogen ("tobesol.out", ios::app);
  ogen<<"y"<<endl;
  ogen<<C.name[0]<<" "<<C.name[1]<<endl;
  ogen<<C.x[C.le]<<" "<<C.x[C.re]<<endl;
  ogen<<"z"<<endl;
  ogen<<C.a<<" "<<C.b<<" "<<C.c<<endl;

};

//dato il vincolo IJ calcola JI
SoftC specular_SoftC(SoftC C){
      SoftC mirror;
      mirror.name[0]=C.name[1];
      mirror.name[1]=C.name[0];
      mirror.re=C.re-C.le;
       int h=(C.x[C.re]);
       int q=C.re;
       for(int j=mirror.le; j<=mirror.re; j++){
           mirror.x[j]=-h;
           mirror.fx[j]=C.fx[q];
           q--;
           h--;
           };
      mirror.a=C.a;
      mirror.b=-(C.b);
      mirror.c=C.c;
return mirror;
 };

//N numero di variabili
//D il dominio della variabile iniziale sara' [0,D]
//R range di espansione per gli intervalli
//V densita'
//pca, pcb, pcc percentuali di perturbazione
```

```
//dei parametri delle parabole
//generatore random
void Rgen(int N, double D,
double R,int V, int pca, int pcb, int pcc){
  fstream ogen ("tobesol.out", ios::app); //output file
  double m=N;
  SoftC * * network;
  network= new SoftC* [100];
   for(int i=0; i<100; i++){
     network[i]= new SoftC[100];
   };
  int u=(int) floor(N/2*(N-1)); //numero totale di vincoli
  int vinc=(u*V)/100; // numero di vincoli da definire
  cout<<"vinc"<<vinc<<endl;
  int i,j,k,tem,l,r,w;
  int s[N];
  s[0]=0;
  int cl;
  cl=clock_seed(); //costruzione della soluzione iniziale
  for(i=1; i<N; i++){
    cl=rnd7(cl);
    j=RANDF2(D,cl);
    s[i]=j;
  };
  for (int vu=0; vu<vinc; vu++){
    do{
      cl=rnd7(cl);
      i=RANDF2(m-1,cl);
       cl=rnd7(cl);
       j=RANDF2(m-1,cl);
     }
    while(i==j || network[i][j].name[0]!=-1);
    network[i][j].name[0]=i;
    network[i][j].name[1]=j;
    r=-s[i]+s[j];
    int pl;
    if (i==0){ pl=0;}
    else{
      cl=rnd7(cl);
      pl=RANDF2(R,cl); //espansione sinistra dell'intervallo
    };
```

```
    int pr;
    if(j==0){ pr=0;}
    else{
      cl=rnd7(cl);  //espansione destra dell'intervallo
      pr=RANDF2(R,cl);
    };
    cout<<"r "<<r<<"pl "<<pl<<"pr "<<pr<<endl;
    l=r-pl;
    r=r+pr;
      cout<<l<<" "<<r<<endl;
    network[i][j].le=0;
    network[i][j].re=r-l;
      int h=l;
    for(k=network[i][j].le; k<=network[i][j].re; k++){
    network[i][j].x[k]=h;  //inizializzazione dell'intervallo
      h++;
    };
    float xmax,ymax,yli,yri,lmax,rmax;
    float a,b,c;
    int flip;
    cl=rnd7(cl);
    flip=RANDF2(14,cl);              //flips the coin
     cout<<"flip"<<flip<<endl;
    if (flip<=4){                       //PARABOLA
      cout<<i<<j<<"parabola"<<endl;;
      float mi=l+(float) (r-l)/2;
      float q=(mi*mi-mi*l-mi*r+r*l);
      a=1/q;           //a
      b=- a*(l+r);   //b
      c=a*l*r;
      do{
float pa,pb,pc;
float ra, rb, rc;
ra=(a*pca)/100;
rb=(b*pcb)/100;
rc=(c*pcc)/100;
cl=rnd7(cl);
pa=RANDF(fabs(ra),cl);
cl=rnd7(cl);
pb=RANDF(fabs(rb),cl);
cl=rnd7(cl);
```

```
pc=RANDF(fabs(rc),cl);
a=a+pa;
b=b+pb;
c=c+pc;
if(a>-0.00000001) a=-0.00000001;
xmax=-b/(2*a);
ymax=parabola(a,b,c,xmax);
yli=parabola(a,b,c,l);
yri=parabola(a,b,c,r);
      }
      while(((xmax<=r && xmax>=l) && (ymax>1 || ymax<=0)) ||
((xmax>r || xmax<l) &&( (yri<=0 && yli<=0) || yli>1 ||yri>1)));
    };
    if(flip>=10){
      cout<<i<<j<<"retta"<<endl;       //RETTA
      int ptA, ptB;
      float pA, pB;
      double irange=r-l;
      cl=rnd7(cl);
      ptA= RANDF2(irange,cl);
      ptA=l+ptA;
      cl=rnd7(cl);
      pA=RANDF2(100,cl);
      pA=pA/100;
      do{
      cl=rnd7(cl);
      ptB=RANDF2(irange,cl);
      ptB=l+ptB;
      }
      while(ptA==ptB);
      cl=rnd7(cl);
      pB=RANDF2(100,cl);
      pB=pB/100;
      a=0;
      c=((ptA*pB)-(ptB*pA))/(ptA-ptB);
      b=(pB-pA)/(ptB-ptA);
    };
    if(flip>4 && flip<=9){
      cout<<"i"<<"j"<<"costante"<<endl;    //COSTANTE//
      a=0;
      b=0;
```

```
      c=(float) (90+flip)/100;
    };
    network[i][j].a=a;
    network[i][j].b=b; //inizializzazione dei campi a, b, c
    network[i][j].c=c;
    network[j][i]=specular_SoftC(network[i][j]);
  };
  for(int ju=1; ju<N; ju++){
    if(network[0][ju].name[0]==-1){
    network[0][ju].name[0]=0;
    network[0][ju].name[1]=ju;
    network[0][ju].le=60;
    network[0][ju].re=120;
    network[ju][0]=specular_SoftC(network[0][ju]);
    };
  };
  for(i=0; i<N; i++){
    for(j=1; j<N; j++){
      if(i<j && network[i][j].name[0]==-1){
network[i][j].name[0]=i;
        network[i][j].name[1]=j;
        network[j][i].name[0]=j;
        network[j][i].name[1]=i;
      };
    };
  };
  ogen<<"N "<<N<<endl;
  for (int q=0; q<N; q++){
    for(int p=1; p<N; p++){
        if (q<p)
  {
    print_SoftC(network[q][p]);
    pfi_SoftC(network[q][p]);
  };
    };
  };
  for(i=0; i<100; i++){
    delete network[i];
  };
};
```

```
void main(){
  ofstream outFile("tobesol.out", ios::app);
  fstream ingen("tobegen.in", ios::in);
  enum poss{y='y', n='n'};
  char ch;
  ingen>>ch;
  while(ch!=n){
    int N, pca,pcb,pcc,V;
    double D, R;
    ingen>>N;              //lettura parametri
    outFile<<N<<endl;
    ingen>>D;
    ingen>>R;
    ingen>>V;
    ingen>>pca;
    ingen>>pcb;
    ingen>>pcc;
    Rgen(N,D,R,V,pca,pcb,pcc);  //chiamata al generatore
    outFile<<"n"<<endl;
    ingen>>ch;
  };
};
```

A.4 Modulo di apprendimento

```
/* STPP_LEARNING_MODULE */

#include<iostream.h>
#include<stdlib.h>
#include <fstream.h>
#include <math.h>
#include <assert.h>
#include "SoftC.h"
#include "PQueue.h"
#include "Stp.h"

#define a15 452807053
#define mask 2147483647
```

```
#define MAXINT 2147483647
#define RANDF(range,seed)range*(float)(seed-MAXINT/2)/MAXINT

//generatore random di numeri
int clock_seed(){
  int seed;
  unsigned int st;
  time_t times;
  times= time(0);
  st=(unsigned int) times / 2;
  seed=(int) st;
  return( ((seed%2) == 0) ? seed+1 : seed );
};

int rnd7 (int IX)
{
  IX=IX * a15;
  return( (IX<0)? IX & mask : IX);
};

//costruttore
SoftC::SoftC(){
  a=0;
  b=0;
  c=1;
  le=0;
  re=0;
  int i;
  name[0]=-1;
  name[1]=-1;
  for(i=0; i<40; i++){
    x[i]=0;
    fx[i]=0;
  };
};

//stampa un vincolo
//temporale con preferenze
void print_SoftC (SoftC C){
  ofstream outFile("lea.out", ios::app);
  int j, h;
```

```
  outFile<<C.name[0]<<C.name[1]<<endl;
  h=C.le;
  outFile<<h<<endl;
  outFile<<C.re<<endl;
  outFile<<C.a<<" "<<C.b<<" "<<C.c<<endl;
  for(j=C.le; j<=C.re; j++){
    outFile<<"x="<<C.x[h]<<" fx="<<C.fx[h]<<endl;
    h++;
  };
};

//stampa in modo ridotto un vincolo temporale con preferenze
void pfi_SoftC(SoftC C){
  fstream ogen ("leatobesol.out", ios::app);
  ogen<<"y"<<endl;
  ogen<<C.name[0]<<" "<<C.name[1]<<endl;
  ogen<<C.x[C.le]<<" "<<C.x[C.re]<<endl;
  ogen<<C.a<<" "<<C.b<<" "<<C.c<<endl;
  ogen<<"d"<<endl;
  for(int i=C.le; i<=C.re; i++){
    ogen<<C.fx[i]<<" ";
  };
  ogen<<endl;
};

float modulo(float g){
  if(g>=0) return g;
  else return -g;
};

//stampa solo i parametri a,b, c dei vincoli
void short_print_SoftC (SoftC C){
  ofstream outFile("lea.out", ios::app);
  int j, h;
  outFile<<C.name[0]<<C.name[1]<<endl;
  h=C.le;
  outFile<<C.x[h];
  outFile<<C.x[C.re]<<endl;
  outFile<<C.a<<" "<<C.b<<" "<<C.c<<endl;
};
```

```
//dato il vincolo IJ calcola JI
SoftC specular_SoftC(SoftC C){
      SoftC mirror;
      mirror.name[0]=C.name[1];
      mirror.name[1]=C.name[0];
      mirror.re=C.re-C.le;
      int h=(C.x[C.re]);
      int q=C.re;
      for(int j=mirror.le; j<=mirror.re; j++){
           mirror.x[j]=-h;
           mirror.fx[j]=C.fx[q];
           q--;
           h--;
           };
 return mirror;
 };

//dato il vincolo IJ calcola JI solo tenendo conto di a,b,c
SoftC short_specular_SoftC(SoftC C){
      SoftC mirror;
      mirror.name[0]=C.name[1];
      mirror.name[1]=C.name[0];
      mirror.re=C.re-C.le;
mirror.a=C.a;
mirror.b=-(C.b);
 mirror.c=C.c;
 return mirror;
 };

//calcola la preferenza di una soluzione (con max-min)
float drater(int *S, int m, SoftC network [40][40]){
  float rating=1;
  int xi, xj, index;
  for (int i=0; i<m; i++){
     for (int j=1; j<m; j++){
       if(i<j){
         xi=S[i];
         xj=S[j];
index=((xj-xi)-network[i][j].x[network[i][j].le]
    +network[i][j].le);
   if(rating>network[i][j].fx[index];
```

```
   rating=network[i][j].fx[index];
 };
       };
     };
  };
   return rating;
};

//calcola y con la formula di fuzzy-parabola
float fuzzy_parabola (float a, float b, float c, int x){
  float y;
  y=a*(x*x)+b*x+c;
  if(y>1) y=1;
  if (y<0) y=0;
  return y;
};

//calcola y con la formula di una parabola
float parabola (float a, float b, float c, int x){
  float y;
  y=a*(x*x)+b*x+c;
  return y;
};

//calcola la preferenza di una soluzione
//solo con i parametri a, b, e c delle parabole
float pdrater(int *S, int m, SoftC network [40][40]){
  float rating=1;
  int xi, xj, temp;
   float index;
   for (int i=0; i<m; i++){
     for (int j=1; j<m; j++){
       if(i<j){
         xi=S[i];
         xj=S[j];
 float a,b,c;
         a=network[i][j].a;
         b=network[i][j].b;
         c=network[i][j].c;
         temp=xj-xi;
         index=parabola(a,b,c,temp);
```

```
         if(rating>index){
   rating=index;
 };
       };
     };
   };
   return rating;
};

//calcola la preferenza di una soluzione
//solo con i parametri a, b, e c delle fuzzy-parabole
float fuzzy_pdrater(int *S, int m, SoftC network [40][40]){
  float rating=1;
  int xi, xj, temp;
   float index;
   for (int i=0; i<m; i++){
     for (int j=1; j<m; j++){
       if(i<j){
         xi=S[i];
         xj=S[j];
 float a,b,c;
         a=network[i][j].a;
         b=network[i][j].b;
         c=network[i][j].c;
         temp=xj;
         index=fuzzy_parabola(a,b,c,temp);
         if(rating>index){
   rating=index;
 };
       };
     };
   };
   return rating;
};

// calcola
// A=  sum          ( e^-(beta*(aj*xj*xj+bj*xj+cj)))
//    j=1........nu
// nu=numero di vincoli
float sum_all_exp(int *S, int m, SoftC tnetwork [40][40]){
  int beta=8;
```

```
  int x;
  float a,b,c;
  float temp1=0;
  float temp2=0;
for (int i=0; i<m; i++){
     for (int j=1; j<m; j++){
       if(i<j){
 x=S[j]-S[i];
         a=tnetwork[i][j].a;
         b=tnetwork[i][j].b;
         c=tnetwork[i][j].c;
         temp1=-(beta)*(parabola(a,b,c,x));
         temp2=temp2+exp(temp1);
       };
     };
};
 return temp2;
};

//funzione che calcola la preferenza di una soluzione
//utilizzando l'approssimazione continua di min
//float crater (int m, float temp2){
//  int beta=4;
//  int q=(m*m-m)/2;
//  float vrating=-(1/beta)*ln(1/q*temp2);
//  return vrating;
//};
//calcola
// B=(d(s)-h(s))*((e^(-beta*(ai*xi*xi+bi*xi+ci)))/A)
float commonAdj(float rating, float vrating, float a,float b,
float c, int x, float temp2){
  int beta=8;
  float part1, part2;
  part1=(exp((-beta)*parabola(a,b,c,x))/temp2);
  part2=rating-vrating;
  float comm=part1*part2;
  return comm;
}

//(dE/dai)=-xi^2*B
float aAdj(float eta, int x, float comm){
```

```
  float newa=(-eta)*(-x*x)*comm;
  return newa;
};

//(dE/dbi)=-xi*B
float bAdj(float eta, int x, float comm){
  float newb=(-eta)*(-x)*comm;
  return newb;
};

//(dE/dci)=-B
float cAdj(float eta, int x, float comm){
  float newc=(-eta)*(-comm);
  return newc;
  };

void main(){
  int nex,tnex; //nex=numero esempi per il training set
  cout<<"n of ex in training set:"<<endl; //tnex per il test
  cin>>nex;
  cout<<"n of ex in test set:"<<endl;
  cin>>tnex;
  float Equem, tempEquem, fuzzy_Equem, fuzzy_tempEquem;
  int iterations=0;
  int logcount=0;
  int stopcount=0;
  float pthres=0;
  float eta=0;
  float temp2, t;
  float error=0;       //errore massimo
  float fuzzy_error=0; //errore massimo con parabole troncate
  float trai_averror=0;
  float fuzzy_trai_averror=0;
  float old_fuzzy_trai_averror=0;
  float better=0;
  int i, j, li, ri, m, k;
  SoftC traine[40][40];  //rete che verra' allenata
  ifstream inet ("tnet.out", ios::in);    //STP iniziale
  ifstream insol ("trset.out", ios::in);  //training set
  ifstream intest ("tstset.out", ios::in); //test set
  fstream outFile ("lea.out", ios::app); //STPP finale
```

```
  fstream log ("lealog.out", ios::app); //,onitoraggio
  insol>>m;
  char ch;
  enum poss {y='y', n='n'}; //caricamento della rete
  inet>>ch;
  while (ch!=n){
    inet>>i;
    inet>>j;
    traine[i][j].name[0]=i;
    traine[i][j].name[1]=j;
    traine[i][j].le=0;
    inet>>li;
    inet>>ri;
    int r = li;
    traine[i][j].re=ri-li;
    for(int p=0; p<=traine[i][j].re; p++){
      traine[i][j].x[p]=li;
      li++;
    };
    float xmax,ymax,yli,yri,lmax,rmax;
    float a,b,c;
    int l=li;
    int mi=l+(r-l)/2;
    float q=(mi*mi-mi*l-mi*r+r*l);
    a=1/q;           //a
    b=- a*(l+r);   //b
    c=a*l*r;       //c
    traine[i][j].a=0;
    traine[i][j].b=0;
    traine[i][j].c=1;
    traine[j][i]=short_specular_SoftC(traine[i][j]);
    inet>>ch;
  };
  for(i=0; i<m; i++){
    for(j=1; j<m; j++){
      if(i<j){
print_SoftC(traine[i][j]);
      };
    };
  };
  float tset[nex][m+1];
```

```
  for(i=0; i<nex; i++){
    for(j=0; j<=m; j++){
      insol>>t;
      tset[i][j]=t;
    };
  };
  do{
    Equem=0;
    for (int ls=0; ls<nex; ls++){ //PER OGNI ESEMPIO
      int s[m];
      for(i=0; i<m; i++){
s[i]=(int) tset[ls][i];
      };
      float crating;   //correct rating  target
      crating=tset[ls][m];
      float trating,temprating;
      trating=pdrater(s,m,traine);
      temp2=sum_all_exp(s,m,traine);
      for(int h=0; h<m; h++){   //PER OGNI VINCOLO
for(int k=1; k<m; k++){
  if(h<k){
    int x;
    x=s[k]-s[h];
    float comm,tempa;
    comm=commonAdj(crating,trating, traine[h][k].a,
            traine[h][k].b, traine[h][k].c,x, temp2);
    tempa=traine[h][k].a+aAdj(eta,x,comm);
    if(tempa<0){      //semiconvex-forced learning
      traine[h][k].a=tempa;
    }
    else{
      traine[h][k].a=0;
    };
    traine[h][k].b=traine[h][k].b+bAdj(eta,x,comm);
    traine[h][k].c=traine[h][k].c+cAdj(eta,x,comm);
    traine[k][h].a= traine[h][k].a ;
    traine[k][h].b= -(traine[h][k].b);
    traine[k][h].c= traine[h][k];
  };
};
      };
```

```
    }; //qui ha finito di guardare a
       //tutti gli esempi del training set
    Equem=0; //errore sum of sq non trunc
    error=0; //errore max sul trai abs
    fuzzy_Equem=0;//errore sum of sq trunc
    fuzzy_error=0; //errore max sul trai trunc
    trai_averror=0; //errore medio abs sul trai n trunc
    old_fuzzy_trai_averror=fuzzy_trai_averror;
    fuzzy_trai_averror=0; //errore medio abs sul trai trunc
    for (int ls=0; ls<nex; ls++){
      float lerror, fuzzy_lerror;
      int s[m];
      for(i=0; i<m; i++){
s[i]=(int) tset[ls][i];
      };
      float crating;   //correct rating
      crating=tset[ls][m];
      float trating,temprating, fuzzy_trating;
      trating=pdrater(s,m,traine);
      fuzzy_trating=fuzzy_pdrater(s,m,traine);
      tempEquem=0.5*(crating-trating)*(crating-trating);
      Equem=Equem+tempEquem;
      fuzzy_tempEquem=0.5*(crating-fuzzy_trating)
      *(crating-fuzzy_trating);
      fuzzy_Equem=fuzzy_Equem+fuzzy_tempEquem;
      lerror=modulo(crating-trating);
      trai_averror=trai_averror+lerror;
      if(error<lerror) {error=lerror;};
      fuzzy_lerror=modulo(crating-fuzzy_trating);
      if(fuzzy_error<fuzzy_lerror) {fuzzy_error=fuzzy_lerror;};
      fuzzy_trai_averror=fuzzy_trai_averror+fuzzy_lerror;
    };
    trai_averror=trai_averror/nex;
    fuzzy_trai_averror=fuzzy_trai_averror/nex;
    if(iterations>+2){
      better=old_fuzzy_trai_averror-fuzzy_trai_averror;
      cout<<"better "<<better<<endl;
      if(better>=pthres) {
pthres=(better*7)/10;
stopcount=0;
      }
```

```
      else{
if(pthres>0){
  pthres=(better*7)/10;
  stopcount++;}
else{
  pthres=0;
  stopcount++;
};
      };
    };
    cout<<"pthres "<<pthres<<endl; //monitoraggio
    cout<<"to stop "<<stopcount<<endl;
    if(logcount==100){
      log<<"max diff non troncate "<<error<<endl;
      log<<"Equem non troncate "<<Equem<<endl;
      log<<"max diff troncate "<<fuzzy_error<<endl;
      log<<"Equem troncate "<<fuzzy_Equem<<endl;
      log<<"Err med abs trai non trunc "
         <<trai_averror<<endl;
      log<<"Err med abs trai trunc "
         <<fuzzy_trai_averror<<endl;
      log<<endl;
      log<<endl;
      logcount=0;
    };
    iterations++;
    logcount++;
    cout<<iterations<<endl;
    eta = 0.0000001;
    cout<<"eta "<<eta<<endl;
    cout<<iterations<<endl;
  }
  while(stopcount<100);
  cout<<"succeded ad iteration no."<<iterations<<endl;
  //apprendimento terminato
  for(i=0; i<m; i++){
    for (j=1; j<m; j++){
      if (i<j){
for(k=0; k<=traine[i][j].re; k++){
  float temp;
  temp=fuzzy_parabola(traine[i][j].a, traine[i][j].b,
```

```
traine[i][j].c,traine[i][j].x[k]);
  temp=temp*100;
  temp=floor(temp);
  temp=temp/100;
  traine[i][j].fx[k]=tem
};
pfi_SoftC(traine[i][j]);
traine[j][i]=specular_SoftC(traine[i][j]);
      };
    };
  };
  float tsterror=0;
  float averror=0;
  for (int ltest=0; ltest<tnex; ltest++){ //verifica sul test set
    float lterror;
    float okrating, mayberating;
    int test[m];
    int v;
    for(i=0; i<m; i++){
      intest>>v;
      test[i]=v;
      outFile<<v<<" ";
    };
    intest>>okrating;
    outFile<<okrating<<" ";
    mayberating=drater(test, m, traine);
    outFile<<mayberating<<endl;
    lterror=modulo(okrating-mayberating);
    averror=averror+lterror;
    if(tsterror<=lterror) tsterror=lterror;
  };
  averror=averror/tnex;
  outFile<<"maximum error on the test set is"<<tsterror<<endl;
  outFile<<"average error on the test set is"<<averror<<endl;
};
```