Tecniche di programmazione lineare per l'identificazione di reti di Petri

Pierandrea Secchi

Tesi di Laurea Specialistica in Ingegneria Elettronica

Corso di LS in Ingegneria Elettronica, Università degli Studi di Cagliari,

Dip. di Ing. Elettrica ed Elettronica, P.za D'Armi, 09123 Cagliari, Italia

Relatori: Alessandro Giua, Maria Paola Cabasino

email: piero.1979@libero.it

23 luglio 2008

Riassunto della tesi

Oggetto di questa tesi è lo sviluppo di uno strumento software in Matlab per l'identificazione di una rete posto/transizione, basato sulla programmazione lineare. La procedura di identificazione riceve in ingresso un linguaggio $\mathcal{L}$ finito e chiuso per prefisso e un intero k maggiore o uguale alla lunghezza della stringa più lunga in esso contenuta, e produce in uscita una rete il cui linguaggio delle parole generate di lunghezza non superiore a k coincide con $\mathcal{L}$.

La procedura viene eseguita nel seguente modo.

1. A partire da $\mathcal{L}$ costruisce gli insiemi delle condizioni di abilitazione $\mathcal{E}$ e di disabilitazione $\mathcal{D}$.

2. Da questi estrae un insieme di vincoli lineari, le cui incognite sono gli elementi della marcatura iniziale M_0 e delle matrici $Post$ e Pre. Il numero di posti della rete è inizialmente pari alla cardinalità di $\mathcal{D}$ o, eventualmente, minore se sono presenti particolari condizioni sugli elementi di $\mathcal{D}$, ma può essere ridotto in base ad informazioni aggiuntive sulla struttura della rete (tecnica di pre-riduzione).

3. L'insieme di vincoli così costruito, a cui eventualmente si possono aggiungere nuovi vincoli se sono presenti altre informazioni sulla rete, viene risolto mediante l'ausilio della libreria software GLPK, determinando la rete con marcatura iniziale M_0 e struttura $Post$ e Pre che genera il linguaggio dato $\mathcal{L}$.

4. Infine i posti ridondanti possono essere rimossi (tecnica di post-riduzione) senza modificare il linguaggio generato.

Le funzioni Matlab che costituiscono lo strumento software eseguono automaticamente i vari passaggi sopra descritti. La funzionalità dei programmi e la correttezza della procedura sono state testate su varie reti. I dati più importanti ricavati dai test riguardano i limiti di affidabilità dei risolutori di GLPK, i linguaggi generati dalla rete, la complessità del problema di programmazione lineare, la variazione del numero di posti durante i vari passi, i tempi di computazione e il limite di applicabilità della procedura dal punto di vista della memoria utilizzata da Matlab.

Indice

Capitolo 1

Introduzione

1.1 Obiettivi della tesi

L'identificazione è un problema classico nella teoria dei sistemi: dato un comportamento osservato, consiste nel determinare un sistema il cui comportamento approssimi quello osservato [14]. Il concetto di identificazione non è generale, ma si adatta ai diversi ambiti in cui si definisce. Per esempio, in automatica l'identificazione dei sistemi è una scienza che si occupa di ricavare dei modelli di sistemi a partire da dati sperimentali. Numerosi sistemi infatti sono difficili da modellare tramite le leggi della fisica o troppo complicati: per questo motivo l'identificazione cerca di trovare un modello che si adegui alle misure effettuate. In genere, data una coppia di segnali ingresso-uscita osservati, il problema di identificazione consiste nel determinare un sistema tale che i segnali in ingresso e in uscita approssimino quelli osservati.

Nel contesto delle reti di Petri è usuale considerare come comportamento osservato il linguaggio della rete, cioè l'insieme delle sequenze di transizioni che possono essere abilitate a partire dalla marcatura iniziale.

In [3] e [13] è stato presentato il seguente problema di identificazione. Assumiamo che sia assegnato un linguaggio $\mathcal{L} \subset T^*$, dove T è un dato insieme di n transizioni. Supponiamo che questo linguaggio sia finito e chiuso per prefisso e supponiamo che k sia un intero maggiore o uguale alla lunghezza della stringa più lunga contenuta in esso. Dato un numero di posti m, il problema di identificazione considerato consiste nel determinare la struttura di una rete di Petri N, cioè le matrici $Post, Pre \in \mathbb{N}^{m \times n}$, e la sua marcatura iniziale $M_0 \in \mathbb{N}^m$ tale che l'insie-

me di tutte le sequenze di transizioni abilitate di lunghezza minore o uguale a k sia $L_k(N, M_0) = \mathcal{L}$. Da notare che l'insieme $\mathcal{L}$ elenca esplicitamente *gli esempi positivi*, cioè le stringhe che appartengono al linguaggio, ma implicitamente definisce anche i *controesempi*, cioè tutte quelle stringhe di lunghezza minore o uguale a k che non appartengono al linguaggio. Perciò dal linguaggio osservato si può costruire un insieme di vincoli di *abilitazione* $\mathcal{E}$, cioè un insieme di coppie (σ, t) tali che la transizione t è abilitata dopo che la sequenza σ è scattata, e un insieme di vincoli di *disabilitazione* $\mathcal{D}$, cioè un insieme di coppie (σ, t) tali che la transizione t non è abilitata dopo che la sequenza σ è scattata.

Nel precedente lavoro [3] è stato dimostrato che questo problema (ed una correlata serie di problemi di identificazione più generali) può essere risolto usando la programmazione intera. Lo svantaggio principale di questo approccio è la sua complessità computazionale, nel senso che il numero di incognite o variabili cresce esponenzialmente con la lunghezza della stringa più lunga in $\mathcal{L}$, rendendo molti problemi facilmente intrattabili.

In [13] è stato mostrato come trattare lo stesso problema utilizzando tecniche di programmazione lineare, che riducono significativamente la complessità della risoluzione del problema di identificazione. L'idea principale è quella di cercare soluzioni particolari del problema di identificazione che sono chiamate $\mathcal{D}$-canoniche, cioè reti con un numero di posti uguale alla cardinalità dell'insieme $\mathcal{D}$, indicata con $|\mathcal{D}|$. È stato dimostrato che: (a) se un dato problema di identificazione ha soluzione, allora ha anche una soluzione $\mathcal{D}$-canonica; (b) questa particolare soluzione può essere calcolata risolvendo un problema di programmazione lineare, cioè considerando una funzione obiettivo che consente di trovare una rete ottima secondo un dato indice di prestazione.

Quindi, la procedura proposta identifica una rete il cui numero di posti è pari alla cardinalità dell'insieme dei vincoli di disabilitazione che può essere grande, sebbene possa esistere una rete equivalente, cioè che genera lo stesso linguaggio, con un numero di posti molto più piccolo. Successivamente viene fornito un criterio per controllare se la soluzione calcolata ha un numero minimo di posti e, se non è questo il caso, allora si discutono due approcci per ridurre questo numero.

L'oggetto di questa tesi è lo sviluppo di un pacchetto Matlab che implementi la procedura descritta in [13] e la validazione sperimentale di tale approccio.

La procedura proposta è stata implementata mediante la versione 6.5 di Matlab. I problemi di programmazione sono stati risolti per mezzo del software GLPK versione 4.9, utilizzato in ambiente Matlab.

1.2 Rassegna della letteratura

L'idea di studiare la struttura di un automa a partire da esempi positivi e controesempi è stata esplorata fin dai primi anni 80 nell'ambito dei linguaggi formali [15, 16].

Uno dei primi approcci originali all'identificazione di reti di Petri sane è stato discusso da Hiraishi [7], che ha presentato un algoritmo per la costruzione di un modello di rete di Petri non etichettata a partire dalla conoscenza di un insieme finito delle sue sequenze di scatto.

Un differente approccio è basato sulla *teoria delle regioni* il cui obiettivo è quello di decidere se un dato grafo è isomorfo al grafo di raggiungibilità di una rete non etichettata e poi costruirlo. Un ottimo esame di quest'approccio, che presenta anche alcuni efficienti algoritmi per la sintesi di una rete basata sull'algebra lineare, può essere trovato nel lavoro di Badouel e Darondeau [1].

Meda e Mellado [9, 10] hanno presentato un approccio per l'identificazione di reti di Petri decodificate non etichettate. Il loro approccio consiste nell'osservare la marcatura di un sottoinsieme di posti e, date alcune informazioni aggiuntive sulla dipendenza tra le transizioni, permettere di ricostruire la parte della struttura di rete relativa ai posti non osservati.

Bourdeaud'huy e Yim [2] hanno presentato un approccio per ricostruire la matrice di incidenza e la marcatura iniziale di una rete non etichettata date alcune informazioni strutturali sulla rete, come l'esistenza di P-invarianti o T-invarianti. Quest'approccio può anche trattare esempi positivi di sequenze di scatto ma non controesempi.

Dotoli *et al.* in [5] hanno considerato un approccio basato sull'ottimizzazione che fonde alcune caratteristiche di [3], [9] e [10]. La loro procedura assume che sia data una produzione della rete, cioè non richiede solo la conoscenza delle sequenze di eventi ma anche delle marcature raggiunte durante questa evoluzione. Sono date condizioni necessarie e sufficienti per la corretta identificazione della rete.

Infine, Sreenivas in [17] ha trattato la minimizzazione di modelli di reti di Petri. Dato un generatore di rete di Petri non etichettata e una funzione peso che associa al generatore un intero non negativo, l'obiettivo è quello di trovare una rete di Petri che genera lo stesso linguaggio della rete originale minimizzando il peso dato.

1.3 Struttura della tesi

Nel seguito verrà illustrata la struttura della tesi.

Nel capitolo 2 vengono fatti dei richiami alle reti P/T introducendo il formalismo utilizzato nella tesi. Viene definita la struttura algebrica e grafica delle reti P/T, in particolare le matrici $Post$ e Pre, e vengono richiamate le leggi che ne governano l'evoluzione dinamica. Poi viene definito il concetto di *rete marcata* o *sistema di rete*, cioè una rete individuata dalle matrici $Post$ e Pre con una marcatura iniziale M_0. Viene introdotto il concetto di linguaggio, sia globale, cioè l'insieme $L(N, M_0)$ di tutte le sequenze di scatto abilitate dalla marcatura iniziale, sia parziale, cioè l'insieme $L_k(N, M_0)$ delle sequenze di scatto di lunghezza minore o uguale a k. Vengono poi definite alcune proprietà strutturali di una rete P/T, cioè indipendenti dalla marcatura iniziale, e infine vengono richiamate alcune classi particolari di reti P/T.

Nel capitolo 3 vengono fatti dei richiami alla programmazione lineare. Si comincia con l'introduzione della formulazione di tale modello di ottimizzazione vincolata, in base al quale un programma lineare è un problema di ottimizzazione con una funzione obiettivo lineare e dei vincoli lineari rispetto a variabili non negative. Viene mostrata la procedura di soluzione grafica per i programmi lineari che coinvolgono due o tre variabili, basata sul concetto di poliedro. Poi viene introdotto il metodo del simplesso, il più utilizzato per la risoluzione di questo tipo di problemi. Successivamente viene presentata la teoria della programmazione duale in base alla quale un problema di programmazione primario viene risolto risolvendo un associato problema duale. Infine viene descritto il metodo del punto interno e vengono dati dei cenni alla programmazione intera.

Nel capitolo 4 viene illustrata la procedura di identificazione di una rete P/T proposta da Cabasino *et al.* in [13]. Il capitolo si apre con una serie di nozioni preliminari: si definisce una classe di vincoli lineari, che chiamiamo *Constraint Set* (CS), e si caratterizzano i CS ideali. Poi viene presentata la procedura di identificazione di base. Si parte dalla definizione del problema in questione in base alla quale, dato un linguaggio $\mathcal{L}$ finito e chiuso per prefisso costituito da parole di lunghezza minore o uguale a k e scelto un determinato numero di posti, si vuole identificare una rete $M_0, Post$ e Pre che genera il linguaggio $\mathcal{L}$. Vengono definiti gli insiemi delle condizioni di abilitazione $\mathcal{E}$ e di disabilitazione $\mathcal{D}$ e si definisce il sistema di rete $\mathcal{D}$-canonico, cioè una rete in cui un differente posto è associato ad ogni elemento di $\mathcal{D}$. Viene definito il CS e viene dimostrato come una sua qualsiasi soluzione intera sia una soluzione $\mathcal{D}$-canonica del problema di identificazione. Poi viene mostrato come il numero di vincoli associati ad $\mathcal{E}$ e $\mathcal{D}$ può essere ri-

dotto consentendo di semplificare il CS. Successivamente viene dimostrato che il problema di identificazione ammette soluzione se e solo se il CS è ammissibile e viene enunciato nuovamente il problema di identificazione associando una funzione obiettivo all'insieme dei vincoli. Quindi vengono illustrati tre esempi di applicazione della procedura. Infine vengono introdotte alcune estensioni alla procedura di base che consistono nell'aggiungere nuovi vincoli al CS, sfruttando delle informazioni addizionali sulla struttura della rete (proprietà e classi di reti P/T) o sulla marcatura iniziale (GMEC), e risolvere così un CS aumentato.

Nel capitolo 5 vengono presentati i due approcci per ridurre il numero di posti, proposti in [13]: la tecnica della pre-riduzione e la tecnica della post-riduzione. I due approcci sono stati proposti per eliminare lo svantaggio più importante della procedura presentata nel capitolo precedente, cioè la richiesta di una rete con un numero di posti pari alla cardinalità di $\mathcal{D}$. Prima viene presentata la tecnica di pre-riduzione che, come suggerisce il nome, viene applicata prima di calcolare la soluzione del problema di programmazione. Viene presentato un risultato generale che consente di controllare se il numero di posti richiesto è minimo. A tal scopo si introduce il concetto di partizionamento a blocchi dell'insieme $\mathcal{D}$, si caratterizza il numero di posti in maniera tale che esso sia pari al numero di blocchi di tale partizione e si determina se una partizione è minima. Siccome il numero di queste partizioni è spesso un numero molto grande, questo procedimento non viene mai utilizzato. Nella pratica la pre-riduzione viene effettuata determinando una partizione di $\mathcal{D}$, sfruttando delle informazioni aggiuntive sulla rete. Questa partizione, come verrà mostrato, non è necessariamente quella minima. Si può ulteriormente ridurre il numero di posti e di vincoli come spiegato nel capitolo 4. Successivamente viene presentata la tecnica di post-riduzione che, come suggerisce il nome, viene applicata dopo aver calcolato la soluzione del problema di programmazione. Prima si controlla se la rete ottenuta ha posti ridondanti che possono essere rimossi senza modificare il linguaggio generato $\mathcal{L}$. A tal scopo viene definito il concetto di *hitting set*. Quindi per verificare se il numero di posti è minimo si ricorre alla nozione di *hitting set* minimo. Viene presentato un algoritmo basato sulla programmazione intera per calcolare l'*hitting set* minimo. Infine vengono riportati due esempi di post-riduzione, dei quali il secondo dimostra che non sempre questa tecnica garantisce di trovare la rete col numero minimo di posti.

Nel capitolo 6 vengono presentati i programmi realizzati con Matlab che consentono di eseguire automaticamente la procedura di identificazione illustrata nei capitoli 4 e 5. Si comincia con una presentazione generale del lavoro svolto mediante l'ausilio di uno schema a blocchi che illustra i vari programmi realizzati, gli ingressi, le uscite e i collegamenti tra i programmi. Poi viene descritto ogni singolo programma realizzato. Nell'ordine, per ciascun programma vengono ri-

portati la sintassi della relativa funzione, la lista degli argomenti di ingresso, la lista degli argomenti di uscita, la descrizione sintetica dell'algoritmo nei suoi vari passi e un esempio di applicazione, con relativa spiegazione, tratto dalla finestra dei comandi di Matlab. Sono stati realizzati 6 programmi. Il primo calcola il linguaggio massimale generato dalla rete di cui abbiamo a disposizione una struttura di partenza, cioè il linguaggio costituito dalle stringhe di massima lunghezza, non superiore a k. Il secondo costruisce gli insiemi $\mathcal{E}$ e $\mathcal{D}$ a partire dal linguaggio massimale. Il terzo costruisce il CS a partire da $\mathcal{E}$ e $\mathcal{D}$, eventualmente sfruttando le informazioni sulla rete che consentono di applicare una pre-riduzione. Il quarto identifica la rete P/T risolvendo il problema di programmazione lineare definito con l'ausilio di GLPK, eventualmente sfruttando informazioni addizionali sulla rete. Il quinto estrae la rete ridotta applicando una post-riduzione. Infine il sesto programma esegue l'intera procedura di identificazione richiamando i 5 programmi precedenti.

Nel capitolo 7 vengono mostrati degli esempi di testing dei programmi effettuati su 4 sistemi di rete al variare di determinati parametri e vengono elencati i risultati ottenuti. Si comincia con una presentazione generale in cui vengono specificati la macchina su cui sono state eseguite le simulazioni e i dati più importanti ricavati dai test, organizzati in tabelle. Questi riguardano i limiti di utilizzo di Matlab e GLPK, i linguaggi generati nei vari passi della procedura, la complessità del problema di programmazione lineare, la variazione del numero di posti e i tempi di computazione. Poi vengono illustrati i 4 esempi per ciascuno dei quali vengono riportate le tabelle contenenti i dati più importanti e vengono riportati i relativi commenti e le considerazioni. Il capitolo si chiude con la presentazione dei risultati finali.

Il capitolo 8 è quello conclusivo.

In appendice si possono trovare i listati Matlab dei programmi e alcune note su GLPK e GLPKMEX, un'interfaccia MEX di GLPK per utilizzare il software in Matlab.

I contributi che questa tesi ha portato alla ricerca sono di natura implementativa (sviluppo di software) e sperimentale:

- Un primo contributo originale della tesi consiste nello sviluppo di un toolbox Matlab per l'identificazione di una rete P/T.
- Un secondo contributo, di natura sperimentale, consiste nella dimostrazione che i risolutori della versione 4.9 di GLPK non sono totalmente affidabili per problemi di grandi dimensioni.

- Un terzo contributo, anch'esso sperimentale, consiste nella dimostrazione che la programmazione lineare riduce sensibilmente il tempo di risoluzione di un problema di identificazione rispetto alla programmazione intera.
- Un quarto contributo sperimentale consiste nella dimostrazione che la tecnica di post-riduzione permette in tutti i casi trattati di estrarre una rete con un numero di posti molto più piccolo rispetto alla rete estratta mediante la risoluzione del problema di programmazione lineare.

Capitolo 2

Richiami alle reti di Petri

2.1 Introduzione

In questo capitolo viene richiamato il formalismo delle reti di Petri utilizzato nella tesi. Facciamo riferimento a [11] per una più completa introduzione alle reti.

Le reti di Petri sono un modello di sistemi ad eventi discreti che trae origine dal lavoro di Carl Adam Petri, un ricercatore tedesco, che nel 1962 discusse la sua tesi di dottorato dal titolo "Kommunication mit Automaten", in cui presentava questo nuovo modello logico che in seguito avrebbe appunto preso il suo nome.

Un sistema ad eventi discreti (SED) è un sistema il cui comportamento dinamico è caratterizzato dall'accadimento asincrono di eventi che individuano lo svolgimento di attività di durata non necessariamente nota. Formalmente, un sistema ad eventi discreti è caratterizzato da:

- Un insieme E degli eventi accadibili.
- Uno spazio di stato costituito da un insieme discreto X.
- Un'evoluzione dello stato regolata dagli eventi: lo stato evolve nel tempo solo in dipendenza dell'accadimento di eventi asincroni, appartenenti all'insieme E.

L'equazione che descrive l'evoluzione dello stato a partire dallo stato iniziale x_0 è

$$x_{k+1} = \delta(x_k, e_k)$$

dove

- x_{k+1} è lo stato del sistema dopo l'accadimento dell'evento k-esimo.
- e_k è il k-esimo evento accaduto dall'istante iniziale considerato, che fa transire lo stato da x_k a x_{k+1}.
- $\delta : X \times E \longrightarrow X$ è la funzione di transizione di stato.

I sistemi ad eventi discreti si dividono in due grandi famiglie: i modelli logici e i modelli temporizzati, a seconda del tipo di traccia (o traiettoria) degli eventi adottata. Nei modelli logici la traccia degli eventi è costituita semplicemente da una sequenza di eventi $\{e_1, e_2, ...\}$, in ordine di occorrenza, senza alcuna informazione circa i tempi di occorrenza degli eventi. Dato uno stato iniziale x_0, la traiettoria dello stato verrà costruita nel tempo come la sequenza di stati $\{x_0, x_1, x_2, ...\}$ risultanti dall'accadimento della sequenza di eventi, ma non è possibile specificare gli istanti di tempo in cui avvengono le transizioni di stato. Nei modelli temporizzati, invece, la traccia degli eventi è costituita da una sequenza di coppie $\{(e_1, \tau_1), (e_2, \tau_2), ...\}$ dove ogni elemento e_i è accoppiato al suo tempo di accadimento τ_i, eventualmente stocastico. Dato uno stato iniziale x_0, la traiettoria dello stato sarà evidentemente ancora la sequenza di stati $\{x_0, x_1, x_2, ...\}$ risultanti dall'accadimento della sequenza di eventi. In questo caso però si sa che le transizioni di stato avvengono negli istanti di occorrenza degli eventi. I modelli logici rendono agevole lo studio delle proprietà qualitative del sistema e consentono quindi di effettuare l'analisi strutturale di un SED, mentre i modelli temporizzati permettono di studiare i diversi comportamenti nel tempo del sistema, pertanto sono indispensabili qualora si voglia effettuare l'analisi prestazionale di un SED. In ogni caso, si può concludere che un modello ad eventi discreti costituisce sempre una descrizione matematica finita dell'insieme infinito di tracce che rappresentano il comportamento di un sistema ad eventi discreti ad un certo livello di astrazione.

Le reti di Petri rappresentano uno degli strumenti più efficaci nell'analisi di tutto quel ramo dell'automatica che si occupa di studiare i sistemi ad eventi discreti. Esse si dividono in logiche e temporizzate. Queste ultime si dividono a loro volta in deterministiche e stocastiche. In questa tesi verranno trattate le reti di Petri logiche o reti posto/transizione (o reti P/T), che consistono in un modello logico che non consente di rappresentare la temporizzazione degli eventi, ma solo l'ordine con cui essi si verificano. Fra i vari modelli ad eventi discreti, le reti di Petri hanno un'importanza predominante a causa di vari fattori:

- Forniscono un formalismo grafico e matematico per la modellazione dei SED.
- Permettono di dare una rappresentazione compatta in termini di spazio di stato (rappresentano SED con un numero infinito di stati, mediante un grafo con un numero finito di nodi).
- Permettono di rappresentare esplicitamente il concetto di concorrenza, cioè di attività che possono venire svolte parallelamente.
- Consentono una rappresentazione modulare; cioè se un sistema è composto da più sottosistemi che interagiscono tra loro, è generalmente possibile rappresentare ciascun sottosistema come una semplice sottorete e poi, mediante operatori di rete, unire le varie sottoreti per ottenere il modello del sistema complessivo.

2.2 Struttura delle reti P/T

In questa sezione viene definita la struttura algebrica e grafica delle reti P/T.

Una rete P/T è un grafo bipartito, orientato e pesato. I due tipi di vertici sono detti: *posti* (rappresentati da cerchi) e *transizioni* (rappresentati da barre o da rettangoli). Gli archi, che devono essere orientati, connettono i posti alle transizioni e viceversa.

Definizione 2.2.1. *Una* rete P/T *è una struttura* $N = (P, T, Pre, Post)$ *dove:*

- $P = \{p_1, p_2, ..., p_m\}$ *è l'insieme degli* m posti.
- $T = \{t_1, t_2, ..., t_n\}$ *è l'insieme delle* n transizioni.
- $Pre : P \times T \longrightarrow \mathbb{N}$*: è la* funzione di pre-incidenza *che specifica gli archi diretti dai posti alle transizioni (detti archi "*pre*") e viene rappresentata mediante una matrice* $m \times n$*.*
- $Post : P \times T \longrightarrow \mathbb{N}$*: è la* funzione di post-incidenza *che specifica gli archi diretti dalle transizioni ai posti (detti archi "*$post$*") e viene rappresentata mediante una matrice* $m \times n$*.* ■

Si suppone che $P \cap T = \emptyset$, cioè posti e transizioni sono insiemi disgiunti e che $P \cup T \neq \emptyset$, cioè la rete è costituita da almeno un posto o da una transizione. Le matrici

Pre e $Post$ sono delle matrici di interi non negativi. Si denota con $Pre(\cdot, t)$ ($Post(\cdot, t)$) la colonna della matrice Pre ($Post$) relativa alla transizione t, e con $Pre(p, \cdot)$ ($Post(p, \cdot)$) la riga della matrice Pre relativa al posto p. Le matrici Pre e $Post$ possono essere compattate in un'unica matrice, detta di incidenza.

Definizione 2.2.2. *Data una rete $N = (P, T, Pre, Post)$, con m posti ed n transizioni, la* matrice di incidenza $C : P \times T \longrightarrow \mathbb{Z}$ *è la matrice $m \times n$ definita come:*

$$C = Post - Pre,$$

cioè il generico elemento di C vale $C(p, t) = Post(p, t) - Pre(p, t)$. ■

Data C potrei non essere in grado di ricostruire perfettamente il grafo, mentre date le matrici Pre e $Post$ ciò è sempre possibile.

Un esempio chiarirà questi concetti.

Esempio 2.2.1. *In Figura 2.1 è rappresentata la rete $N = (P, T, Pre, Post)$ con insieme dei posti $P = \{p_1, p_2, p_3, p_4\}$ e insieme delle transizioni $T = \{t_1, t_2, t_3, t_4, t_5\}$. Le matrici Pre e Post valgono:*

$$Pre = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \qquad Post = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

La matrice di incidenza vale:

$$C = \begin{bmatrix} 0 & -1 & 0 & 0 & 1 \\ 0 & 2 & -1 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}.$$

Si noti che $Post(p_2, t_2) = 2$ e dunque vi sono due archi che vanno dalla transizione t_2 al posto p_2. Nella figura, invece di rappresentare i due archi è usata una notazione semplificata che consiste nel rappresentare un solo arco avente per etichetta un numero (2 in questo caso) che indica la sua molteplicità. Si noti anche che tra il posto p_1 e la transizione t_1 vi è sia un arco "pre" che un arco "$post$". Si dice che p_1 e t_1 costituiscono un cappio*, cioè un ciclo orientato costituito da una sola transizione e da un solo posto. In questo caso la somma algebrica di Pre e*

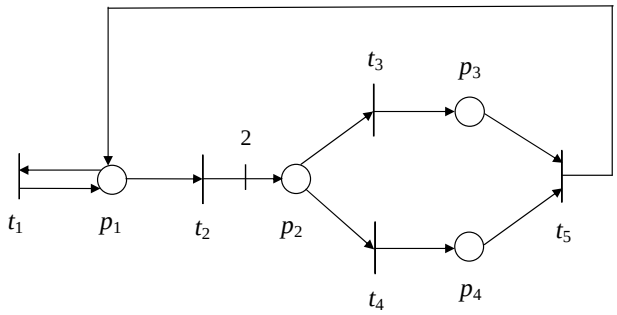

Figura 2.1: Una rete posto-transizione.

$Post$ determina un elemento $C(p_1, t_1) = 0$, cioè dall'esame della sola C non è possibile rilevare che vi sono archi tra questi due vertici. □

Infine, data una transizione si definiscono i seguenti sistemi di posti:

${}^\bullet t = \{p \in P \mid Pre(p,t) > 0\}$: è l'insieme dei posti in ingresso a t.
$t^\bullet = \{p \in P \mid Post(p,t) > 0\}$: è l'insieme dei posti in uscita da t.
${}^\bullet p = \{t \in T \mid Post(p,t) > 0\}$: è l'insieme delle transizioni in ingresso a p.
$p^\bullet = \{t \in T \mid Pre(p,t) > 0\}$: è l'insieme delle transizioni in uscita da p.

Ad esempio nella rete in Figura 2.1 vale ${}^\bullet t_2 = \{p_1\}$, $t_2^\bullet = \{p_2\}$, ${}^\bullet p_2 = \{t_2\}$, $p_2^\bullet = \{t_3, t_4\}$.

2.3 Marcatura e sistema di rete

In questa sezione vediamo come, aggiungendo ad una struttura di rete una *marcatura*, si ottiene una *rete marcata* (o *sistema di rete*), cioè un sistema ad eventi discreti, di cui si studieranno le leggi che ne governano l'evoluzione dinamica e che vedremo in seguito.

Mediante la *marcatura* è possibile definire lo stato di una rete P/T.

Definizione 2.3.1. *Una* marcatura *è una funzione $M : P \to \mathbb{N}$ che assegna ad ogni posto un numero intero non negativo di* marche *(o* gettoni*) rappresentate graficamente con dei pallini neri dentro i posti.* ■

Considerando l'esempio in Figura 2.1, una marcatura possibile M è $M(p_1) =$

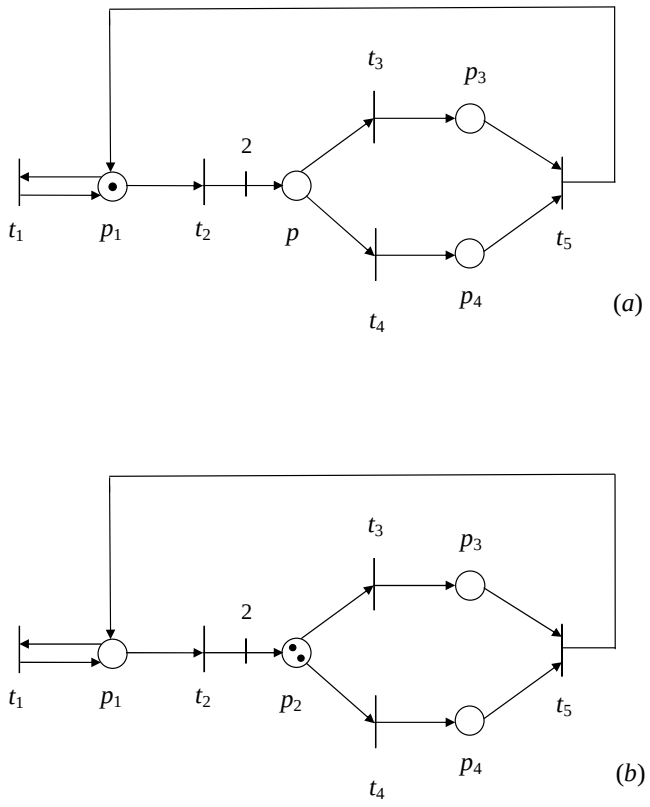

Figura 2.2: Evoluzione di una rete marcata. (a) Marcatura iniziale. (b) Marcatura raggiunta dopo lo scatto di t_2.

$1, M(p_2) = M(p_3) = M(p_4) = 0$, come mostrato in Figura 2.2(a). Un'altra marcatura possibile è quella mostrata in Figura 2.2(b), dove $M(p_1) = 0, M(p_2) = 2, M(p_3) = M(p_4) = 0$.

Definizione 2.3.2. *Una rete N con una marcatura iniziale M_0 è detta* rete marcata *o* sistema di rete*, e viene indicata come* $\langle N, M_0 \rangle$. ■

Una rete marcata è in effetti un sistema ad eventi discreti a cui è associato un comportamento dinamico come vedremo nella prossima sezione.

2.4 Abilitazione e scatto

In questa sezione vengono definiti l'abilitazione e lo scatto di una transizione.

Definizione 2.4.1. *Una transizione t è* abilitata *dalla marcatura M se*

$$M \geqslant Pre(\cdot, t),$$

cioè se ogni posto $p \in P$ della rete contiene un numero di marche pari o superiore a $Pre(p,t)$. Per indicare che t è abilitata da M si scrive $M[t\rangle$. Per indicare che t' non è stata abilitata da M si scrive $\neg M[t'\rangle$. ■

Una transizione che non possiede archi in ingresso è detta *transizione sorgente* ed è sempre abilitata.

Definizione 2.4.2. *Una transizione t abilitata da una marcatura M può* scattare. *Lo* scatto *di t rimuove $Pre(p,t)$ marche da ogni posto p appartenente a P e aggiunge $Post(p,t)$ in ogni posto p appartenente a P, determinando una nuova marcatura M'. Cioè vale:*

$$M' = M - Pre(\cdot, t) + Post(\cdot, t) = M + C(\cdot, t).$$

Per indicare che lo scatto di t da M determina la marcatura M' si scrive $M[t\rangle M'$.

■

Dall'ispezione visiva di una rete marcata è facile rendersi conto se una transizione t è abilitata: occorre che ogni posto in ingresso alla transizione, cioè ogni posto $p \in {}^\bullet t$, abbia un numero di marche pari o superiore agli archi "pre" che vanno dal posto alla transizione. Nella rete marcata in Figura 2.2(a), l'insieme delle transizioni abilitate è $\{t_1, t_2\}$. Nella rete marcata in Figura 2.2(b), l'insieme delle transizioni abilitate è $\{t_3, t_4\}$. La marcatura in Figura 2.2(b) è stata ottenuta dalla marcatura in Figura 2.2(a) con lo scatto di t_2.

Definizione 2.4.3. *Una sequenza di transizioni $\sigma = t_{j_1} t_{j_2} ... t_{j_r} \in T^*$[1] è* abilitata *da una marcatura M se: la transizione t_{j_1} è abilitata da M e il suo scatto porta a $M_1 = M + C(\cdot, t_{j_1})$; la transizione t_{j_2} è abilitata da M_1 e il suo scatto porta a $M_2 = M_1 + C(\cdot, t_{j_2})$, ecc. Una sequenza abilitata σ viene anche detta* sequenza di scatto *e ad essa corrisponde la* traiettoria*:*

$$M[t_{j_1}\rangle M_1[t_{j_2}\rangle M_2 ... [t_{j_r}\rangle M_r.$$

■

Per indicare che la sequenza σ è abilitata da M si scrive $M[\sigma\rangle$. Per indicare che lo scatto di σ da M determina la marcatura M' si scrive $M[\sigma\rangle M'$.

[1] T^* indica l'insieme di tutte le sequenze (di lunghezza 0, 1, 2, ecc.), composte dalla concatenazione di elementi di T.

Ad esempio nella rete in Figura 2.2(b) una possibile sequenza di transizioni abilitata dalla marcatura data è $\sigma = t_3t_4t_5t_1$ il cui scatto porta alla marcatura iniziale $M_0 = [\ 1 \quad 0 \quad 0 \quad 0\ ]^T$.

Da notare che in questa tesi si assume sempre che due o più transizioni non possano scattare simultaneamente (ipotesi di non-concorrenza).

Una marcatura M è *raggiungibile* in $\langle N, M_0 \rangle$ se e solo se esiste una sequenza di scatto σ tale che $M_0[\sigma\rangle M$. L'insieme di tutte le marcature raggiungibili da M_0 definisce *l'insieme di raggiungibilità* di $\langle N, M_0 \rangle$ ed è indicato con $R(N, M_0)$.

Il *vettore di scatto* di σ è denotato $\vec{\sigma}$. Il vettore di scatto è un vettore di interi non negativi a n componenti, la cui generica componente indica quante volte la transizione t_i compare in σ. Ad esempio nella rete in Figura 2.2 si consideri la sequenza $\sigma = t_1t_1t_2t_3$. Poiché t_1 compare due volte in σ, mentre t_2 e t_3 vi compaiono una sola volta e t_4 e t_5 non vi compaiono, il vettore caratteristico di tale sequenza è $\vec{\sigma} = [\ 2 \quad 1 \quad 1 \quad 0 \quad 0\ ]^T$.

Dato un sistema di rete di Petri $\langle N, M_0 \rangle$ si definisce *linguaggio-libero*[2] o *comportamento* l'insieme delle sequenze di scatto abilitate dalla marcatura iniziale:

$$L(N, M_0) = \{\sigma \in T^* \mid M_0[\sigma\rangle\}.$$

Viene anche definito l'insieme delle sequenze di scatto di lunghezza minore o uguale a $k \in \mathbb{N}$ come:

$$L_k(N, M_0) = \{\sigma \in L(N, M_0) \mid |\sigma| \leq k\},$$

dove $|\sigma|$ indica la lunghezza della sequenza σ.

2.5 Equazione di stato

La Definizione 2.4.2 permette di calcolare la marcatura raggiunta dallo scatto di una transizione mediante una semplice equazione matriciale. Questa proprietà può essere estesa a una sequenza di transizioni σ.

[2] *libero* significa che nessuna funzione di etichettatura è assegnata al sistema di rete di Petri considerato.

Definizione 2.5.1. *Sia $\langle N, M_0 \rangle$ una rete marcata e C sia la sua matrice di incidenza. Se M è raggiungibile da M_0 scattando la sequenza di transizioni σ vale:*

$$M = M_0 + C \cdot \vec{\sigma}.$$

$\vec{\sigma}$ è il vettore di scatto. ■

Questa definizione, così come la 2.4.1, ci aiuterà a capire da dove nascono i vincoli del CS quando si andrà a definire il problema di programmazione lineare.

2.6 Proprietà e classi di reti P/T

In questa sezione si definiscono e si caratterizzano alcune proprietà di una rete P/T e alcune classi di reti P/T. Queste caratteristiche possono essere usate come informazioni aggiuntive sulla rete nella procedura di identificazione come vedremo più avanti.

2.6.1 Proprietà di una rete P/T

Graficamente, è facile verificare una condizione strutturale, indipendente dalla marcatura iniziale, sufficiente affinché una rete sia strettamente conservativa.

Proposizione 2.6.1. *La rete marcata $\langle N, M_0 \rangle$ è strettamente conservativa se per ogni $t \in T$ vale*

$$\sum_{p \in P} Pre(p, t) = \sum_{p \in P} Post(p, t),$$

cioè se ogni transizione ha un numero di archi "pre" uguale al numero di archi "$post$". ■

Ripetitività. La ripetitività di una sequenza di transizioni assicura che essa possa venire ripetuta infinite volte.

Definizione 2.6.1. *Una rete marcata $\langle N, M_0 \rangle$ è* ripetitiva *se esiste una sequenza ripetitiva in $L(N, M_0)$.* ■

Definizione 2.6.2. *Una sequenza ripetitiva σ abilitata da M è detta:*

- Stazionaria *se* $M[\sigma\rangle M$.

- Crescente *se* $M[\sigma\rangle M'$ *con* $M' \gneq M$. ■

Definizione 2.6.3. *Data una rete* N *con* m *posti e* n *transizioni, sia* C *la sua matrice di incidenza. Un vettore* $\vec{x} \in \mathbb{N}^m$ *e tale che* $\vec{x} \gneq 0$ *(cioè un vettore non nullo di interi non negativi con tante componenti quanti sono i posti della rete) è detto:*

- P-vettore invariante *(o semplicemente* P-invariante*): se* $\vec{x}^T \cdot C = 0^T$.
- P-vettore crescente*: se* $\vec{x}^T \cdot C \geq 0^T$.
- P-vettore decrescente*: se* $\vec{x}^T \cdot C \leq 0^T$.

Un vettore $\vec{y} \in \mathbb{N}^n$ *e tale che* $\vec{y} \gneq 0$ *(cioè un vettore non nullo di interi non negativi con tante componenti quante sono le transizioni della rete) è detto:*

- T-vettore invariante *(o semplicemente* T-invariante*): se* $C \cdot \vec{y} = 0$.
- T-vettore crescente*: se* $C \cdot \vec{y} \geq 0$.
- T-vettore decrescente*: se* $C \cdot \vec{y} \leq 0$. ■

Si osservi che un invariante è anche un caso particolare di vettore crescente e di vettore decrescente.

Si noti, in particolare, l'importanza della nozione di P-invariante x che specifica una sorta di "legge di conservazione delle marche", assegnando però peso x_i alle marche contenute nel posto p_i.

La seguente proposizione permette di dare una interpretazione fisica ai T-vettori di una rete.

Proposizione 2.6.2. *Data una rete* N*, sia* σ *una sequenza di transizioni abilitata da una marcatura iniziale* M_0*,* $\vec{\sigma}$ *il corrispondente vettore di scatto e valga* $M_0[\sigma\rangle M$*. Valgono le seguenti proprietà:*

- $\vec{\sigma}$ *è un T-invariante* $\iff M = M_0$*, cioè* σ *è ripetitiva e stazionaria.*
- $\vec{\sigma}$ *è un T-vettore crescente (crescente e non invariante)* $\iff M \geq M_0 (M \gneq M_0)$*, cioè la sequenza* σ *è ripetitiva (ripetitiva crescente).*
- $\vec{\sigma}$ *è un T-vettore decrescente (decrescente e non invariante)* $\iff M \leq M_0 (M \lneq M_0)$. ■

Limitatezza strutturale
Tale proprietà può essere caratterizzata in termini di P-vettori decrescenti. Sia N una rete P/T e $p_i \in P$ un suo posto.

1. Il posto p_i è strutturalmente limitato se e solo se esiste un P-vettore decrescente $\vec{x}$ con $x_i > 0$.

2. La rete N è strutturalmente limitata se e solo se esiste un P-vettore decrescente di interi positivi $\vec{x} \in \mathbb{N}_+^m$.

Conservatività strutturale
Tale proprietà può essere caratterizzata in termini di P-invarianti.

Proposizione 2.6.3. *Sia N una rete P/T.*

1. *N è strutturalmente strettamente conservativa se e solo se il vettore $\vec{1} = \{1\}^m$ è un P-invariante.*

2. *N è strutturalmente conservativa se e solo se esiste un P-invariante di interi positivi $\vec{x} \in \mathbb{N}_+^m$.* ■

Si noti infine che la conservatività strutturale implica la limitatezza strutturale ma non è vero il viceversa.

Ripetitività strutturale e consistenza
Tali proprietà sono gli equivalenti strutturali delle proprietà di ripetitività e stazionarietà viste per le sequenze di transizioni.

Possono essere caratterizzate in termini di T-vettori.

Proposizione 2.6.4. *Sia N una rete P/T.*

1. *N è ripetitiva se e solo se ammette un T-vettore crescente di interi positivi $\vec{y} \in \mathbb{N}_+^n$.*

2. *N è consistente se e solo se ammette un T-invariante di interi positivi $\vec{y} \in \mathbb{N}_+^n$.* ■

2.6.2 Classi di reti P/T

La definizione di rete posto/transizione data in questo capitolo corrisponde a un modello che è talvolta chiamato *rete di Petri generale*. È possibile tuttavia considerare classi di reti in cui si pongono alcune restrizioni strutturali. In questa sottosezione vengono richiamate alcune di queste strutture.

Definizione 2.6.4. *Una rete P/T $N = (P, T, Pre, Post)$ è detta:*

- Ordinaria *se $Pre : P \times T \longrightarrow \{0, 1\}$ e $Post : P \times T \longrightarrow \{0, 1\}$, cioè se ogni arco ha molteplicità unitaria.*
- Pura *se per ogni posto p e transizione t vale $Pre(p, t) \cdot Post(p, t) = 0$, cioè se la rete non contiene alcun cappio.*
- Ristretta *se è ordinaria e pura.* ■

Definizione 2.6.5. *Una* macchina di stato *(o* grafo di stato*) è una rete ordinaria in cui ogni transizione ha esattamente un arco in ingresso e un arco in uscita, cioè per ogni $t \in T$, $\sum_{p \in P} Pre(p, t) = \sum_{p \in P} Post(p, t) = 1$.* ■

La particolare struttura delle macchine di stato porta a un modello che è più limitato delle reti ordinarie, nel senso che non è possibile in generale trovare una macchina di stato equivalente a una rete ordinaria data.

Proposizione 2.6.5. *Si supponga che N sia una macchina di stato connessa*[3] *(ma non necessariamente fortemente connessa*[4]*).*

1. *Il vettore $\vec{1}$ di dimensione $m \times 1$ è il P-invariante minimale di N, cioè non esiste un P-invariante $\vec{x}' \in \mathbb{N}^m$ tale che $\vec{x}' \lneq \vec{1}$. Ciò implica che una macchina di stato è sempre* strettamente conservativa*.*
2. *Sia $\Gamma = \{\gamma_1, ..., \gamma_r\}$ l'insieme dei cicli elementari orientati di N, e per ogni ciclo $\gamma_i \in \Gamma$ si consideri il vettore caratteristico $\vec{y}^{(i)}$ delle transizioni lungo il ciclo, avente dimensione $n \times 1$ cioé*

$$\vec{y}^{(i)} = \begin{cases} y_j^{(i)} = 1 & \text{se } t_j \in \gamma_i \\ y_j^{(i)} = 0 & \text{se } t_j \notin \gamma_i \end{cases}$$

[3] Un grafo si dice connesso se per ogni coppia di nodi (v, w) esiste un percorso che li unisce. Nelle reti di Petri il grafo è la rete, i nodi sono i posti e le transizioni, i percorsi sono gli archi.

[4] Un grafo si dice fortemente connesso se per ogni coppia di nodi (v, w) esiste un percorso orientato che li unisce.

Ogni $\vec{y}^{(i)}$ è un T-invariante minimale di N. ■

Definizione 2.6.6. *Un* grafo marcato *(o* grafo di eventi*) è una rete ordinaria in cui ogni posto ha esattamente un arco in ingresso e un arco in uscita, cioè per ogni* $p \in P$, $\sum_{t \in T} Pre(p,t) = \sum_{t \in T} Post(p,t) = 1$. ■

Si noti che una rete può essere allo stesso tempo una macchina di stato e un grafo marcato.

Proposizione 2.6.6. *Si supponga che N sia un grafo marcato connesso (ma non necessariamente fortemente connesso).*

1. *Sia γ_i un ciclo elementare orientato di N. Il vettore caratteristico $\vec{x}^{(i)}$ di dimensione $m \times 1$ che conta i posti lungo γ_i è un P-invariante minimale di N.*

2. *Il vettore $\vec{1}$ di dimensione $n \times 1$ è il T-invariante minimale di N. Dunque data una sequenza σ abilitata da una marcatura M, essa è stazionaria se e solo se contiene tutte le transizioni un egual numero di volte.* ■

Capitolo 3

Richiami alla programmazione lineare

3.1 Introduzione

In questo capitolo vengono fatti dei richiami alla programmazione lineare prendendo come spunto il libro [18]. Viene fatto un sommario delle idee di base della programmazione lineare e del metodo del simplesso, viene trattato il metodo del punto interno e viene fatto qualche cenno alla programmazione intera.

La programmazione lineare rappresenta forse *il* modello di base dell'ottimizzazione vincolata. Il metodo del simplesso per risolvere problemi di programmazione lineare è stata una delle più significative scoperte algoritmiche del 900. Sviluppato da Dantzig nel 1947, il metodo del simplesso è stato applicato a migliaia di applicazioni in campi diversi come l'agricoltura, le comunicazioni, l'informatica, l'ingegneria, la finanza, ecc.

Un programma lineare è un problema di ottimizzazione con una funzione obiettivo lineare, un insieme di vincoli lineari e un insieme di restrizioni non negative imposte sulle variabili di decisione. Quindi, è un modello di ottimizzazione della forma

$$\begin{cases} Minimizza & \sum_{j=1}^{q} c_j x_j & \\ tale\ che & \sum_{j=1}^{q} a_{ij} x_j = b_i & \forall i = 1, ..., p \\ & x_j \geq 0 & \forall j = 1, ..., q \end{cases} \tag{3.1}$$

Questo problema ha q variabili di decisione x_j non negative e p vincoli di uguaglianza. Assumiamo, moltiplicando l'i-esima equazione per -1, se necessario, che il termine noto b_i di ogni vincolo $i = 1, ..., p$ sia non negativo. Possiamo notare che potremmo formulare un programma lineare in diversi modi. Per esempio, la funzione obiettivo potrebbe essere definita nella forma di massimizzazione e i vincoli in una forma minore-uguale o maggiore-uguale. In letteratura frequentemente ci si riferisce alla formulazione (3.1) come alla forma standard di un programma lineare.

In notazione matriciale, il modello di programmazione lineare ha la seguente forma:

$$\begin{cases} Minimizza & \vec{c} \cdot \vec{x} \\ tale\ che & \mathcal{A} \cdot \vec{x} = \vec{b} \\ & x_j \geq 0. \end{cases}$$

In questa formulazione la matrice $\mathcal{A} = a_{ij}$ ha p righe e q colonne, il vettore $\vec{c} = (c_j)$ è un vettore riga q-dimensionale, i vettori $\vec{x} = (x_j)$ e $\vec{b} = (b_i)$ sono vettori colonna q- e p-dimensionali, rispettivamente. Assumiamo che le righe di $\mathcal{A}$ siano linearmente indipendenti, cioè la matrice $\mathcal{A}$ ha rango pieno, perciò il sistema $\mathcal{A} \cdot \vec{x} = \vec{b}$ non contiene equazioni ridondanti.

3.2 Procedura di soluzione grafica

I programmi lineari che coinvolgono solo due o tre variabili hanno una conveniente rappresentazione grafica, che aiuta a capire la natura della programmazione lineare e del metodo del simplesso. Illustriamo brevemente questa procedura ricorrendo al seguente esempio:

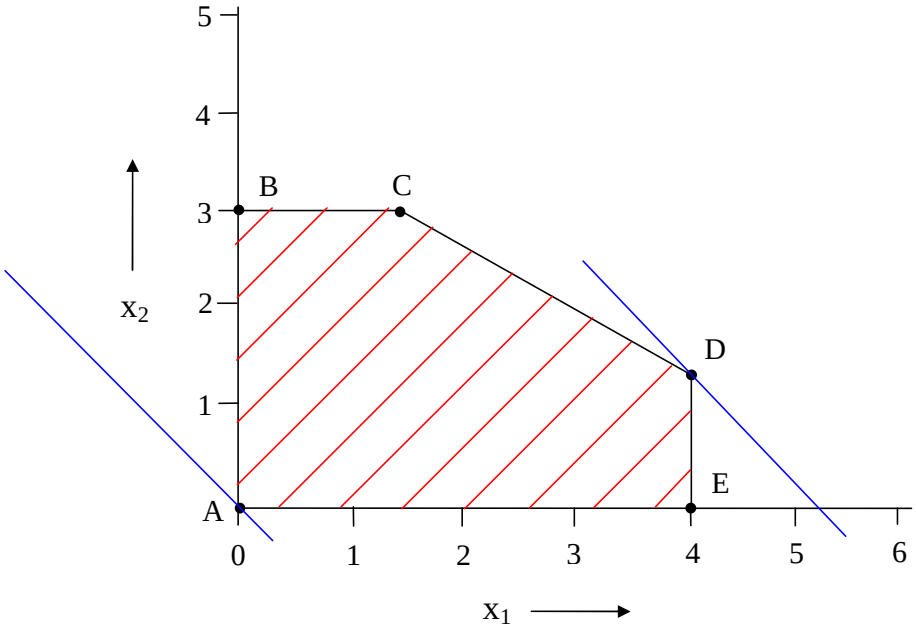

Figura 3.1: Insieme delle soluzioni ammissibili per un programma lineare.

$$\begin{cases} \textit{Massimizza} & z = x_1 + x_2 \\ \textit{tale che} & 2x_1 + 3x_2 \leq 12 \\ & x_1 \leq 4 \\ & x_2 \leq 3 \\ & x_1, x_2 \geq 0. \end{cases}$$

La regione a striscie rosse in Figura 3.1, detta poliedro, è l'insieme delle soluzioni ammissibili per questo problema. I punti A, B, C, D ed E sono i punti estremi del poliedro, formati dall'intersezione delle rette corrispondenti ai vari vincoli. Il problema di programmazione lineare consiste nel cercare un punto (x_1, x_2) nel poliedro $ABCDEA$ che realizza il massimo valore possibile di $x_1 + x_2$. equivalentemente, desideriamo determinare il più grande valore di z per il quale la retta $x_1 + x_2 = z$ ha almeno un punto in comune con il poliedro $ABCDEA$. Le rette ottenute per differenti valori di z sono parallele (linee blu in figura). Poiché queste rette si allontanano dall'origine all'aumentare di z, per massimizzare $x_1 + x_2$ dobbiamo far scorrere la retta $x_1 + x_2 = z$ il più lontano possibile dall'origine in modo che essa abbia qualche punto in comune col poliedro $ABCDEA$. Possiamo fare così fino a che non raggiungiamo qualche punto estremo, il punto D nel nostro caso, in cui la retta ha solo un punto in comune con il poliedro. A que-

sto punto, per qualsiasi ulteriore traslazione della retta lontano dall'origine, non importa quanto piccola, la retta non ha punti in comune con il poliedro. Perciò il punto D è una soluzione ottimale del problema di programmazione lineare. Il valore della sua funzione obiettivo è 16/3.

3.3 Soluzioni di base ammissibili

La descrizione del metodo del simplesso richiede che il programma lineare da risolvere sia enunciato nella forma standard (3.1). Qualunque programma lineare non standard può essere portato in tale forma mediante semplici trasformazioni. Durante la sua esecuzione, il metodo del simplesso modifica il programma lineare originale eseguendo una serie di una o più delle seguenti operazioni riga elementari:

1. Moltiplicare una riga (un vincolo) per una costante.
2. Aggiungere una riga ad un'altra riga o alla funzione obiettivo.

Poiché sono stati definiti tutti i vincoli in forma di uguaglianza, le operazioni riga non modificano l'insieme delle soluzioni ammissibili del programma lineare. Illustriamo quanto detto con un esempio.

Esempio 3.3.1. *Consideriamo il seguente programma lineare:*

$$\begin{cases} Minimizza & z(\vec{x}) = x_1 + x_2 - 8x_3 + 6x_4 \\ tale\ che & 2x_1 + x_2 - 14x_3 + 10x_4 = 16 \\ & x_1 + x_2 - 11x_3 + 7x_4 = 10 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{cases} \tag{3.2}$$

Se sottraiamo il secondo vincolo di (3.2) da $z(\vec{x})$ e dal primo otteniamo il seguente programma lineare equivalente:

$$\begin{cases} Minimizza & z(\vec{x}) = 0x_1 + 0x_2 + 3x_3 - x_4 + 10 \\ tale\ che & x_1 \quad\quad - 3x_3 + 3x_4 = 6 \\ & x_1 + x_2 - 11x_3 + 7x_4 = 10 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{cases} \tag{3.3}$$

Se sottraiamo il primo vincolo di (3.3) dal secondo otteniamo un'altra formulazione equivalente di questo programma lineare:

$$\begin{cases} Minimizza & z(\vec{x}) = 0x_1 + 0x_2 + 3x_3 - x_4 + 10 \\ tale\ che & x_1 \quad - 3x_3 + 3x_4 = 6 \\ & \quad x_2 - 8x_3 + 4x_4 = 4 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{cases} \tag{3.4}$$

Poiché il programma lineare (3.4) è equivalente al (3.2), entrambi hanno le stesse soluzioni ottimali.

Da notare che il termine noto nella funzione obiettivo tecnicamente rende la funzione non più lineare, ma strettamente affine. In ogni caso quest'aspetto può essere ignorato ai fini della minimizzazione. □

Un programma lineare si dice in forma canonica se soddisfa la seguente proprietà canonica.

Proprietà canonica. La formulazione del problema ha una variabile di decisione isolata in ogni vincolo; la variabile isolata in un dato vincolo ha un coefficiente di +1 in quel vincolo e non appare nè negli altri vincoli nè nella funzione obiettivo.

La precedente formulazione soddisfa la proprietà canonica perché in (3.4) x_1 è isolato nel primo vincolo e x_2 nel secondo. La forma canonica in (3.4) ha la seguente caratteristica interessante. L'assegnazione di valori arbitrari a x_3 e x_4 determina univocamente i valori di x_1 e x_2. Infatti, porre $x_3 = x_4 = 0$ immediatamente fornisce la soluzione $x_1 = 6$ e $x_2 = 4$. Soluzioni come queste, note come *soluzioni di base ammissibili*, giocano un ruolo fondamentale nel metodo

del simplesso. In generale, data una forma canonica per qualunque programma lineare, otteniamo una soluzione di base ponendo la variabile isolata nel vincolo i, chiamata la i-esima *variabile di base*, uguale al termine noto dell'i-esimo vincolo, e ponendo le rimanenti variabili, chiamate *non di base*, tutte a 0. L'insieme delle variabili di base è noto come *base*.

In generale risulta

$$\vec{x}_{\mathbf{B}} = \mathcal{B}^{-1} \cdot \vec{b} \quad e \quad \vec{x}_{\mathbf{L}} = 0,$$

dove $\mathbf{B} = \{1, 2, ..., p\}$ denota l'insieme indice delle variabili di base, $\mathbf{L} = \{p + 1, p+2, ..., q\}$ denota l'insieme indice delle variabili non di base e $\mathcal{B} = [\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_p]$ partiziona la matrice $\mathcal{A}$ in colonne che devono essere linearmente indipendenti. Chiamiamo questa struttura *soluzione di base ammissibile* se il valore di ogni variabile di base è non negativo ($\vec{x}_{\mathbf{B}} \geq 0$).

Supponiamo che $\vec{c}_{\mathbf{B}} = \{c_1, c_2, ..., c_p\}$ e $\vec{c}_{\mathbf{L}} = \{c_{p+1}, c_{p+2}, ..., c_q\}$ indichino i vettori di costo associati alle variabili di base e non di base rispettivamente. In una forma canonica di un programma lineare ogni variabile di base ha un coefficiente nullo nella funzione obiettivo. Otteniamo questa forma speciale della funzione obiettivo eseguendo una sequenza di operazioni riga elementari. In notazione matriciale viene selezionato un vettore $\vec{\pi}$ tale che

$$\vec{\pi} \cdot \mathcal{B} = \vec{c}_{\mathbf{B}}.$$

Chiamiamo gli elementi di $\vec{\pi}$ *moltiplicatori del simplesso* associati alla base $\mathbf{B}$ e $c_j^{\vec{\pi}} = c_j - \sum_{i=1}^{p} \pi_i a_{ij}$ costo ridotto della variabile x_j.

Per concludere questa sezione mostriamo che punti estremi e soluzioni di base sono manifestazioni geometriche e algebriche dello stesso concetto. Quindi, trovando una soluzione di base ottimale stiamo ottenendo una soluzione punto estremo ottimale.

Teorema 3.3.1. ***(Punti estremi e soluzioni di base ammissibili).*** *Una soluzione ammissibile $\vec{x}$ per un programma lineare è un punto estremo se e solo se le colonne $\{\mathcal{A}_j : x_j > 0\}$ della matrice dei vincoli corrispondenti alle variabili strettamente positive sono linearmente indipendenti.* ■

3.4 Metodo del simplesso

Il metodo del simplesso mantiene una soluzione di base ammissibile ad ogni passo. Data una soluzione di base ammissibile, il metodo prima applica i criteri di ottimalità per verificare l'ottimalità della soluzione corrente. Se la soluzione corrente non soddisfa questa condizione, l'algoritmo esegue un'operazione, nota come *operazione pivot*, per ottenere un'altra struttura di base con un costo inferiore o identico. Il metodo del simplesso itera questo processo fino a che la corrente soluzione di base ammissibile soddisfa i criteri di ottimalità. Descriviamo questi passi senza entrare in troppi dettagli.

Struttura base iniziale
Utilizzando una semplice tecnica, possiamo trovare una soluzione di base ammissibile di un programma lineare collegato a quello standard e usarla per inizializzare il metodo del simplesso. Questa tecnica consiste nell'introdurre una *variabile artificiale* x_{q+i} con un costo M sufficientemente grande per ogni vincolo i e definendo un programma lineare aumentato come:

$$\begin{cases} Minimizza & \sum_{j=1}^{q} c_j x_j + \sum_{j=q+1}^{q+p} M x_j & \\ tale\ che & \sum_{j=1}^{q} a_{ij} x_j + x_{q+i} = b_i & \forall i = 1, ..., p \\ & x_j \geq 0 & \forall j = 1, ..., q. \end{cases} \tag{3.5}$$

È facile mostrare che il programma lineare originale (3.1) ha una soluzione ammissibile se e solo se ogni variabile artificiale ha valore zero in ogni soluzione ottimale del programma lineare aumentato (3.5). L'insieme di tutte le variabili artificiali costituisce una base iniziale e il valore dell'i-esima variabile di base è $b_i \geq 0$.

Criteri di ottimalità
Nella forma canonica associata a questa struttura base la funzione obiettivo è

$$Minimizza \quad z(\vec{x}) = z_0 + \sum_{j=p+1}^{q} c_j^{\vec{\pi}} x_j.$$

Il coefficiente $c_j^{\vec{\pi}}$ è il costo ridotto della variabile non di base x_j rispetto ai moltiplicatori del simplesso correnti $\vec{\pi}$. Si può affermare che, se $c_j^{\vec{\pi}} \geq 0$ per ogni variabile non di base x_j, la soluzione di base ammissibile corrente $\vec{x}$ è una soluzione ottimale del programma lineare.

Operazione pivot
Se $c_j^{\vec{\pi}} < 0$ per qualche variabile non di base x_j, la soluzione di base ammissibile corrente può non essere ottimale. Il metodo del simplesso seleziona una tale variabile non di base, detta x_s, e prova ad aumentare il suo valore sino a una quantità θ. Quando il metodo del simplesso aumenta x_s il più possibile mantenendo tutte le altre variabili non di base a zero, qualche variabile di base, detta x_r, raggiunge il valore zero. Quest'ultima viene rimpiazzata da x_s, definendo una nuova struttura di base. Allora il metodo aggiorna l'inverso della base e ripete i calcoli. Poiché $c_s^{\vec{\pi}} < 0$, ponendo x_s grande quanto vogliamo, possiamo rendere la funzione obiettivo arbitrariamente piccola, al limite $-\infty$. In questo caso si dice che il programma lineare ha una soluzione illimitata.

Adesso focalizziamo l'attenzione sulle situazioni in cui θ è finito. Se poniamo $x_s = \theta$, come detto una delle variabili di base, detta x_r, diventa zero. Chiamiamo x_r *variabile uscente*. In pratica θ è il valore minimo dei rapporti tra il termine noto e il coefficiente di x_s in ogni vincolo: *regola del minimo rapporto*. Quindi definiamo x_s, che chiamiamo *variabile entrante*, come variabile di base, x_r come variabile non di base, e aggiorniamo la forma canonica del programma lineare in modo che soddisfi la proprietà canonica rispetto alla nuova base. Facciamo questo eseguendo una sequenza di operazioni riga elementari.

Aggiornamento della tabella del simplesso
Nella nuova base, la variabile entrante x_s diventa la variabile di base per l'r-esima riga, che richiede che nella nuova forma canonica la variabile x_s abbia un coefficiente +1 nell'r-esima riga e un coefficiente 0 in tutte le altre righe. Il coefficiente di x_s nella funzione obiettivo diventa zero. Denotiamo questo insieme di calcoli come un'*operazione pivot*.

Chiariamo ora questi passi del metodo del simplesso con un esempio.

Esempio 3.4.1. *Consideriamo il programma lineare (3.4) dell'Esempio 3.3.1.*

$$\begin{cases} Minimizza & z(\vec{x}) = 0x_1 + 0x_2 + 3x_3 - x_4 + 10 \\ tale\ che & x_1 \quad - 3x_3 + 3x_4 = 6 \\ & x_2 - 8x_3 + 4x_4 = 4 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{cases}$$

Le variabili di base sono x_1 e x_2. La variabile non di base x_4 ha un costo ridotto negativo e la selezioniamo come variabile entrante. Applicando la regola del minimo rapporto, troviamo che $\theta = min\{6/3, 4/4\} = 1$. Il valore minimo è realizzato nel secondo vincolo, che contiene la variabile di base x_2. La variabile x_2 è la variabile uscente. Nella base successiva, x_1 e x_4 sono le variabili di base, quindi è necessario modificare la forma canonica. Poiché x_4 è la nuova variabile di base per il secondo vincolo, dividiamo questo vincolo per 4 in modo che x_4 abbia coefficiente +1. Successivamente moltiplichiamo tale vincolo modificato prima per -3 e sommiamo il risultato al primo vincolo, e poi per +1 e sommiamo il risultato a $z(\vec{x})$. Queste operazioni producono la seguente (equivalente) formulazione del programma lineare:

$$\begin{cases} Minimizza & z(\vec{x}) = 0x_1 + 0x_4 + 1/4x_2 + x_3 + 9 \\ tale\ che & x_1 \quad - 3/4x_2 + 3x_3 = 3 \\ & x_4 + 1/4x_2 - 2x_3 = 1 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{cases}$$

In questa forma canonica, i costi ridotti di entrambe le variabili non di base x_2 e x_3 sono non negativi, quindi la corrente soluzione di base ammissibile, $x_1 = 3, x_4 = 1$ e $x_2 = x_3 = 0$, è ottimale. Questa soluzione ha un valore della funzione obiettivo pari a 9. □

Un modo di eseguire l'operazione pivot è di fare i calcoli iterativamente sulla matrice $\mathcal{A}$. Questo insieme di calcoli può essere molto costoso per i programmi lineari che contengono molte variabili (come è usuale nella pratica e quindi anche in questa tesi quando andremo a trattare degli esempi di test su varie reti). Il *metodo del simplesso revisionato* (implementato anche in GLPK) è una particolare implementazione del metodo del simplesso che ci permette di evitare molti di questi calcoli. Per descrivere l'approccio base del metodo del simplesso revisionato, supponiamo (concettualmente) di aggiungere un insieme (vettore) di variabili fittizie $\vec{y}$ al programma lineare originale e formare un nuovo programma lineare coi vincoli:

$$\begin{cases} \mathcal{A} \cdot \vec{x} + \mathcal{I} \cdot \vec{y} = \vec{b} \\ \vec{x} \geq 0, \vec{y} \geq 0. \end{cases}$$

In questa formulazione $\mathcal{I}$ è una matrice identità. Se dovessimo eseguire l'operazione pivot sull'intera matrice $\mathcal{A}$ passo per passo, i coefficienti delle variabili fittizie y_i sarebbero la base inversa. Questa osservazione mostra che non abbiamo bisogno di effettuare le operazioni pivot sull'intera matrice $\mathcal{A}$. Invece, possiamo eseguire queste operazioni sulle colonne associate alla matrice identità iniziale $\mathcal{I}$ (non vengono introdotte formalmente le variabili y_i). Poiché i calcoli risultanti forniscono la base inversa $\mathcal{B}^{-1}$, possiamo usare questa matrice per calcolare i moltiplicatori del simplesso $\vec{\pi} = \vec{c}_{\mathbf{B}} \cdot \mathcal{B}^{-1}$ e poi utilizzarli per calcolare il costo ridotto di ogni variabile. Dopo aver identificato la variabile x_r che deve lasciare la base, eseguiamo le operazioni riga elementari sulla matrice $\mathcal{B}^{-1}$ corrente e otteniamo la base inversa aggiornata. Questa modifica nell'algoritmo, apparentemente poco rilevante, spesso ha drammatici effetti sulla sua efficienza perché i dati originali $\mathcal{A}$ per la maggior parte dei problemi incontrati nella pratica sono molto sparsi nel senso che il 90% o più dei suoi coefficienti sono nulli. Nel metodo del simplesso revisionato, utilizzando appropriate strutture dati, possiamo evitare tutti i calcoli corrispondenti ad elementi nulli risparmiando parecchio tempo.

Conclusioni
In sintesi, il metodo del simplesso si muove da una soluzione di base ammissibile all'altra eseguendo un'operazione pivot. Ad ogni iterazione il valore della soluzione migliora della quantità $c_s^{\vec{\pi}}\theta$. Se $\theta > 0$ si dice che il pivot è *non degenere*, altrimenti si dice *degenere*. Un pivot non degenere decrementa il valore della funzione obiettivo associato alla soluzione di base ammissibile. Se ogni pivot è non degenere il metodo del simplesso terminerà dopo un numero finito di passi perché un programma lineare ha al più un numero finito di soluzioni di base ammissibili distinte e l'algoritmo non può mai ripetere una soluzione di base ammissibile (poiché il valore della funzione obiettivo è strettamente decrescente). Tuttavia, il metodo del simplesso può eseguire pivot degeneri e, senza ulteriori modifiche, può ripetere una soluzione di base ammissibile e quindi non terminare in un tempo finito. Fortunatamente i ricercatori hanno sviluppato diversi modi per implementare il metodo del simplesso in modo che converga in un tempo finito. Il metodo del simplesso termina con uno dei seguenti tre risultati:

- **Caso 1.** Il metodo termina con una soluzione illimitata. In questo caso il programma lineare non ha una soluzione ottimale con un valore finito.

- **Caso 2.** Il metodo termina con una soluzione finita in cui qualche varia-

bile artificiale ha un valore positivo. In questo caso il programma lineare originale non ha una soluzione ammissibile.

- **Caso 3.** Il metodo termina con una soluzione finita in cui tutte le variabili artificiali hanno valore nullo. Questa soluzione è una soluzione ottimale del programma lineare originale.

3.5 Programmazione lineare duale

Ogni problema di programmazione lineare, che chiameremo *problema primario*, ha un problema di programmazione lineare associato strettamente affine, chiamato *problema duale*, e l'insieme di questi due problemi definisce una teoria della dualità che è una parte importante della programmazione lineare (il metodo del simplesso duale è implementato anche in GLPK). Assumiamo che il programma lineare originale sia stato enunciato nella seguente forma:

$$\begin{cases} Minimizza & \sum_{j=1}^{q} c_j x_j & \\ tale\ che & \sum_{j=1}^{q} a_{ij} x_j \geq b_i & \forall i = 1, ..., p \\ & x_j \geq 0 & \forall j = 1, ..., q. \end{cases} \tag{3.6}$$

In questa formulazione permettiamo a b_i di avere un segno arbitrario. Definiamo questa forma la *forma simmetrica* di un programma lineare. È possibile mostrare che possiamo convertire qualsiasi programma lineare in forma non simmetrica nella forma simmetrica. Per definire il problema duale associamo una variabile duale π_i ad ogni i-esimo vincolo in (3.6). Rispetto a queste variabili il problema duale è:

$$\begin{cases} Massimizza & \sum_{i=1}^{p} b_i \pi_i & \\ tale\ che & \sum_{i=1}^{p} a_{ij} \pi_i \leq c_j & \forall j = 1, ..., q \\ & \pi_i \geq 0 & \forall i = 1, ..., p. \end{cases}$$

Da notare che ogni vincolo nel primario ha una variabile associata nel duale e che il termine noto di questo vincolo diventa il coefficiente di costo della variabile associata. Inoltre ogni variabile nel primario ha un vincolo associato nel duale e il coefficiente di costo di questa variabile diventa il termina noto del vincolo associato. Il primo risultato concernente la coppia primario-duale è noto come *teorema della dualità debole*.

Teorema 3.5.1. ***(Teorema della dualità debole).*** *Se $\vec{x}$ è una qualunque soluzione ammissibile del problema primario e $\vec{\pi}$ è una qualunque soluzione ammissibile del problema duale, allora $\sum_{i=1}^{p} b_i \pi_i \leq \sum_{j=1}^{q} c_j x_j$.* ■

Grazie al teorema della dualità debole si possono enunciare i seguenti risultati:

1. Il valore della funzione obiettivo di qualunque soluzione duale ammissibile è un limite inferiore sul valore della funzione obiettivo di ogni soluzione primaria ammissibile.

2. Se il problema primario ha una soluzione illimitata, il problema duale è inammissibile.

3. Se il problema duale ha una soluzione illimitata, il problema primario è inammissibile.

4. Se il problema primario ha una soluzione ammissibile $\vec{x}$ e il problema duale ha una soluzione ammissibile $\vec{\pi}$ e $\sum_{i=1}^{p} b_i \pi_i = \sum_{j=1}^{q} c_j x_j$, allora $\vec{x}$ è una soluzione ottimale del problema primario e $\vec{\pi}$ è una soluzione ottimale del problema duale.

Il secondo risultato concernente la coppia primario-duale è noto come *teorema della dualità forte*.

Teorema 3.5.2. ***(Teorema della dualità forte).*** *Il problema primario ammette soluzione ottimale finita se e solo se il problema duale ammette soluzione ottimale finita. In tal caso entrambi hanno gli stessi valori della funzione obiettivo.* ■

Proprietà di rilassatezza complementare. Si dice che una coppia $(\vec{x}, \vec{\pi})$ di soluzioni ammissibili primaria e duale soddisfa la proprietà di rilassatezza complementare se:

$$\pi_i \left[\sum_{j=1}^{q} a_{ij} x_j - b_i \right] = 0 \qquad \forall i = 1, ..., p$$

e

$$x_j \left[c_j - \sum_{i=1}^{p} a_{ij} \pi_i \right] = 0 \qquad \forall j = 1, ..., q.$$

Osserviamo dalla formulazione (3.6) del problema primario che $[\sum_{j=1}^{q} a_{ij} x_j - b_i]$ è la quantità di rilassamento nell'i-esimo vincolo primario e π_i è la variabile duale associata a questo vincolo. Similmente, $[c_j - \sum_{i=1}^{p} a_{ij} \pi_i]$ è la quantità di rilassamento nel j-esimo vincolo duale e x_j è la variabile primaria associata a questo vincolo.

La proprietà di rilassatezza complementare afferma che per ogni vincolo primario e duale il prodotto del rilassamento nel vincolo per la sua variabile primaria o duale associata è zero.

Teorema 3.5.3. ***(Condizioni di ottimalità della rilassatezza complementare).*** *Una soluzione ammissibile primaria $\vec{x}$ e una soluzione ammissibile duale $\vec{\pi}$ sono soluzioni ottimali dei problemi primario e duale se e solo se queste soddisfano la proprietà di rilassatezza complementare.*

3.6 Metodo del punto interno

In pratica l'algoritmo del simplesso funziona molto bene, ma in teoria ha una complessità computazionale che non è polinomiale nelle dimensioni del problema. In

particolare si possono costruire speciali istanze in cui l'algoritmo richiede di visitare un numero di vertici esponenziale nelle dimensioni del problema. Uno strumento che spesso viene preferito al metodo del simplesso per problemi di grandi dimensioni è il metodo del punto interno (implementato anche in GLPK). Questo si adatta specialmente a problemi di programmazione lineare molto sparsi e consente di risolvere tali problemi molto più velocemente del metodo del simplesso. In breve, il metodo procede eseguendo i seguenti passaggi:

1. Si introducono tante variabili ausiliarie o di scarto quante sono le disequazioni che rappresentano i vincoli in modo che ciascuna di esse diventi un'equazione.

2. Si determinano tutte le soluzioni di base, che vengono determinate da punti aventi $n + m$ coordinate (dove n è il numero delle variabili iniziali, o d'azione, ed m è il numero delle variabili ausiliarie o di scarto). Tra queste $n + m$ coordinate n devono essere sempre nulle.

3. Per determinare il numero delle soluzioni di base basta calcolare il numero delle combinazioni di $n + m$ elementi presi ad n ad n.

4. Si scelgono tra le soluzioni di base quelle ammissibili date dai punti precedenti che hanno le coordinate tutte positive, poiché le variabili sono positive.

5. Si sostituiscono le coordinate di tali punti nella funzione obiettivo data e si sceglie quel punto che porta al valore massimo oppure minimo come si desidera dal genere della funzione obiettivo data.

Facciamo un esempio pratico.

Esempio 3.6.1. *Si desidera risolvere il seguente programma lineare:*

$$\begin{cases} Minimizza & z(\vec{x}) = 5x_1 + 3x_2 + x_3 \\ tale\ che & x_1 + 2x_2 - x_3 \leq 15 \\ & 2x_1 - x_2 + x_3 \leq 10 \\ & x_1, x_2, x_3 \geq 0. \end{cases}$$

I passi da compiere sono i seguenti:

1. *Poiché ci sono 2 vincoli si introducono 2 variabili ausiliarie, x_4 e x_5, e quindi i vincoli diventano*

$$\begin{cases} x_1 + 2x_2 - x_3 + x_4 = 15 \\ 2x_1 - x_2 + x_3 + x_5 = 10 \\ x_1, x_2, x_3, x_4, x_5 \geq 0. \end{cases}$$

2. *Poiché ci sono 3 variabili iniziali e 2 variabili ausiliarie, il numero delle soluzioni di base è dato dal numero delle combinazioni di 5 elementi presi a 3 a 3, quindi è pari a 10.*

3. *Le soluzioni di base si trovano attribuendo alle 5 (numero totale di variabili) coordinate dei rispettivi punti alternativamente 3 (numero di variabili iniziali) zeri e calcolando le rimanenti 2 coordinate attraverso la funzione obiettivo data ed attraverso le 2 equazioni che rappresentano i vincoli. Si avranno in questo caso i seguenti 10 punti:* $A1(0,0,0,15,10)$; $A2(7,4,0,0,0)$; $A3(25/3,0,-20/3,0,0)$; $A4(5,0,0,10,0)$; $A5(15,0,0,0,-20)$; $A6(0,25,35,0,0)$; $A7(0,10,0,35,0)$; $A8(0,15/2,0,0,35/2)$; $A9(0,0,10,25,0)$; $A10(0,0,-15,0,25)$. *Notare che dei punti precedenti si devono scartare i punti* $A3$, $A5$ *e* $A10$ *in quanto contengono nelle loro coordinate valori negativi. Quindi le soluzioni di base ammissibili sono:* $A1, A2, A4, A6, A7, A8$ *e* $A9$.

4. *Sostituiti i suddetti punti nella funzione obiettivo data, si ricava il minimo che si trova in* $A1$ *e vale 0.* □

3.7 Cenni alla programmazione intera

La programmazione intera risolve sempre problemi lineari, ma si limita alla risoluzione di quelli che presentano al loro interno solo variabili intere, cioè variabili che possono assumere solo i valori interi contenuti all'interno del loro dominio di esistenza. Generalmente i problemi di programmazione intera sono NP-*hard*, ciò significa che molti problemi di questo tipo richiedono, nel caso peggiore, un tempo di computazione della soluzione esponenziale rispetto alla dimensione dei dati in ingresso. In pratica si tratta di problemi "difficili" da risolvere. Tuttavia alcuni di questi, di uso comune nelle applicazioni, sono stati studiati a fondo e oggi esistono delle tecniche che permettono di risolverli in tempo ragionevole nelle applicazioni più comuni. Tipicamente, poi, si cerca di ricondurre la soluzione di un problema di programmazione intera NP-*hard* a quella di uno più semplice (tecniche di rilassamento) o già studiato (euristiche). In questo modo è possibile affrontare una grande varietà di problemi. Per questa classe di problemi esiste un

algoritmo di risoluzione molto importante, chiamato *branch and bound* che è anche quello utilizzato da GLPK per risolvere problemi di programmazione misto-intera (in generale problemi che presentano al loro interno sia variabili continue che variabili intere). Questo algoritmo deve la sua importanza al fatto che è un metodo di risoluzione esatto. Ciò significa che permette di trovare la miglior soluzione ammissibile, qualora questa esista, che risolve il problema studiato. Non entriamo nei dettagli del *branch and bound* in quanto nell'ambito di questa tesi ci basta sapere solamente che questo algoritmo è implementato in GLPK.

Capitolo 4

Identificazione di una rete P/T

In questo capitolo viene presentata la procedura di identificazione di una rete P/T tramite programmazione lineare proposta da Cabasino *et al.* in [13].

Dopo aver introdotto alcune nozioni preliminari viene illustrata la procedura di base, successivamente vengono introdotte alcune varianti ed estensioni alla procedura standard.

4.1 Nozioni preliminari

In questa sezione si definisce una classe speciale di insiemi di vincoli lineari (*Constraint Set*, che d'ora in poi indicherò con l'acronimo CS), e si dimostra un'importante proprietà di tale classe, che sarà utile nella soluzione del problema di identificazione.

Definizione 4.1.1. *Dati $A \in \mathbb{R}^{m\times n}$ e $\vec{b} \in \mathbb{R}^m$, consideriamo l'insieme dei vincoli lineari:*

$$\mathcal{C}(A, \vec{b}) = \{\vec{x} \in \mathbb{R}^n \mid A \cdot \vec{x} \geq \vec{b}\}.$$

L'insieme $\mathcal{C}(A, \vec{b})$ è chiamato:

- Ideale: *se $\vec{x} \in \mathcal{C}(A, \vec{b})$ implica che $\alpha\vec{x} \in \mathcal{C}(A, \vec{b})$ per ogni $\alpha \in \mathbb{N}^+ = \{1, 2, ...\}$.*

- Razionale: *se $A \in \mathbb{Q}^{m \times n}$ e $\vec{b} \in \mathbb{Q}^m$, cioè se gli elementi della matrice A e del vettore $\vec{b}$ sono razionali.* ■

Il seguente risultato fornisce una semplice caratterizzazione dei CS ideali.

Proposizione 4.1.1. *Un insieme di vincoli lineari $\mathcal{C}(A, \vec{b})$ è ideale se $\vec{b} \geq 0$.*

Dimostrazione. *Poiché $A \cdot \vec{x} \geq \vec{b} \geq 0$ allora per ogni $\alpha \in \mathbb{N}^+$ risulta $A(\alpha\vec{x}) \geq A \cdot \vec{x} \geq \vec{b}$, quindi $\mathcal{C}(A, \vec{b})$ è ideale.* □

Proposizione 4.1.2. *Se un CS è ideale e razionale, allora ha una soluzione ammissibile se e solo se ha una soluzione ammissibile intera.*

Dimostrazione. *La parte* se *è banale.*

Per dimostrare la parte solo se *si ragiona come segue. Se esiste una soluzione esiste quindi una soluzione di base $\vec{x}_B$ tale che*

$$\vec{x}_B = A_B^{-1} \cdot \vec{b},$$

dove A_B è ottenuto da A selezionando un insieme di colonne di base. Se il CS è razionale gli elementi di A_B e $\vec{b}$ sono razionali, quindi anche gli elementi di A_B^{-1} e di $\vec{x}_B$ sono razionali. Se il CS è ideale è solamente necessario moltiplicare il vettore razionale $\vec{x}_B$ per un opportuno intero positivo per ottenere una soluzione intera. □

4.2 Procedura di base

In questa sezione viene illustrata la procedura di identificazione standard di una rete di Petri proposta in [13].

Il problema considerato in questa tesi può essere formalmente enunciato nel modo seguente.

Problema 4.2.1. *Supponiamo che $\mathcal{L} \subset T^*$ sia un linguaggio finito e chiuso per prefisso*[1] *e*

$$k \geq \max_{\sigma \in \mathcal{L}} |\sigma|$$

[1] Si dice che un linguaggio $\mathcal{L}$ è chiuso per prefisso se per ogni stringa (o parola) $\sigma \in \mathcal{L}$ tutti i prefissi di σ sono in $\mathcal{L}$. Per esempio, se $\sigma = t_1t_2t_3$ appartiene ad $\mathcal{L}$, allora anche t_1t_2 e t_1 appartengono ad $\mathcal{L}$.

sia un intero maggiore o uguale alla lunghezza della stringa più lunga di $\mathcal{L}$. Scelto un insieme di posti P di cardinalità[2] *m, si vuole identificare la struttura di una rete $N = (P, T, Pre, Post)$ e una marcatura iniziale M_0 tale che*

$$L_k(N, M_0) = \mathcal{L}.$$

Le incognite da determinare sono gli elementi delle due matrici $Pre, Post \in \mathbb{N}^{m \times n}$ e gli elementi del vettore $M_0 \in \mathbb{N}^m$. ■

Il problema enunciato consiste quindi nell'identificare un sistema di rete di Petri $M_0, Post$ e Pre con m posti (che caratterizzeremo nella seguente sottosezione) che generi il linguaggio $\mathcal{L}$ osservato, costituito da stringhe di lunghezza minore o uguale a k.

4.2.1 Il sistema di rete $\mathcal{D}$-canonico

Associati ad un problema di identificazione vi sono i due insiemi definiti nel seguito.

Definizione 4.2.1. *Supponiamo che $\mathcal{L} \subset T^*$ sia un linguaggio finito e chiuso per prefisso e supponiamo che $k \in \mathbb{N}$ sia definito come nel Problema 4.2.1.*

*Si definiscono l'*insieme delle condizioni di abilitazione

$$\mathcal{E} = \{(\sigma, t) \mid \sigma \in \mathcal{L}, |\sigma| < k, \sigma t \in \mathcal{L}\} \subset T^* \times T \tag{4.1}$$

*e l'*insieme delle condizioni di disabilitazione

$$\mathcal{D} = \{(\sigma, t) \mid \sigma \in \mathcal{L}, |\sigma| < k, \sigma t \notin \mathcal{L}\} \subset T^* \times T. \tag{4.2}$$

■

Quindi, ciascun elemento di $\mathcal{E}$ è costituito da una sequenza σ appartenente al linguaggio e di lunghezza minore di k, seguita da una transizione t tale per cui anche la sequenza σt appartiene ad $\mathcal{L}$.

In generale si possono indicare due semplici regole meccaniche per la costruzione dell'insieme $\mathcal{E}$ a partire da $\mathcal{L}$:

[2] La cardinalità di un insieme indica il numero di elementi in esso contenuti.

1. Il generico elemento t di $\mathcal{L}$ di lunghezza unitaria diventa l'elemento (ε, t) di $\mathcal{E}$.

2. Il generico elemento di $\mathcal{L}$ di lunghezza maggiore di 1 diventa un elemento di $\mathcal{E}$ mettendo una virgola prima dell'ultima transizione e racchiudendo il tutto tra parentesi tonde.

Invece, ciascun elemento di $\mathcal{D}$ è costituito da una sequenza σ appartenente al linguaggio e di lunghezza minore di k, seguita da una transizione t tale per cui la sequenza σt non appartiene ad $\mathcal{L}$.

Esempio 4.2.1. *Supponiamo che*

$$\mathcal{L} = \{\varepsilon, t_1, t_1t_1, t_1t_2, t_1t_1t_2, t_1t_2t_1\}$$

e $k = 3$.

Allora gli insiemi delle condizioni di abilitazione e di disabilitazione risultano rispettivamente:

$$\mathcal{E} = \{(\varepsilon, t_1), (t_1, t_1), (t_1, t_2), (t_1t_1, t_2), (t_1t_2, t_1)\}$$

e

$$\mathcal{D} = \{(\varepsilon, t_2), (t_1t_1, t_1), (t_1t_2, t_2)\}.$$

Come appare evidente, risulta immediato passare da $\mathcal{L}$ ad $\mathcal{E}$, un pò meno passare da $\mathcal{L}$ a $\mathcal{D}$. □

Ho sviluppato un programma, facente parte del pacchetto software illustrato nel capitolo 6 che implementa la procedura di identificazione, che consente di calcolare automaticamente gli insiemi $\mathcal{E}$ e $\mathcal{D}$ a partire dal linguaggio costituito dalle stringhe di lunghezza massima.

Chiaramente, una soluzione al Problema 4.2.1 è una rete $\langle N, M_0 \rangle$ tale che:

- Per ogni $(\sigma, t) \in \mathcal{E}$ la transizione t è abilitata dopo lo scatto di σ, cioè $M_0[\sigma\rangle M_\sigma[t\rangle$.
- Per ogni $(\sigma, t) \in \mathcal{D}$ la transizione t è disabilitata dopo lo scatto di σ, cioè $M_0[\sigma\rangle M_\sigma \neg[t\rangle$.

È possibile caratterizzare il numero di posti richiesto per risolvere il problema di identificazione.

Definizione 4.2.2. *Supponiamo che $\mathcal{L}$ sia un linguaggio finito e chiuso per prefisso sull'alfabeto T, le cui parole hanno lunghezza minore o uguale a k. Dato l'insieme delle condizioni di disabilitazione (4.2) supponiamo che $m_{\mathcal{D}} = |\mathcal{D}|$.*

Si dice che un sistema di rete di Petri $\langle N, M_0\rangle$, con insieme di posti P e $L_k(N, M_0) = \mathcal{L}$, è $\mathcal{D}$-canonico *se*

1. $|P| = m_{\mathcal{D}}$.
2. *Esiste una corrispondenza biunivoca $h : \mathcal{D} \to P$ tale che*

$$M_0(p) + C(p, \cdot) \cdot \vec{\sigma} < Pre(p, t),$$

cioè il posto $p = h(\sigma, t)$ disabilita t dopo σ. ■

In poche parole, un sistema di rete $\langle N, M_0\rangle$ è *$\mathcal{D}$-canonico* se un differente posto è associato ad ogni elemento nell'insieme dei vincoli di disabilitazione $\mathcal{D}$.

Proposizione 4.2.1. *Supponiamo che $\mathcal{L}$ sia un linguaggio finito e chiuso per prefisso sull'alfabeto T, le cui parole hanno lunghezza minore o uguale a k. Se esiste un sistema di rete $\langle \tilde{N}, \tilde{M}_0\rangle$ tale che $L_k(\tilde{N}, \tilde{M}_0) = \mathcal{L}$, allora esiste un sistema di rete $\langle N, M_0\rangle$ che è $\mathcal{D}$-canonico.*

Dimostrazione. *Supponiamo che*

$$P_{(\sigma,t)} = \{p \in \tilde{P} \mid \tilde{M}_0(p) + \tilde{C}(p, \cdot) \cdot \vec{\sigma} < \tilde{Pre}(p, t)\}$$

sia l'insieme di tutti i posti di $\tilde{N}$ che disabilitano la transizione t dopo che la sequenza σ è scattata (qui $\tilde{C}$ è la matrice d'incidenza di $\tilde{N}$). Per ogni coppia $(\sigma, t) \in \mathcal{D}$ supponiamo che $h(\sigma, t)$ sia un posto arbitrario selezionato da $P_{(\sigma,t)}$; supponiamo che P sia l'insieme di posti selezionati e $m = |P|$.

Due differenti casi possono verificarsi.

Caso 1: *$\underline{m = m_{\mathcal{D}}}$, cioè un differente posto è stato selezionato da qualsiasi insieme $P_{(\sigma,t)}$. In tal caso si definisce N come la rete ottenuta da $\tilde{N}$ rimuovendo tutti i posti che non sono in P (se ce ne sono) e assumendo M_0 come la restrizione di $\tilde{M}_0$ ai posti in P. Si richiede che $L_k(N, M_0) = \mathcal{L}$. Infatti, poiché sono stati rimossi posti da $\langle \tilde{N}, \tilde{M}_0\rangle$ allora $L(\tilde{N}, \tilde{M}_0) \subseteq L(N, M_0)$. Dall'altro lato, per costruzione si sa che per ogni parola wt di lunghezza minore o uguale a k risulta*

$$wt \notin L(\tilde{N}, \tilde{M}_0) \quad \Longrightarrow \quad wt \notin L(N, M_0),$$

quindi $L_k(N, M_0) = L_k(\tilde{N}, \tilde{M}_0) = \mathcal{L}$. Per costruzione, un differente posto in P è associato a qualsiasi coppia $(\sigma, t) \in \mathcal{D}$, a dimostrazione che $\langle N, M_0 \rangle$ è $\mathcal{D}$-canonica.

Caso 2: *$m < m_{\mathcal{D}}$, cioè qualche posto $p \in P$ è stato selezionato da $P_{(\sigma,t)}$, $P_{(\sigma',t')}$, $P_{(\sigma'',t'')}$,..., per due o più differenti coppie (σ, t),(σ', t'), (σ'', t''),... in $\mathcal{D}$. In tal caso vengono aggiunti alla rete, senza cambiare il suo linguaggio, posti addizionali p', p''... tali che $Pre(p, \cdot) = Pre(p', \cdot) = Pre(p'', \cdot) = \cdots, Post(p, \cdot) = Post(p', \cdot) = Post(p'', \cdot) = \cdots$ e $M_0(p) = M_0(p') = M_0(p'') = \cdots$, perciò ottenendo una rete con $m_{\mathcal{D}}$ posti. Si ridefiniscono $h(\sigma', t') = p'$ e $h(\sigma'', t'') = p'', \ldots$. La funzione h è ora biiettiva[3] e il sistema di rete risultante è $\mathcal{D}$-canonico.* □

4.2.2 Il *Constraint Set* (CS)

Il seguente teorema fornisce una caratterizzazione algebrico-lineare delle reti posto/transizione con $m_{\mathcal{D}}$ posti ed n transizioni tali che $L_k(N, M_0) = \mathcal{L}$.

Teorema 4.2.1. *Supponiamo di considerare un linguaggio finito e chiuso per prefisso sull'alfabeto T le cui parole hanno lunghezza minore o uguale a k e supponiamo che $\mathcal{E}$ e $\mathcal{D}$ siano i corrispondenti insiemi delle condizioni di abilitazione e di disabilitazione. Supponiamo che*

$\mathcal{N}(\mathcal{E}, \mathcal{D}) \triangleq$

$$\begin{cases} M_0 + Post \cdot \vec{\sigma} - Pre \cdot (\vec{\sigma} + \vec{t}) \geq \vec{0} & \forall (\sigma, t) \in \mathcal{E} \\ M_0(p_{(\sigma,t)}) + Post(p_{(\sigma,t)}, \cdot) \cdot \vec{\sigma} - Pre(p_{(\sigma,t)}, \cdot) \cdot (\vec{\sigma} + \vec{t}) \leq -1 & \forall (\sigma, t) \in \mathcal{D} \\ M_0 \in \mathbb{R}_{\geq 0}^{m_{\mathcal{D}}} \\ Post, Pre \in \mathbb{R}_{\geq 0}^{m_{\mathcal{D}} \times n} \end{cases} \tag{4.3}$$

Consideriamo un sistema di rete $\langle N, M_0 \rangle$ con $N = (P, T, Pre, Post)$. Il sistema $\langle N, M_0 \rangle$ è una soluzione $\mathcal{D}$-canonica del problema di identificazione 4.2.1 se e solo se $M_0, Post$ e Pre sono soluzioni intere del CS (4.3).

Dimostrazione. *Per prima cosa si dimostra che qualsiasi soluzione intera $\langle N, M_0 \rangle$ del CS (4.3) è una soluzione del Problema 4.2.1.*

[3] Una funzione $f : A \leftrightarrow B$ si dice biiettiva o biunivoca se ogni elemento di A è in relazione con uno e un solo elemento di B.

- *Qualunque vincolo $M_0 + Post \cdot \vec{\sigma} - Pre \cdot (\vec{\sigma} + \vec{t}) \geq \vec{0}$ può essere riscritto come $M_\sigma = M_0 + (Post - Pre) \cdot \vec{\sigma} \geq Pre(\cdot, t)$ o equivalentemente $M_\sigma \geq Pre(\cdot, t)$ dove $M_0[\sigma\rangle M_\sigma$. Questo dimostra che la transizione t è abilitata su $\langle N, M_0\rangle$ dalla marcatura M_σ e per induzione sulla lunghezza di σ (poiché il linguaggio $\mathcal{L}$ è chiuso per prefisso) si conclude che $\sigma t \in \mathcal{L}$.*

- *Si assume che la sequenza σ possa scattare sulla rete e $M_0[\sigma\rangle M_\sigma$. Se per almeno un posto p nella rete risulta $M_0(p)+Post(p,\cdot)\cdot\vec{\sigma}-Pre(p,\cdot)\cdot(\vec{\sigma}+\vec{t}) \leq -1$, allora $M_\sigma = M_0 + (Post - Pre) \cdot \vec{\sigma} \ngeq Pre(\cdot, t)$ o equivalentemente $M_\sigma \ngeq Pre(\cdot, t)$. Questo dimostra che la transizione t non è abilitata su $\langle N, M_0\rangle$ dalla marcatura M_σ e si conclude che $\sigma t \notin \mathcal{L}$.*

Poiché la rete $\langle N, M_0\rangle$ soddisfa tutti i vincoli di abilitazione e disabilitazione, $L_k(N, M_0) = \mathcal{L}$.

Adesso si dimostra che qualsiasi soluzione del CS (4.3) è $\mathcal{D}$-canonica. Infatti, la corrispondenza $h(\sigma, t) = p_{(\sigma,t)}$ per ogni coppia $(\sigma, t) \in \mathcal{D}$ è biunivoca.

Ora si dimostra che qualunque sistema di rete $\langle N, M_0\rangle$ $\mathcal{D}$-canonico con $L_k(N, M_0) = \mathcal{L}$ è una soluzione del CS (4.3). Infatti, supponiamo che $h : \mathcal{D} \to P$ sia la funzione biiettiva del sistema di rete. Se si definisce $p_{(\sigma,t)} = h(\sigma, t)$ per ogni $(\sigma, t) \in \mathcal{D}$, allora tutte le equazioni nel CS (4.3) sono soddisfatte. □

Proposizione 4.2.2. *Il CS (4.3) lineare è ideale e razionale.*

Dimostrazione. *Per prima cosa si osserva che il CS (4.3) lineare può essere riscritto come un insieme di disequazioni lineari della forma $A \cdot \vec{x} \geq \vec{b}$ nel modo seguente. Supponiamo di indicare come pre_i e $post_i$ l'i-$esima$ riga delle matrici Pre e $Post$, rispettivamente, per $i = 1, ..., m_\mathcal{D}$. Per ogni $(\sigma, t) \in \mathcal{E}$ la prima disequazione matriciale nel (4.3) può essere riscritta come il seguente insieme di $m_\mathcal{D}$ disequazioni scalari:*

$$\begin{bmatrix} 1 & \vec{\sigma}^T & -\vec{\sigma}^T - \vec{t}^T \end{bmatrix} \cdot \begin{bmatrix} M_{0,i} \\ post_i^T \\ pre_i^T \end{bmatrix} \geq 0,$$

dove $i = 1, ..., m_\mathcal{D}$. Analogamente, per ogni $(\sigma, t) \in \mathcal{D}$ la seconda disequazione scalare nel (4.3) può essere riscritta come:

$$\left[\begin{array}{ccc} -1 & -\vec{\sigma}^T & \vec{\sigma}^T + \vec{t}^T \end{array} \right] \cdot \left[\begin{array}{c} M_{0,i} \\ post_i^T \\ pre_i^T \end{array} \right] \geq 1.$$

Questo definisce la matrice A e il vettore $\vec{b}$. Poiché A e $\vec{b}$ hanno elementi interi, il CS (4.3) è razionale. Poiché $\vec{b} \geq 0$, per la Proposizione 4.1.1 il CS (4.3) è ideale.

□

Il numero totale di disequazioni scalari associate ad $\mathcal{E}$ e a $\mathcal{D}$ è il numero di righe di A. Il numero di variabili incognite del problema o colonne di A è pari a $m_{\mathcal{D}} + 2(m_{\mathcal{D}} \cdot n)$. La formula precedente rappresenta la somma del numero di elementi di M_0 più la somma del numero di elementi di $Post$ e Pre.

Quindi, la dimensione della matrice dei coefficienti A è un indicatore della complessità del problema di programmazione e del sistema di rete da identificare.

Nei precedenti lavori [3] e [12] sulla programmazione intera è stato calcolato che il numero totale di incognite nel caso peggiore è pari a $m + 2(m \cdot n) + m \cdot n^k$, dove m è il numero di posti, n è il numero di transizioni e k è la lunghezza della stringa più lunga del linguaggio, cioè la complessità computazionale cresce in maniera esponenziale rispetto a k. Quando andremo a testare la procedura di identificazione sviluppata con Matlab ci renderemo conto dei vantaggi che offre la programmazione lineare rispetto a quella intera. In particolare faremo un confronto tra i risultati delle simulazioni numeriche condotte su una rete molto comune, il processo emittente-ricevente, in un caso e nell'altro.

4.2.3 Il *Constraint Set* (CS) semplificato

In questa sottosezione viene mostrato come sia possibile ridurre l'insieme dei vincoli e dei posti della rete.

Ragioniamo sulle dimensioni del CS (4.3) e quindi sulla complessità del problema.

Il numero dei vincoli matriciali (o disequazioni matriciali) corrispondenti all'insieme $\mathcal{E}$ a prima vista sembrerebbe sempre pari alla cardinalità di $\mathcal{E}$ perché per ogni elemento di $\mathcal{E}$ posso scrivere un vincolo matriciale.

Però potrebbe capitare che in $\mathcal{E}$ ci siano elementi (σ, t) che abbiano lo stesso vettore di scatto $\vec{\sigma}$ e contemporaneamente lo stesso $\vec{t}$, dove $\vec{t}$ è il vettore colonna di lunghezza n avente $n-1$ componenti nulle e la componente i-esima, dove i è l'indice della transizione t, pari a 1. Per esempio, se n=5: per $(\sigma, t) = (t_2 t_3, t_1)$ $\vec{\sigma} = [0\ 1\ 1\ 0\ 0]^T$ e $\vec{t} = [1\ 0\ 0\ 0\ 0]^T$, per $(\sigma, t) = (t_3 t_2, t_1)$ $\vec{\sigma} = [0\ 1\ 1\ 0\ 0]^T$ e $\vec{t} = [1\ 0\ 0\ 0\ 0]^T$. In questo caso abbiamo vincoli matriciali uguali. Infatti, se riprendiamo il CS (4.3) e consideriamo i vincoli matriciali associati ad $\mathcal{E}$, notiamo che M_0 è moltiplicato per una matrice identità, $Post$ è moltiplicata per $\vec{\sigma}$ e i $\vec{\sigma}$ sono uguali, Pre è moltiplicata per $\vec{\sigma} + \vec{t}$ e i $\vec{\sigma} + \vec{t}$ sono uguali. In questo modo otteniamo righe di A uguali perciò ridondanti. Quindi si possono eliminare i vincoli matriciali uguali tranne uno senza alterare la correttezza del problema e col vantaggio di diminuire le dimensioni della matrice dei coefficienti A e di conseguenza la complessità del problema e della rete. Come detto, tra i vincoli uguali uno non dev'essere eliminato: infatti i vincoli associati a tutti gli altri elementi di $\mathcal{E}$ sono sicuramente diversi da questo. Risulta allora che il numero dei vincoli matriciali associati ad $\mathcal{E}$ è pari al numero di differenti coppie $(\vec{\sigma}, \vec{\sigma} + \vec{t})$. Il numero dei vincoli scalari è pari al numero dei vincoli matriciali per il numero di posti $m_{\mathcal{D}}$ della rete: naturalmente è stato ridotto anch'esso dopo la riduzione dei vincoli matriciali.

Da notare che non è sufficiente avere vettori di scatto $\vec{\sigma}$ uguali per l'uguaglianza dei vincoli matriciali, ma occorre che anche i vettori di scatto $\vec{\sigma} + \vec{t}$ siano uguali. Per esempio $(t_2 t_3, t_1)$ e $(t_3 t_2, t_4)$ hanno il medesimo vettore di scatto $\vec{\sigma}$ ($[0\ 1\ 1\ 0\ 0]^T$), ma poiché la transizione abilitata dopo lo scatto delle rispettive σ è diversa (in un caso t_1 e nell'altro t_4) i rispettivi $\vec{\sigma} + \vec{t}$ sono diversi e quindi lo sono anche i vincoli corrispondenti.

Per quanto riguarda l'insieme $\mathcal{D}$, per la $\mathcal{D}$-canonicità un differente posto, e quindi un differente vincolo scalare (o disequazione scalare) è associato ad ogni suo elemento. Per cui è evidente che il numero di vincoli scalari associato a $\mathcal{D}$ è pari alla sua cardinalità ($m_{\mathcal{D}}$). Tuttavia, anche per $\mathcal{D}$ si può ridurre il numero di vincoli, e quindi di posti, se sono presenti coppie $(\vec{\sigma}, \vec{\sigma} + \vec{t})$ uguali. Cioè, è possibile ridurre il numero di posti della rete, e quindi la complessità computazionale del problema, individuando l'esistenza di posti ridondanti che si possono eliminare, come discusso nel seguito. Vorrei precisare che i ragionamenti che seguono prescindono dalle tecniche di riduzione dei posti che verranno presentate nel prossimo capitolo.

Per problemi complessi, a volte addiritura anche per problemi semplici, la cardinalità di $\mathcal{D}$ può essere un numero molto grande, nel caso peggiore pari a $|T|^k$ [3]. Di conseguenza il numero di vincoli del problema di programmazione può

crescere in maniera spropositata al punto che il software utilizzato per implementare la procedura (nel nostro caso Matlab) può avere difficoltà nello scriverli e la complessità computazionale aumentata rende il problema difficilmente gestibile (affronteremo il problema in maniera diretta quando illustrerò degli esempi a cui applicare la procedura di identificazione).

Ma, riflettiamo su un fatto. Cosa significa che più elementi hanno lo stesso vettore di scatto $\vec{\sigma}$? Vuol dire che lo scatto delle relative sequenze σ porta alla stessa marcatura, ce lo dice l'equazione di stato, infatti ricordiamo che per essa risulta $M = M_0 + C \cdot \vec{\sigma}$. Inoltre, se gli stessi elementi hanno anche lo stesso $\vec{\sigma} + \vec{t}$ allora, una volta raggiunta la marcatura M dopo lo scatto di σ, verrà disabilitata la stessa transizione t. Quindi, è ridondante assegnare un posto a ciascuno di questi elementi, ma è sufficiente utilizzare un unico posto che, dopo lo scatto della sequenza σ e il raggiungimento della marcatura M, disabiliti la transizione t.

In altre parole, se esistono in $\mathcal{D}$ elementi aventi la stessa coppia $(\vec{\sigma}, \vec{\sigma} + \vec{t})$ questo viene ridotto. Infatti, vincoli corrispondenti a tali elementi sono uguali e, dunque, possono essere eliminati tranne uno, così come è stato fatto per elementi simili di $\mathcal{E}$. Di conseguenza, anche il numero di vincoli scalari associati ad $\mathcal{E}$ verrebbe ulteriormente ridotto, dato che il numero di posti ridefinito è inferiore a quello caratterizzato nella sezione 4.2. È evidente che la complessità del problema diminuisce.

In virtù della semplificazione dei vincoli ridondanti, gli insiemi $\mathcal{E}$ e $\mathcal{D}$ possono essere ridefiniti in questo modo:

$$\mathcal{E}' = \{(\vec{\sigma}, t) \mid \exists \sigma \in \mathcal{L}, |\sigma| < k, \sigma t \in \mathcal{L}\} \subset \mathbb{N}^n \times T$$

e

$$\mathcal{D}' = \{(\vec{\sigma}, t) \mid \exists \sigma \in \mathcal{L}, |\sigma| < k, \sigma t \notin \mathcal{L}\} \subset \mathbb{N}^n \times T.$$

Quindi, ciascun elemento di $\mathcal{E}$ è costituito da un vettore di scatto $\vec{\sigma}$, tale per cui esiste una corrispondente sequenza σ appartiene al linguaggio e di lunghezza minore di k, seguito da una transizione t tale per cui anche la sequenza σt appartiene ad $\mathcal{L}$.

Invece, ciascun elemento di $\mathcal{D}$ è costituito da un vettore di scatto $\vec{\sigma}$, tale per cui esiste una corrispondente sequenza σ appartiene al linguaggio e di lunghezza minore di k, seguito da una transizione t tale per cui la sequenza σt non appartiene ad $\mathcal{L}$.

4.2.4 Risoluzione del problema di programmazione lineare

Il seguente teorema fornisce una pratica ed efficiente procedura per risolvere il problema di identificazione.

Teorema 4.2.2. *Il problema di identificazione 4.2.1 ammette una soluzione se e solo se il CS (4.3) (lineare) è ammissibile.*

Dimostrazione. (se) *Per la Proposizione 4.2.2 il CS (4.3) è ideale e razionale quindi per la Proposizione 4.1.2 se ha soluzioni, allora ha anche soluzioni ammissibili intere. Tuttavia per il Teorema 4.2.1 questo implica anche che tali soluzioni intere siano anche soluzioni del problema di identificazione 4.2.1.*

(solo se) *Se esiste una soluzione del Problema 4.2.1, per la Proposizione 4.2.1 esiste anche un sistema di rete che è $\mathcal{D}$-canonico. Ma tutti i sistemi $\mathcal{D}$-canonici sono soluzioni del CS (4.3) per il Teorema 4.2.1, perciò il CS (4.3) è ammissibile.*

□

Il teorema precedente, in altre parole, afferma che un sistema di rete $\langle N, M_0 \rangle$ è una soluzione del problema di identificazione 4.2.1 se e solo se esso soddisfa l'insieme dei vincoli algebrici lineari definito nel Teorema 4.2.1.

Infine notiamo che, una volta che una soluzione del CS (4.3) è ricavata, se questa soluzione è razionale si può sempre trovare una soluzione intera semplicemente moltiplicando $M_0, Post$ e Pre per un opportuno intero non negativo α.

In generale la soluzione del CS (4.3) non è unica per cui esiste più di un sistema di rete di Petri $\langle N, M_0 \rangle$ tale che $L_k(N, M_0) = \mathcal{L}$. Per selezionare uno tra questi sistemi di rete si sceglie un dato indice di prestazione o funzione obiettivo e risolvendo un opportuno problema di programmazione lineare (che d'ora in poi indicherò con l'acronimo PPL) si determina un sistema di rete di Petri che minimizza la funzione obiettivo considerata (chiaramente, anche in questo caso la soluzione può non essere unica).

In particolare, se $f(M_0, Post, Pre)$ è la funzione obiettivo considerata, un problema di identificazione può essere formalmente enunciato nel modo seguente.

Problema 4.2.2. *Consideriamo il problema di identificazione 4.2.1 e supponiamo che $f(M_0, Post, Pre)$ sia una data funzione obiettivo. La soluzione del problema di identificazione 4.2.1 che minimizza $f(M_0, Post, Pre)$ può essere calcolata risolvendo il seguente PPL:*

$$\begin{cases} minimizza & f(M_0, Post, Pre) \\ tale\ che & \mathcal{N} = (\mathcal{E}, \mathcal{D}). \end{cases}$$

■

Di particolare interesse sono quelle funzioni obiettivo che sono lineari nelle incognite, in modo che il problema da risolvere sia un PPL. Come già detto nel capitolo 1, ho utilizzato un risolutore commerciale (GLPK) per risolvere questo tipo di problemi. Come esempio di funzione obiettivo lineare, si è assunto di voler determinare un sistema di rete di Petri che minimizza la somma pesata dei gettoni nella marcatura iniziale e dei pesi degli archi.

Il caso generale è:

$$f(M_0, Post, Pre) = \sum_{i=1}^{m} \left(a_i M_0(p_i) + \left(\sum_{j=1}^{n} b_{i,j} Pre(p_i, t_j) + c_{i,j} Post(p_i, t_j) \right) \right), \tag{4.4}$$

dove a_i, $b_{i,j}$ e $c_{i,j}$ sono coefficienti dati. Una tipica scelta, a cui rimaniamo fedeli nel prosieguo della tesi, è quella di scegliere tutti i coefficienti uguali a 1. In questo caso la (4.4) può essere riscritta:

$$f(M_0, Post, Pre) = \vec{1}_m^T \cdot M_0 + \vec{1}_m^T \cdot Pre \cdot \vec{1}_n + \vec{1}_m^T \cdot Post \cdot \vec{1}_n. \tag{4.5}$$

4.2.5 Esempi

In questa sottosezione vengono illustrati due esempi di applicazione della procedura presentata nelle sottosezioni precedenti.

Esempio 4.2.2. *Riprendiamo l'Esempio 4.2.1 in cui* $\mathcal{L} = \{\varepsilon, t_1, t_1t_1, t_1t_2, t_1t_1t_2, t_1t_2t_1\}$ *e* $k = 3$*. Assumiamo di voler determinare il sistema di rete di Petri che minimizza la somma dei gettoni iniziali e di tutti gli archi in modo tale che* $L_3(N, M_0) = \mathcal{L}$*. Questo richiede la soluzione di un PPL della forma (4.3) dove*

$$\mathcal{E} = \{(\varepsilon, t_1), (t_1, t_1), (t_1, t_2), (t_1t_1, t_2), (t_1t_2, t_1)\}$$

e

$$\mathcal{D} = \{(\varepsilon, t_2), (t_1t_1, t_1), (t_1t_2, t_2)\}.$$

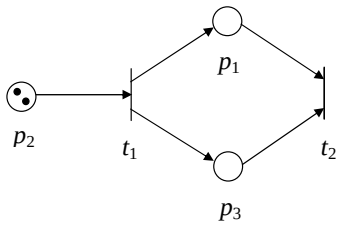

Figura 4.1: Il sistema di rete di Petri dell'Esempio 4.2.2.

Poiché $|\mathcal{D}| = 3$*, la rete da identificare ha* $m_{\mathcal{D}} = 3$ *posti e a* $\mathcal{D}$ *corrispondono 3 vincoli scalari. I vettori di scatto dei 3 elementi* $(\sigma, t) \in \mathcal{D}$ *sono rispettivamente:* $\vec{\sigma}_1 = [0\ 0]^T$, $\vec{\sigma}_2 = [2\ 0]^T$, $\vec{\sigma}_3 = [1\ 1]^T$, $\vec{\sigma}_1 + \vec{t}_1 = [0\ 1]^T$, $\vec{\sigma}_2 + \vec{t}_2 = [3\ 0]^T$ *e* $\vec{\sigma}_3 + \vec{t}_3 = [1\ 2]^T$.

Per quanto riguarda l'insieme $\mathcal{E}$*, i vettori di scatto sono rispettivamente:* $\vec{\sigma}_1 = [0\ 0]^T$, $\vec{\sigma}_2 = [1\ 0]^T$, $\vec{\sigma}_3 = [1\ 0]^T$, $\vec{\sigma}_4 = [2\ 0]^T$, $\vec{\sigma}_5 = [1\ 1]^T$, $\vec{\sigma}_1 + \vec{t}_1 = [1\ 0]^T$, $\vec{\sigma}_2 + \vec{t}_2 = [2\ 0]^T$, $\vec{\sigma}_3 + \vec{t}_3 = [1\ 1]^T$, $\vec{\sigma}_4 + \vec{t}_4 = [2\ 1]^T$ *e* $\vec{\sigma}_5 + \vec{t}_5 = [2\ 1]^T$*. Poiché non sono presenti coppie* $(\vec{\sigma}, \vec{\sigma} + \vec{t})$ *uguali, abbiamo un numero di vincoli matriciali pari a* $|\mathcal{E}| = 5$ *e un numero di vincoli scalari pari a* $|\mathcal{E}| \cdot m_{\mathcal{D}} = 5 \cdot 3 = 15$.

Il numero di vincoli scalari o disequazioni del CS (4.3) è quindi pari a $15+3 = 18$*. Il numero di incognite è pari a* $m_{\mathcal{D}}+2(m_{\mathcal{D}} \cdot n) = 3+2(3 \cdot 2) = 15$*, cioè la matrice dei coefficienti ha 18 righe e 15 colonne.*

La procedura di identificazione, da me implementata in Matlab e che verrà illustrata nel capitolo 6, identifica un sistema di rete con

$$M_0 = \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \quad e \quad Pre = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix},$$

cioè il sistema di rete in Figura 4.1. □

Ai fini della tesi, comunque, non è necessario disegnare la rete, ma è sufficiente ricavare la soluzione in termini matriciali.

Esempio 4.2.3. *Consideriamo una variante dell'esempio precedente e supponiamo che* $\mathcal{L} = \{\varepsilon, t_1, t_2, t_1t_2, t_2t_1, t_1t_2t_1, t_2t_1t_1\}$ *e* $k = 3$*. Gli insiemi delle condizioni di abilitazione e di disabilitazione sono rispettivamente*

$$\mathcal{E} = \{(\varepsilon, t_1), (\varepsilon, t_2), (t_1, t_2), (t_2, t_1), (t_1t_2, t_1), (t_2t_1, t_1)\}$$

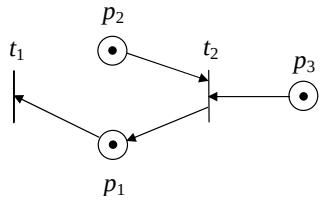

Figura 4.2: Il sistema di rete di Petri dell'Esempio 4.2.3.

e

$$\mathcal{D} = \{(t_1, t_1), (t_2, t_2), (t_1t_2, t_2), (t_2t_1, t_2)\}.$$

Alla terza e alla quarta sequenza di $\mathcal{D}$, che hanno la stessa coppia $(\vec{\sigma}, \vec{\sigma} + \vec{t})$ (pari a $([1\ 1]^T, [1\ 2]^T)$), assegnamo un unico posto invece che 2, per cui la rete da identificare ha 3 posti invece che 4, e a $\mathcal{D}$ corrispondono 3 vincoli scalari.

In $\mathcal{E}$ il quinto e il sesto elemento hanno la stessa coppia $(\vec{\sigma}, \vec{\sigma} + \vec{t})$, pari a $([1\ 1]^T$, $[2\ 1]^T)$, che porterebbe a scrivere 2 vincoli matriciali uguali. Per cui una di queste si può eliminare in modo da avere 5 vincoli matriciali invece che 6.

Poiché il numero di posti è pari a 3, il numero totale di vincoli scalari è pari a $5 \cdot 3 + 3 = 18$ invece che a 28 e il numero di variabili è pari a $3 + 2(3 \cdot 2) = 15$ invece che a 20. Ne deriva una matrice dei coefficienti di dimensioni 18×15.

Gli insiemi $\mathcal{E}$ e $\mathcal{D}$ ridotti sono:

$$\mathcal{E}' = \{([0\ 0]^T, t_1), ([0\ 0]^T, t_2), ([1\ 0]^T, t_2), ([0\ 1]^T, t_1), ([1\ 1]^T, t_1)\}$$

e

$$\mathcal{D}' = \{([1\ 0]^T, t_1), ([0\ 1]^T, t_2), ([1\ 1]^T, t_2)\}.$$

La procedura di identificazione in Matlab identifica una rete con

$$M_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad e \quad Pre = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix},$$

cioè la rete in Figura 4.2. □

Il vantaggio dell'eliminazione dei vincoli ridondanti risulterà evidente per problemi più complessi come verrà mostrato nel capitolo 7.

4.3 Procedura estesa

In alcuni casi l'informazione disponibile sulla rete da identificare non è limitata al suo linguaggio. Per esempio, può essere noto che la rete abbia una particolare struttura o può essere data qualche informazione parziale sulla marcatura iniziale (in termini di risorse disponibili). In questa sezione viene mostrato come questa informazione addizionale può essere facilmente inserita nella procedura di identificazione descritta precedentemente.

Altre informazioni sulla struttura di rete che richiedono una modifica dei vincoli, e non una semplice aggiunta come verrà fatto in questa sezione, verranno discusse nel capitolo 5.

4.3.1 Vincoli strutturali

P-vettori
Assumiamo che sia noto che alcuni posti della rete appartengano ad una componente conservativa, cioè la somma pesata dei loro gettoni nella componente rimanga costante durante qualsiasi evoluzione. Questo è equivalente a dire che alcuni P-invarianti per la rete sono noti. Più in generale la conoscenza di un qualsiasi P-vettore può essere messa in conto aggiungendo al CS (4.3) un opportuno insieme di vincoli.

- Assumiamo che $\vec{x} \in \mathbb{R}^q$ sia un *P-invariante*. Quindi dobbiamo aggiungere al CS (4.3) il seguente vincolo

$$\vec{x}^T \cdot (Post - Pre) = \vec{0}_n^T$$

che impone $\vec{x}^T \cdot C = \vec{0}_n^T$.

- Assumiamo che $\vec{x} \in \mathbb{R}^q$ sia un *P-vettore crescente*. Quindi dobbiamo aggiungere al CS (4.3) i seguenti vincoli

$$\begin{cases} \vec{x}^T \cdot (Post - Pre) \geq \vec{0}_n^T \\ \vec{x}^T \cdot (Post - Pre) \cdot \vec{1}_n \geq 1. \end{cases}$$

Il primo vincolo impone che $\vec{x}^T \cdot C \geq \vec{0}_n^T$ e il secondo impone che $\vec{x}^T \cdot C \neq \vec{0}_n^T$.

- Assumiamo che $\vec{x} \in \mathbb{R}^q$ sia un *P-vettore decrescente*. Quindi dobbiamo aggiungere al CS (4.3) i seguenti vincoli

$$\begin{cases} \vec{x}^T \cdot (Post - Pre) \leq \vec{0}_n^T \\ \vec{x}^T \cdot (Post - Pre) \cdot \vec{1}_n \leq -1. \end{cases}$$

  Il primo vincolo impone che $\vec{x}^T \cdot C \leq \vec{0}_n^T$ e il secondo impone che $\vec{x}^T \cdot C \neq \vec{0}_n^T$.

Il problema di questa informazione dipende dal fatto che in genere il numero dei posti non è noto a priori come il numero di transizioni, ma dipende da $\mathcal{D}$.

T-vettori
Assumiamo che sia noto che una data sequenza di scatto sia stazionaria, cioè il numero dei gettoni della rete non viene modificato dallo scatto di questa sequenza. Questo è equivalente a dire che alcuni T-invarianti per questa rete sono noti. Più in generale la conoscenza di un qualsiasi T-vettore può essere messa in conto aggiungendo al CS (4.3) un opportuno insieme di vincoli.

- Assumiamo che $\vec{y} \in \mathbb{R}^n$ sia un *T-invariante*. Quindi dobbiamo aggiungere al CS (4.3) il seguente vincolo

$$(Post - Pre) \cdot \vec{y} = \vec{0}_q$$

  che impone $C \cdot \vec{y} = \vec{0}_q$.

- Assumiamo che $\vec{y} \in \mathbb{R}^n$ sia un *T-vettore crescente*. Quindi dobbiamo aggiungere al CS (4.3) i seguenti vincoli

$$\begin{cases} (Post - Pre) \cdot \vec{y} \geq \vec{0}_q \\ \vec{1}_q^T \cdot (Post - Pre) \cdot \vec{y} \geq 1. \end{cases}$$

  Il primo vincolo impone che $C \cdot \vec{y} \geq \vec{0}_q^T$ e il secondo impone che $C \cdot \vec{y} \neq \vec{0}_q^T$.

- Assumiamo che $\vec{y} \in \mathbb{R}^n$ sia un *T-vettore decrescente*. Quindi dobbiamo aggiungere al CS (4.3) i seguenti vincoli

$$\begin{cases} (Post - Pre) \cdot \vec{y} \leq \vec{0}_q \\ \vec{1}_q^T \cdot (Post - Pre) \cdot \vec{y} \leq -1. \end{cases}$$

  Il primo vincolo impone che $C \cdot \vec{y} \leq \vec{0}_q^T$ e il secondo impone che $C \cdot \vec{y} \neq \vec{0}_q^T$.

Classi di reti
Consideriamo i vincoli che dobbiamo aggiungere al CS (4.3) per assicurare che la rete identificata appartenga ad alcune particolari classi di reti definite sotto.

- *Macchina di stato*:
$$\begin{cases} \vec{1}_q^T \cdot Post = 1 \\ \vec{1}_q^T \cdot Pre = 1. \end{cases}$$
- *Grafo marcato*:
$$\begin{cases} Post \cdot \vec{1}_n = 1 \\ Pre \cdot \vec{1}_n = 1. \end{cases}$$
- *Rete ordinaria*:
$$Post, Pre \in \{0,1\}^{q \times n}.$$

Tutti questi risultati derivano immediatamente dalle definizioni di macchina di stato, di grafo marcato e di rete ordinaria.

Non sono stati considerati i casi di rete pura, ristretta e a scelta libera perché i relativi vincoli su $Post$ e Pre non si possono esprimere in maniera lineare e quindi non possono essere aggiunti al CS (4.3).

4.3.2 Vincoli sulla marcatura iniziale

Un tipo di vincolo generale che può essere imposto sulle marcature di una rete di Petri è chiamato GMEC (*Generalized Mutual Exclusion Constraint*) e può essere rappresentato dalla coppia $(\vec{w}, c)$, dove $\vec{w} \in \mathbb{Z}^q$ e $c \in \mathbb{Z}$. Questo vincolo definisce un insieme di marcature legali:

$$\mathcal{M}(\vec{w}, c) = \{M \in \mathbb{N}^q \mid \vec{w}^T \cdot M \leq c\}.$$

Se è noto che $M_0 \in \mathcal{M}(\vec{w}, c)$ allora il vincolo

$$\vec{w}^T \cdot M_0 \leq c$$

deve essere aggiunto al CS (4.3).

Capitolo 5

Riduzione dei posti

5.1 Introduzione

Uno svantaggio della procedura di identificazione delineata nel capitolo precedente consiste nella richiesta che la rete contenga un numero di posti pari a $m_{\mathcal{D}} = |\mathcal{D}|$ sebbene possa esistere una rete equivalente, cioè che genera lo stesso linguaggio, con un numero più piccolo di posti. Abbiamo visto che ciò si verifica nel caso in cui ci siano elementi di $\mathcal{D}$ con la stessa coppia $(\vec{\sigma}, \vec{\sigma} + \vec{t})$. Nonostante sia possibile ottenere una rete più piccola eliminando vincoli ridondanti, esiste comunque un limite alla riduzione dei vincoli effettuata in questo modo. Ricordiamo che nel caso peggiore la cardinalità di $\mathcal{D}$ può essere pari a $|T|^k$.

In [13] sono stati proposti due (tipi di) approcci per superare questi problemi.

Nel primo approccio, che si chiama *pre-riduzione dei posti*, il CS (4.3) è scritto in una forma modificata, usando un numero ridotto di posti. Il numero di posti viene ridotto "a monte", cioè prima di calcolare la soluzione del problema di programmazione lineare (PPL) a partire dal CS. Questa tecnica non è sempre utilizzabile e talvolta, come vedremo, non consente notevoli semplificazioni del problema.

Nel secondo approccio, che si chiama *post-riduzione dei posti*, prima si determina una soluzione del CS (4.3) standard ottenendo una rete con $m_{\mathcal{D}}$ posti o con un numero più piccolo e poi si identificano posti ridondanti che possono essere rimossi senza intaccare la correttezza del risultato. Quindi la tecnica di post-riduzione, come suggerisce il nome, viene applicata "a valle", cioè una volta ricavata una soluzione del PPL. Questa tecnica, a differenza della pre-riduzione, è

sempre applicabile. La sua efficacia verrà valutata nel capitolo 7.

5.2 Pre-riduzione dei posti

5.2.1 Il *Constraint Set* (CS) modificato

Iniziamo con un risultato generale che consente di controllare se la rete ottenuta risolvendo il CS (4.3) ha un numero minimo di posti. Il test proposto richiede di risolvere una serie di CS modificati e questo è il motivo per cui questo risultato viene presentato nella sezione dedicata alla pre-riduzione.

Definizione 5.2.1. *Consideriamo una partizione* $\Pi(\mathcal{D}) = \{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_q\}$ *dell'insieme* $\mathcal{D}$*. Gli insiemi* $\mathcal{D}_i$ *sono chiamati blocchi della partizione* $\Pi(\mathcal{D})$*. Si definisce il seguente CS*

$\mathcal{N}(\mathcal{E}, \Pi(\mathcal{D})) \triangleq$

$$\begin{cases} M_0 + Post \cdot \vec{\sigma} - Pre \cdot (\vec{\sigma} + \vec{t}) \geq \vec{0} & \forall (\sigma, t) \in \mathcal{E} \\ M_0(p_i) + Post(p_i, \cdot) \cdot \vec{\sigma} - Pre(p_i, \cdot) \cdot (\vec{\sigma} + \vec{t}) < 0 & \forall (\sigma, t) \in \mathcal{D}_i, \\ & i = 1, ..., q \\ M_0 \in \mathbb{R}^q_{\geq 0} \\ Post, Pre \in \mathbb{R}^{q \times n}_{\geq 0} \end{cases} \tag{5.1}$$

dove $\mathcal{E}$ *e* $n = |T|$ *hanno l'usuale significato.* ■

La sola differenza tra il CS (5.1) e il CS (4.3) consiste nel fatto che nel primo solo q posti (tanti quanti i blocchi della partizione $\Pi(\mathcal{D})$) vengono utilizzati: il posto p_i $(i = 1, ..., q)$ assicurerà che tutte le condizioni di disabilitazione in $\mathcal{D}_i$ siano imposte. È immediato dimostrare, con lo stesso ragionamento della Proposizione 4.2.2, che il CS (5.1) è razionale e ideale e che qualunque delle sue soluzioni intere è una soluzione al problema di identificazione 4.2.1.

Definizione 5.2.2. *Dato il problema di identificazione 4.2.1, una partizione* $\Pi(\mathcal{D}) = \{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_q\}$ *con* q *blocchi è detta:*

- Ammissibile *se il CS* $\mathcal{N}(\mathcal{E}, \Pi(\mathcal{D}))$ *ammette una soluzione.*

- Minima *se è ammissibile e non esiste un'altra partizione* $\Pi'(\mathcal{D})$ *con* $q' < q$ *che è ammissibile.* ■

La seguente proposizione deriva banalmente dai precedenti risultati.

Proposizione 5.2.1. *Una rete con* q *posti, soluzione del problema di identificazione 4.2.1, esiste se e solo se esiste una partizione* $\Pi(\mathcal{D}) = \{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_q\}$ *con* q *blocchi.* ■

Di conseguenza il numero di blocchi di una partizione minima rappresenta il numero minimo di posti che può avere una rete risolvente il problema di identificazione assegnato.

Adesso verrà enunciato un risultato intuitivo che consente di determinare se una partizione è minima.

Proposizione 5.2.2. *Una partizione ammissibile con* q *blocchi è minima se e solo se non esiste una partizione ammissibile con* $q - 1$ *blocchi.*

Dimostrazione. *La parte* solo se *segue dalla definizione di partizione minima.*

Per dimostrare la parte se *è necessario dimostrare che, se non esiste alcuna partizione ammissibile con* $q - 1$ *blocchi, allora nessuna partizione con un numero più piccolo di blocchi è ammissibile. Questo può essere dimostrato per contraddizione, per mezzo dello stesso argomento usato nella dimostrazione del caso 2 della Proposizione 4.2.1. Infatti, si assume che esista una partizione ammissibile con* $q' < q - 1$ *blocchi; allora esiste una rete risolvente il problema di identificazione con* q' *posti. Tuttavia, si può aggiungere un numero arbitrario di posti duplicati a questa rete e ciò implica che esista una rete risolvente il problema di identificazione con* $q' + 1, q' + 2, ..., q - 1$ *posti. Perciò, in accordo con la Proposizione 5.2.1, esiste una partizione ammissibile con* $q - 1$ *blocchi che è una contraddizione.* □

In accordo con la precedente proposizione, per dimostrare che una partizione ammissibile $\Pi(\mathcal{D})$ con q blocchi è minima è necessario controllare l'ammissibilità di tutte le partizioni di $\mathcal{D}$ con $q - 1$ blocchi. Tuttavia, il numero di partizioni di un insieme di cardinalità n in k blocchi è dato dal *numero di Sterling del secondo tipo* $S(n, k)$ [4][1] che può essere troppo grande per un'analisi esaustiva. Con la ter-

[1] una formula esplicita per il numero di Sterling del secondo tipo è

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\frac{n!}{k!(n-k)!}j^n.$$

minologia introdotta in questa sottosezione, il CS (4.3) può essere visto come un caso speciale del CS (5.1) quando la partizione considerata $\Pi(\mathcal{D})$ contiene tutti i singoli elementi, cioè è l'unica partizione con $m_{\mathcal{D}}$ blocchi. Di conseguenza è possibile controllare se questa partizione è minima verificando che tutte le partizioni di $m_{\mathcal{D}} - 1$ blocchi (ottenute fondendo qualsiasi tra due singoli elementi) non siano ammissibili. Esistono $S(m_{\mathcal{D}}, m_{\mathcal{D}} - 1) = m_{\mathcal{D}}(m_{\mathcal{D}} - 1)/2$ di queste partizioni.

Sebbene i risultati precedenti forniscano una procedura per ridurre il numero di posti di una rete di Petri, una ricerca brutale per determinare una partizione minima ammissibile non è praticabile dato il gran numero di tali partizioni. Al limite, la procedura di pre-riduzione dei posti illustrata si potrebbe utilizzare quando la cardinalità di $\mathcal{D}$ vale poche unità, ma in generale è più conveniente seguire altre strade, come vedremo nella prossima sottosezione.

5.2.2 Informazioni aggiuntive sulla rete

In questa sottosezione viene fatta una discussione informale su come può essere possibile sfruttare alcune informazioni aggiuntive sulla rete per determinare una partizione ammissibile di cardinalità inferiore a $m_{\mathcal{D}}$. In ogni caso non è detto che una partizione così determinata sia anche minima.

Come esempio, si assuma che sia noto che una transizione $\tilde{t}$ abbia solo un posto in ingresso: un caso simile è quello in cui la rete da identificare o una sottorete di questa contenente $\tilde{t}$ sia una macchina di stato. In tal caso, è possibile considerare una partizione di $\mathcal{D}$ in cui un singolo blocco

$$\tilde{\mathcal{D}} = \{(\sigma_1, \tilde{t}), ..., (\sigma_r, \tilde{t})\} \subset \mathcal{D}$$

contenga tutte le condizioni di disabilitazione per la transizione $\tilde{t}$ e un singolo posto $\tilde{p}$ verrà utilizzato per disabilitare $\tilde{t}$ dopo tutti i σ_i.

Come secondo esempio, si assuma che sia noto che due transizioni t e t' siano in una relazione di scelta libera, cioè esiste un posto $p = {}^\bullet t = {}^\bullet t'$ che è l'unico posto in ingresso per entrambe le transizioni. In tal caso, è possibile considerare una partizione di $\mathcal{D}$ in cui un singolo blocco $\tilde{\mathcal{D}} = \{(\sigma_1, t), ..., (\sigma_r, t), (\sigma'_1, t'), ..., (\sigma_{r'}, t')\}$ $\subset$ $\mathcal{D}$ contenga tutte le condizioni di disabilitazione per le transizioni t e t' che saranno forzate dal posto p.

Vorrei rimarcare che non stiamo riducendo la cardinalità dell'insieme $\mathcal{D}$, ma stiamo semplicemente raggruppando i suoi elementi. Quello che cambia è solo il numero di posti, che viene ridotto ed è pari al numero di blocchi della partizione

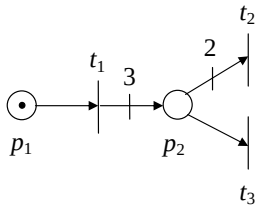

Figura 5.1: Il sistema di rete di Petri considerato nell'Esempio 5.2.1.

$\Pi(\mathcal{D})$. Alle condizioni di disabilitazione per le transizioni che hanno più di un posto in ingresso assegnamo (per adesso) un posto per ciascuna condizione e quindi abbiamo blocchi corrispondenti di un solo elemento.

Esempio 5.2.1. *Consideriamo un linguaggio* $\mathcal{L} = \{\varepsilon, t_1, t_1t_2, t_1t_3, t_1t_2t_3, t_1t_3t_2, t_1t_3t_3\}$ *e supponiamo che* k *sia pari a 3. Abbiamo le seguenti informazioni aggiuntive: la transizione* t_1 *ha solo un posto in ingresso e le transizioni* t_2 *e* t_3 *sono in una relazione di scelta libera. Gli insiemi dei vincoli di abilitazione e di disabilitazione sono rispettivamente:* $\mathcal{E} = \{(\varepsilon, t_1), (t_1, t_2), (t_1, t_3), (t_1t_2, t_3), (t_1t_3, t_2), (t_1t_3, t_3)\}$ *e* $\mathcal{D} = \{(\varepsilon, t_2), (\varepsilon, t_3), (t_1, t_1), (t_1t_2, t_1), (t_1t_2, t_2), (t_1t_3, t_1)\}$.

Le informazioni aggiuntive ci permettono di considerare due differenti blocchi di $\mathcal{D}$*:* $\tilde{\mathcal{D}}_1 = \{(t_1, t_1), (t_1t_2, t_1), (t_1t_3, t_1)\} \subset \mathcal{D}$ *e* $\tilde{\mathcal{D}}_2 = \{(\varepsilon, t_2), (\varepsilon, t_3), (t_1t_2, t_2)\} \subset \mathcal{D}$*. Osserviamo che* $\mathcal{D} = \tilde{\mathcal{D}}_1 \cup \tilde{\mathcal{D}}_2$*, quindi la soluzione rete di Petri ha* $|P| = 2$*. Un sistema di rete soluzione di* $\mathcal{N}(\mathcal{E}, \mathcal{D})$ *calcolato con GLPK è riportato in Figura 5.1. Ricordiamo che una tale soluzione viene determinata associando la funzione obiettivo lineare (4.5) e risolvendo il risultante PPL. Da notare infine che la soluzione trovata è intera.* □

Adesso vediamo come si potrebbero ridurre ulteriormente il numero di vincoli e il numero di posti sulla base delle osservazioni fatte a tal proposito nel capitolo precedente.

Per ogni blocco di $\Pi(\mathcal{D})$ si dovrebbe avere un numero di vincoli pari al numero degli elementi contenuti nel blocco. In realtà, all'interno di ciascun blocco si possono eliminare quegli elementi che hanno vettori di scatto $\vec{\sigma}$ uguali e che contemporaneamente disabilitano la stessa transizione t dopo lo scatto di σ, lasciandone solo uno. In questo modo il numero di vincoli associati ad un singolo blocco viene ridotto e risulta pari al numero di differenti coppie $(\vec{\sigma}, \vec{\sigma} + \vec{t})$ relative agli elementi del blocco. Il numero di posti associato a ciascun blocco non cambia, è sempre 1. Alle condizioni di disabilitazione per le transizioni che hanno più di un posto in ingresso non assegnamo più un singolo posto per ciascuna

condizione, ma anche qui eliminiamo gli elementi che hanno vettori di scatto $\vec{\sigma}$ uguali e che contemporaneamente disabilitano la stessa transizione dopo lo scatto di σ, lasciandone solo uno a cui assegnamo un posto. In questo modo si riduce il numero di posti e ovviamente il numero di vincoli. Nell'Esempio 5.2.1 non si può eliminare alcun elemento. Comunque valuteremo nel capitolo 7 il vantaggio di questa riduzione.

5.3 Post-riduzione dei posti

5.3.1 Il minimo *hitting set*

Una volta identificata una rete risolvendo il CS (4.3) (o anche il CS (5.1)) è sempre possibile controllare se alcuni posti sono ridondanti e possono essere rimossi senza intaccare la correttezza del risultato. Questa verifica è basata sulla nozione di *minimo hitting set* definita in seguito.

Proposizione 5.3.1. *Consideriamo un sistema di rete $\langle N, M_0\rangle$ soluzione del problema di identificazione 4.2.1 e definiamo per tutte le condizioni di disabilitazione $(\sigma, t) \in \mathcal{D}$ l'insieme*

$$P_{(\sigma,t)} = \{p \in P \mid M_0(p) + C(p,\cdot) \cdot \vec{\sigma} < Pre(p,t)\} \tag{5.2}$$

consistente di tutti i posti della rete che disabilitano la transizione t dopo che la sequenza σ è stata eseguita. Si assume che $\hat{P} \subset P$ sia un hitting set per tutti i $P_{(\sigma,t)}$, cioè $\hat{P} \cap P_{(\sigma,t)} \neq \emptyset$ per ogni $(\sigma, t) \in \mathcal{D}$. Allora il sistema di rete $\langle \hat{N}, \hat{M}_0\rangle$ ottenuto da $\langle N, M_0\rangle$ rimuovendo tutti i posti in $P \backslash \hat{P}$ è una soluzione del problema di identificazione 4.2.1.

Dimostrazione. *Come già discusso nella dimostrazione della Proposizione 4.2.1 la rimozione di un posto non modifica alcuna condizione di abilitazione. Inoltre, se $\hat{P}$ è un hitting set per tutti i $P_{(\sigma,t)}$ allora è capace di forzare tutte le condizioni di disabilitazione in $\mathcal{D}$. Perciò $L_k(\hat{N}, \hat{M}_0) = L_k(N, M_0)$.* □

I posti in $P \setminus \hat{P}$ che possono essere rimossi dal sistema di rete $\langle N, M_0\rangle$ senza modificare il suo linguaggio $L_k(N, M_0)$ sono detti *posti ridondanti*.

Poiché la rete $\hat{N}$ ha un insieme di posti $\hat{P}$, per ottenere una rete con un insieme minimo di posti è necessario determinare il minimo hitting set. È noto che questo problema è NP-hard ed esistono diversi modi per calcolare i minimi hitting set

[8]. In [13] è stato presentato un semplice algoritmo basato sulla programmazione intera.

Proposizione 5.3.2. *Consideriamo un sistema di rete $\langle N, M_0\rangle$ con $q = |P|$ posti soluzione del problema di identificazione 4.2.1. Per tutte le condizioni di disabilitazione $(\sigma, t) \in \mathcal{D}$, sia $\vec{y}_{(\sigma,t)} \in \{0,1\}^q$ il vettore caratteristico dell'insieme $P_{(\sigma,t)}$ definito nella (5.2), cioè $y_{(\sigma,t)}(p) = 1$ se $p \in P_{(\sigma,t)}$, altrimenti $y_{(\sigma,t)}(p) = 0$.*

Consideriamo il seguente problema di programmazione intera (PPI):

$$\begin{cases} minimizza & \vec{1}^T \cdot \vec{x} \\ tale\ che & \vec{x}^T \cdot \vec{y}_{(\sigma,t)} \geq 1 \qquad \forall (\sigma,t) \in \mathcal{D} \\ & \vec{x} \in \{0,1\}^q \end{cases} \tag{5.3}$$

e sia $\vec{x}^*$ una soluzione ottima. Allora un minimo hitting set per tutti i $P_{(\sigma,t)}$ è l'insieme $\hat{P} = \{p \in P \mid x^*(p) = 1\}$.

Dimostrazione. *È immediato vedere che qualunque soluzione ammissibile $\vec{x}$ del PPI (5.3) è il vettore caratteristico di un hitting set per tutti i $P_{(\sigma,t)}$ perché contiene almeno un elemento da ognuno di questi insiemi ($\vec{x}^T \cdot \vec{y}_{(\sigma,t)} \geq 1$). La soluzione ottima $\vec{x}^*$ ha il minimo numero di componenti non nulle, quindi corrisponde ad un minimo hitting set.* □

5.3.2 Esempi

Esempio 5.3.1. *Supponiamo che $\mathcal{L} = \{\varepsilon, t_1, t_2\}$ e $k = 2$. Osserviamo che questo è un caso particolare di un linguaggio non contenente alcuna parola di lunghezza k, cioè tutte le parole di lunghezza k devono essere disabilitate. Gli insiemi dei vincoli di abilitazione e disabilitazione sono rispettivamente: $\mathcal{E} = \{(\varepsilon, t_1), (\varepsilon, t_2)\}$ e $\mathcal{D} = \{(t_1, t_1), (t_1, t_2), (t_2, t_1), (t_2, t_2)\}$.*

Un sistema di rete soluzione di $\mathcal{N}(\mathcal{E}, \mathcal{D})$ è riportato in Figura 5.2(a): in questo caso vale ovviamente $|P| = m_{\mathcal{D}} = 4$. Come già noto, una tale soluzione si determina associando la funzione obiettivo lineare (4.5) a $\mathcal{N}(\mathcal{E}, \mathcal{D})$ e risolvendo il risultante PPL.

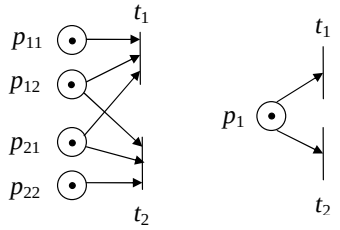

Figura 5.2: I sistemi di rete di Petri considerati nell'Esempio 5.3.1.

Adesso proviamo a ridurre il numero di posti utilizzando l'approccio della post-riduzione. A tal scopo per ogni (σ, t) si può calcolare l'insieme $P_{(\sigma,t)}$ definito come nell'equazione (5.2). Essendo $P_{(t_1,t_1)} = \{p_{11}, p_{12}, p_{21}\}$, $P_{(t_1,t_2)} = \{p_{12}, p_{21}\}$, $P_{(t_2,t_1)} = \{p_{12}, p_{21}\}$, $P_{(t_2,t_2)} = \{p_{12}, p_{21}, p_{22}\}$, è immediato vedere che un possibile hitting set per tutti i $P_{(\sigma,t)}$ è $\hat{P} = \{p_{21}\}$, che è anche minimo. Il risultante sistema di rete $\langle \hat{N}, \hat{M}_0 \rangle$ ottenuto da quello precedente rimuovendo tutti i posti in $P \setminus \hat{P}$ è mostrato in Figura 5.2(b). Tutte le soluzioni, ottenute con GLPK, sono intere. □

È importante osservare che la procedura di post-riduzione dei posti non necessariamente assicura che la rete risultante sia la soluzione di una data procedura di identificazione con il minimo numero di posti. Infatti la post-riduzione determina solamente, tra le soluzioni di una data procedura di identificazione *che può essere ottenuta da una soluzione N rimuovendo dei posti*, quella con un numero minimo di posti. Il seguente esempio chiarirà questo punto.

Esempio 5.3.2. *Supponiamo che $\mathcal{L} = \{\varepsilon, t_1, t_2, t_3\}$ e $k = 2$, perciò come nell'Esempio 5.3.1, tutte le parole di lunghezza k devono essere disabilitate. Gli insiemi dei vincoli di abilitazione e disabilitazione sono rispettivamente: $\mathcal{E} = \{(\varepsilon, t_1), (\varepsilon, t_2), (\varepsilon, t_3)\}$ e $\mathcal{D} = \{(t_1, t_1), (t_1, t_2), (t_1, t_3), (t_2, t_1), (t_2, t_2), (t_2, t_3), (t_3, t_1), (t_3, t_2), (t_3, t_3)\}$.*

Un sistema di rete soluzione del CS (4.3) è mostrato in Figura 5.3(a), dove ovviamente $|P| = m_{\mathcal{D}} = 9$. Esso è stato determinato risolvendo un PPL la cui funzione obiettivo è come sempre la (4.5).

Ora, essendo $P_{(t_1,t_1)} = \{p_{11}, p_{12}, p_{13}, p_{21}, p_{31}\}$, $P_{(t_1,t_2)} = \{p_{12}, p_{21}\}$, $P_{(t_1,t_3)} = \{p_{13}, p_{31}\}$, $P_{(t_2,t_1)} = \{p_{12}, p_{21}\}$, $P_{(t_2,t_2)} = \{p_{11}, p_{21}, p_{22}, p_{23}, p_{32}\}$, $P_{(t_2,t_3)} = \{p_{23}, p_{32}\}$, $P_{(t_3,t_1)} = \{p_{13}, p_{31}\}$, $P_{(t_3,t_2)} = \{p_{23}, p_{32}\}$ e $P_{(t_3,t_3)} = \{p_{13}, p_{23}, p_{31}, p_{32}, p_{33}\}$, è facile vedere che un possibile hitting set per tutti i $P_{(\sigma,t)}$ è $\hat{P} = \{p_{21}, p_{31}, p_{32}\}$. Il sistema di rete $\langle \hat{N}, \hat{M}_0 \rangle$ ottenuto da $\langle N, M_0 \rangle$ rimuovendo tutti i posti in $P \setminus \hat{P}$

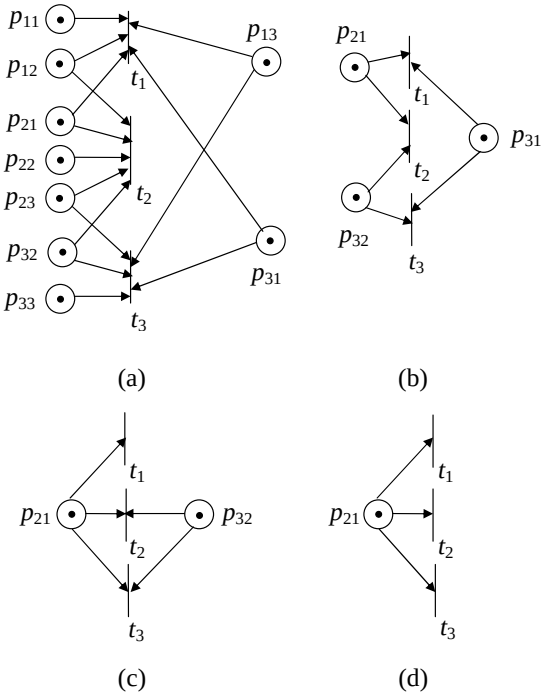

Figura 5.3: I sistemi di rete di Petri considerati nell'Esempio 5.3.2.

è mostrato in Figura 5.3(b). Da notare che lo stesso hitting set può anche essere ottenuto risolvendo il PPI (5.3).

Una tale soluzione non è minima in termini di posti, come può essere verificato utilizzando la procedura di pre-riduzione dei posti presentata nella sottosezione 5.2.1. A questo scopo dapprima viene esaminata una soluzione con due posti, cioè utilizzando la notazione della Definizione 5.2.1 si controlla se esiste una soluzione con solo due blocchi $\mathcal{D}'_1$ e $\mathcal{D}'_2$, ottenuta fondendo due dei tre blocchi $\mathcal{D}_1 = \{(t_1,t_1),(t_1,t_2),(t_2,t_1),(t_2,t_2)\}$, $\mathcal{D}_2 = \{(t_2,t_3),(t_3,t_2),(t_3,t_3)\}$ e $\mathcal{D}_3 = \{(t_1,t_3),(t_3,t_1)\}$ relativi rispettivamente ai posti p_{21}, p_{32} e p_{31} della rete in Figura 5.3(b). In particolare, si trova che il CS (5.1) ammette una soluzione quando $\mathcal{D}'_1 = \{(t_1,t_1),(t_1,t_2),(t_1,t_3),(t_2,t_1),(t_2,t_2),(t_3,t_1)\}$ e $\mathcal{D}'_2 = \{(t_2,t_3),(t_3,t_2),(t_3,t_3)\}$, dove $\mathcal{D}'_1 = \mathcal{D}_1 \cup \mathcal{D}_3$. La rete risultante $\langle \bar{N}, \bar{M}_0 \rangle$ è mostrata in Fig. 5.3(c), dove $\bar{P} = \{p_{21}, p_{32}\}$.

Si può ulteriormente ridurre la rete trovando un minimo hitting set. Essendo $P_{(t_1,t_1)} = \{p_{21}\}$, $P_{(t_1,t_2)} = \{p_{21}\}$, $P_{(t_1,t_3)} = \{p_{21}\}$, $P_{(t_2,t_1)} = \{p_{21}\}$, $P_{(t_2,t_2)} = \{p_{21}, p_{32}\}$, $P_{(t_2,t_3)} = \{p_{21}, p_{32}\}$, $P_{(t_3,t_1)} = \{p_{21}\}$, $P_{(t_3,t_2)} = \{p_{21}, p_{32}\}$ e $P_{(t_3,t_3)} = \{p_{21}, p_{32}\}$, l'unico hitting set minimo è $\hat{P}' = \{p_{21}\}$. Il sistema di rete $\langle \hat{N}', \hat{M}'_0 \rangle$

ottenuto da $\langle \bar{N}, \bar{M}_0 \rangle$ rimuovendo tutti i posti in $\bar{P} \setminus \hat{P}'$ è minimo ed è mostrato in Figura 5.3(d).

Da notare che in tutti i casi le soluzioni di rete ottenute con GLPK sono intere. □

L'applicazione della procedura di pre-riduzione della sottosezione 5.2.1 dopo la post-riduzione e l'applicazione di più post-riduzioni all'interno della stessa procedura di identificazione sono operazioni che non verranno più svolte nel prosieguo della tesi. Nell'esempio precedente è stato fatto questo solo per mostrare che la post-riduzione non garantisce di trovare una soluzione col minimo numero di posti. È stato semplice applicare la procedura di pre-riduzione in questo caso visto che partivamo da una rete con soli 3 posti ma, come ho già detto alla fine della sottosezione 5.2.1, in generale si preferisce non utilizzarla, ma si sfruttano eventualmente alcune informazioni sulla rete, poiché solitamente si ha a che fare con un numero elevato di possibili partizioni dell'insieme $\mathcal{D}$. In definitiva nelle simulazioni, una volta identificata una rete, si esegue un'unica post-riduzione e ci si ferma.

Come nota finale osserviamo che la nozione di posto ridondante data qui è differente da quella di *posto implicito* utilizzata da altri autori [6]. Infatti, un posto implicito è un posto che può essere rimosso da una rete $\langle N, M_0 \rangle$ senza cambiare il suo comportamento globale $L(N, M_0)$. Al contrario, un posto ridondante secondo la definizione data in questa tesi è un posto che può essere rimosso dalla rete senza cambiare il prefisso finito $L_k(N, M_0)$ del comportamento globale. Perciò la nozione data qui è più debole e le tecniche utilizzate nel [6] per determinare i posti impliciti non possono essere usate nella nostra struttura.

Capitolo 6

Toolbox di Matlab

6.1 Presentazione generale

In questo capitolo descriverò i programmi che ho realizzato col software Matlab i quali implementano la procedura di identificazione di una rete P/T descritta nei capitoli 4 e 5.

In Figura 6.1 viene mostrato uno schema a blocchi della procedura in cui ogni blocco corrisponde ad ingressi e/o a uscite dei programmi. In particolare, i blocchi rappresentati da rettangoli continui contengono degli argomenti che sono sempre utilizzati in qualsiasi istanza della procedura mentre i blocchi rappresentati da rettangoli tratteggiati sono parti opzionali e non sempre presenti. Le frecce si possono distinguere tra frecce etichettate le cui etichette rappresentano determinati ingressi e frecce non etichettate che corrispondono all'esecuzione di un determinato programma. Anche per queste, come per i blocchi, si fa la stessa distinzione per quanto riguarda le modalità di utilizzo tra frecce continue e tratteggiate.

Adesso mostrerò in generale quali programmi ho realizzato riferendomi allo schema in Figura 6.1, poi nelle sezioni successive procederò con la descrizione dei singoli programmi.

Il primo programma realizzato è quello corrispondente alla freccia diretta dal blocco "**maxL, n, (k)**" al blocco "**E, D**" che estrae gli insiemi delle condizioni di abilitazione e di disabilitazione a partire da un prefisso finito del linguaggio osservato. Esattamente **maxL** indica il linguaggio massimale, cioè contiene solo le stringhe di massima lunghezza. Ho fatto ciò per comodità di lettura dato che tutte le altre

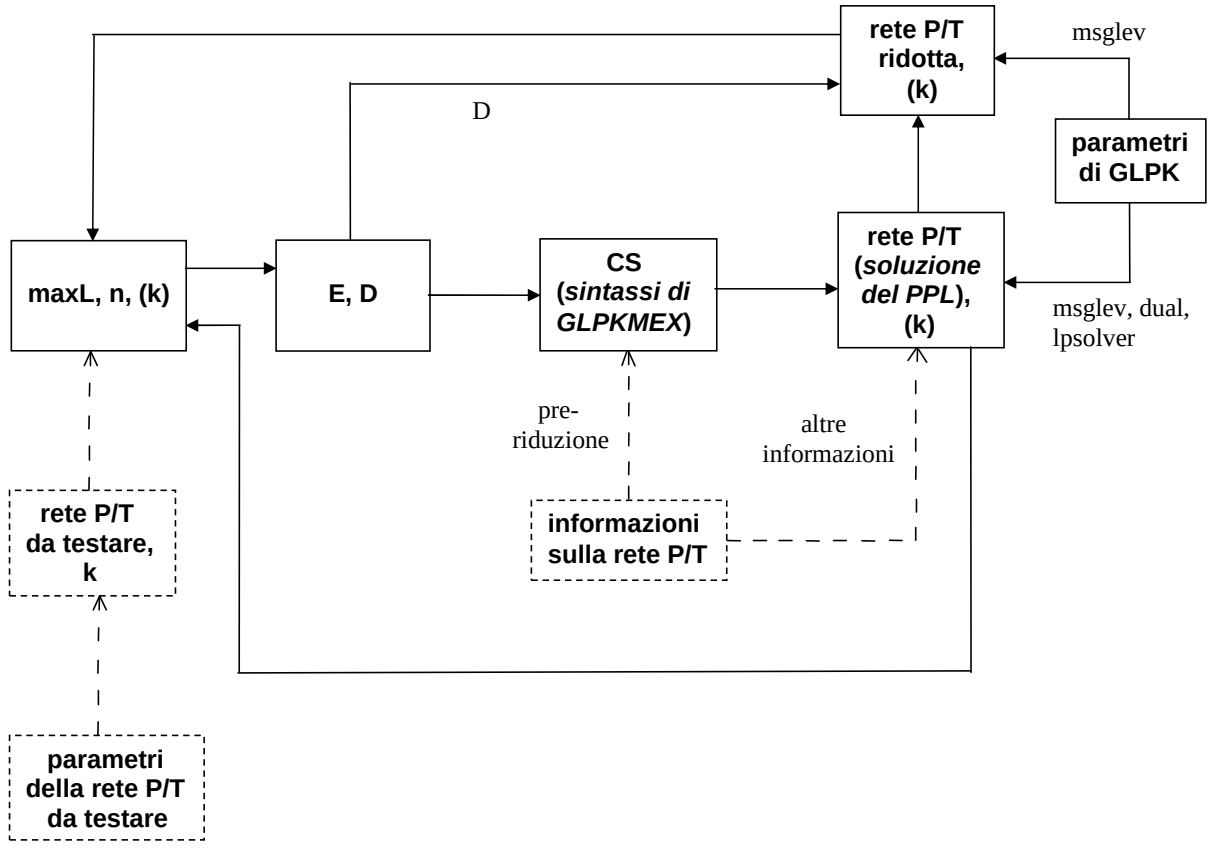

Figura 6.1: Schema a blocchi della procedura di identificazione.

stringhe del linguaggio sono prefissi o sottostringhe di queste. L'intero **k** è racchiuso tra parentesi perché non è sia ingresso che uscita di programmi, ma solo ingresso.

Se inizialmente ho a disposizione una rete da testare da cui mi voglio ricavare il linguaggio generato, ho bisogno di un altro programma che compie questa operazione. Tale programma è quello corrispondente alla freccia diretta dal blocco "**rete P/T da testare, k**" al blocco "**maxL, n, (k)**". Il blocco "**rete P/T da testare, k**" è tratteggiato in quanto non sempre ho una struttura di rete nota in partenza e quindi non sempre devo far uso di questo programma. Se la rete da testare è parametrica, ho bisogno di un programma che ne ricava la struttura al variare dei parametri e corrisponde alla freccia tratteggiata che va dal blocco "**parametri della rete P/T da testare**" al blocco "**rete P/T da testare, k**".

Per ottenere il *Constraint Set* (**CS**) da **E** e **D** ho sviluppato un altro programma che prevede anche la possibilità di applicare la pre-riduzione se ci sono transizioni con un solo posto in ingresso e transizioni in relazione di scelta libera. Gli ingressi al programma sono inseriti nei due blocchi "**E, D**" e "**informazioni sulla rete P/T**". Quest'ultimo, così come la freccia etichettata con "pre-riduzione", è tratteggiato perché non sempre ho delle informazioni sulla rete. L'uscita corrisponde al blocco "**CS (*sintassi di GLPKMEX*)**" ad evidenziare che i vincoli del problema di programmazione lineare (PPL) devono essere espressi secondo la sintassi di GLPKMEX.

A questo punto è necessario richiamare proprio GLPKMEX per risolvere il problema di programmazione e ricavare la rete. A tal scopo ho realizzato un programma che parte dal CS e sfruttando alcune informazioni opzionali sulla rete (freccia etichettata con "altre informazioni"), che consentono di aggiungere nuovi vincoli al CS stesso, identifica la rete (blocco "**rete P/T (*soluzione del PPL*), (k)**"). Tali informazioni sono state inserite in questo momento e non precedentemente, insieme a quelle per la pre-riduzione, perché per definire le informazioni legate al numero di posti (P-vettori, grafo marcato, GMEC) occorre conoscere prima quanti posti ha la rete identificata per poter aggiungere correttamente i nuovi vincoli. Altri ingressi a questo programma sono contenuti nel blocco "**parametri di GLPK**". Questi permettono di scegliere il metodo di risoluzione del PPL e di definire le modalità di visualizzazione dell'uscita nella finestra dei comandi di Matlab (freccia etichettata con "msglev, dual, lpsolver").

Prima di applicare la post-riduzione, per controllare la correttezza della procedura di identificazione verifico che la rete trovata generi un linguaggio massimale esattamente uguale a quello osservato. Bisogna quindi rilanciare il programma

che calcola il linguaggio. Anche in questo caso **k** è tra parentesi perché è solo ingresso.

Dunque, adesso è necessario un altro programma che implementi la post-riduzione e questo corrisponde alla freccia che va dal blocco "**rete P/T (*soluzione del PPL*), (k)**" al blocco "**rete P/T ridotta, (k)**". Abbiamo bisogno anche dell'ingresso **D** (freccia etichettata con "D") per calcolare i vettori caratteristici e inoltre ho settato anche l'ingresso msglev (freccia etichettata con "msglev") per scegliere come visualizzare sullo schermo l'uscita. Ancora una volta il **k** tra parentesi è solo ingresso.

Successivamente bisogna valutare la bontà della procedura andando a verificare che il linguaggio generato sia lo stesso di quello di partenza. Quindi richiamiamo ancora una volta il programma che estrae il linguaggio della rete. Questo passaggio corrisponde alla freccia che va dal blocco "**rete P/T ridotta, (k)**" al blocco "**maxL, (k), n**".

Infine, ho implementato un programma che esegue l'intera procedura richiamando i programmi sopracitati.

Ho espresso il linguaggio, gli insiemi $\mathcal{E}$ e $\mathcal{D}$, il CS, le reti e molte variabili utilizzate all'interno dei programmi come array. Per costruire gli algoritmi ho utilizzato preferibilmente come strutture di controllo di flusso cicli *for* e *while* e istruzioni *if* e *break*. Quando è stato possibile ho usato la pre-allocazione degli array per ottenere una più alta velocità di Matlab.

Le funzioni realizzate hanno la seguente struttura standard. La prima riga contiene la sintassi della funzione, poi c'è una brevissima descrizione su cosa fa la funzione, a seguire ci sono le liste degli argomenti in ingresso e in uscita coi relativi commenti e il numero di questi argomenti memorizzato nelle variabili *nargin* e *nargout*. Il resto del file rappresenta codice eseguibile Matlab con relativi commenti.

Ho creato altri M-file come script, uno per ciascun programma, che contengono i dati in ingresso al programma in modo da poterli settare facilmente. In realtà non sempre è necessario ricorrere ad un file per definire gli ingressi, ma è sufficiente che questi siano nel Workspace di Matlab. I file degli ingressi contengono la sintassi della corrispondente funzione, una sua brevissima descrizione e la lista degli ingressi.

Nelle sezioni successive per ciascun programma creato riporterò la sintassi della funzione che lo realizza, la lista degli argomenti di ingresso e di uscita, la de-

scrizione di ciò che esso fà senza entrare nel dettaglio delle singole istruzioni e un esempio di applicazione del programma tratto direttamente dalla finestra dei comandi di Matlab.

6.2 Calcolo del linguaggio massimale

Il programma che calcola il linguaggio massimale è stato realizzato attraverso la funzione `PN_k2maxL_n` che ha la seguente sintassi:

```
[maxL,n]=PN_k2maxL_n(M0,Post,Pre,k).
```

Gli argomenti di ingresso sono il vettore colonna marcatura iniziale $M0$ di dimensione m e le matrici $Post$ e Pre di dimensione $m \times n$ che rappresentano la struttura della rete da testare e k che, ricordiamo, è un numero naturale maggiore o uguale alla lunghezza delle stringhe del linguaggio massimale. Ricordiamo anche che tale linguaggio è un sottoinsieme del linguaggio globale della rete.

Gli argomenti di uscita sono la matrice $maxL$ che rappresenta appunto il linguaggio massimale e l'intero n ossia il numero di transizioni. Ogni riga di $maxL$ contiene gli indici i delle transizioni t_i che compongono una sequenza di scatto del linguaggio, disposte nell'ordine in cui compaiono in tale sequenza. Come verrà spiegato nella prossima sezione, l'uscita n è stata inclusa perché serve come ingresso per il programma che costruisce gli insiemi $\mathcal{E}$ e $\mathcal{D}$.

L'algoritmo implementato dalla funzione esegue i seguenti passi:

1. Ricava il valore di n che corrisponde al numero di colonne di $Post$ e Pre.

2. Se $k = 0$ definisce $maxL$ come una matrice vuota, altrimenti esegue il passo successivo.

3. Calcola le sequenze di scatto di lunghezza unitaria, in pratica le transizioni abilitate dalla marcatura iniziale, le nuove marcature raggiunte e aggiunge quelle sequenze a $maxL$, inizializzata come matrice vuota.

4. Se $k > 1$ calcola il resto di $maxL$ col seguente ciclo *for*. Per ogni marcatura raggiunta dallo scatto di una determinata sequenza di lunghezza $j - 1$, conta il numero di nuove sequenze abilitate di lunghezza j, calcola le nuove

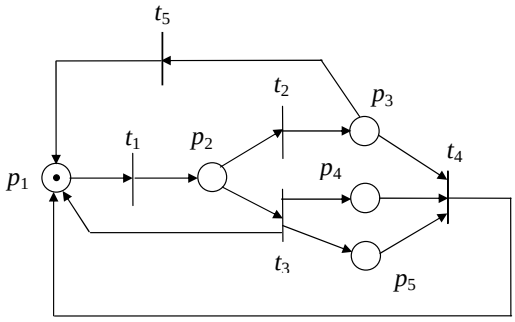

Figura 6.2: La rete di Petri dell'esempio trattato in questo capitolo.

marcature raggiunte e le nuove sequenze, concatenando verticalmente queste ultime a $maxL$. Se durante un'iterazione non ci sono nuove sequenze abilitate termina la sua esecuzione. Se durante un'iterazione ci sono nuove sequenze abilitate, elimina le righe di $maxL$ corrispondenti alle sequenze di lunghezza $j-1$ e procede con l'iterazione successiva. Da notare che se non si eliminassero queste righe si ricaverebbe l'intero linguaggio L_k e non il linguaggio massimale.

Esempio 6.2.1. *Consideriamo la rete di Petri in Figura 6.2. Ho inserito i seguenti dati in ingresso nel file* `inputdata_PN_k2maxL_n`*:*

$$M0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, Post = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}, Pre = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \text{ e } k = 3.$$

Ho scelto $k = 3$ per ottenere poi un Constraint Set *moderatamente piccolo e rapidamente visualizzabile.*

Eseguiamo il programma in Matlab e riportiamo in Figura 6.3 il risultato visualizzato nella finestra dei comandi. Si può vedere come il linguaggio massimale ricavato sia costituito dalle due sequenze $t_1t_2t_5$ e $t_1t_3t_1$, quindi il linguaggio completo è pari a

$$\mathcal{L} = \{\varepsilon, t_1, t_1t_2, t_1t_3, t_1t_2t_5, t_1t_3t_1\}.$$

Osservando la Figura 6.2 si può verificare come il linguaggio calcolato sia effettivamente quello generato dalla rete. Il numero di transizioni della rete è pari a 5. □

```
>> inputdata_PN_k2maxL_n
>> [maxL,n]=PN_k2maxL_n(M0,Post,Pre,k)

maxL =

     1   2   5
     1   3   1


n =

     5
```

Figura 6.3: Finestra dei comandi di Matlab.

6.3 Costruzione degli insiemi $\mathcal{E}$ e $\mathcal{D}$

Il programma che costruisce gli insiemi $\mathcal{E}$ e $\mathcal{D}$ è stato realizzato attraverso la funzione `maxL_n_k2E_D` che ha la seguente sintassi:

```
[E,D]=maxL_n_k2E_D(maxL,n,k).
```

Gli argomenti di ingresso sono la matrice $maxL$ e l'intero n, che sono argomenti di uscita di `PN_k2maxL_n` solo quando si ha a disposizione una rete da testare, e l'intero k, che allo stesso modo è argomento di ingresso di `PN_k2maxL_n` solo quando si ha una rete da testare. Come ho già anticipato nella sezione precedente, ho inserito l'argomento n in ingresso perché non sempre n è il valore massimo contenuto in $maxL$ e quindi non sempre può essere ricavato da questo. Infatti potrebbe capitare che ci siano transizioni mai abilitate dalla rete e quindi non contenute in $maxL$. Anche l'ingresso k sembrerebbe ridondante in quanto dovrebbe coincidere col numero di colonne di $maxL$. Invece è da inserire anch'esso per includere quel caso particolare, che ho già trattato negli esempi 5.3.1 e 5.3.2, in cui le parole del linguaggio massimale hanno lunghezza inferiore a k.

Gli argomenti di uscita sono le matrici E e D che rappresentano rispettivamente l'insieme delle condizioni di abilitazione $\mathcal{E}$ e di disabilitazione $\mathcal{D}$. Ogni riga di E contiene gli indici i delle transizioni t_i che compongono una condizione (σ, t) di $\mathcal{E}$, disposte nell'ordine in cui compaiono in tale condizione. La matrice D ha la stessa tipologia di struttura.

L'algoritmo implementato dalla funzione esegue i seguenti passi:

1. Assegna $maxL$ ad E visto che tutte le stringhe del linguaggio massimale

fanno anche parte di $\mathcal{E}$.

2. Se il numero di colonne di $maxL$ è maggiore di 1, costruisce il resto di E eseguendo il seguente ciclo *while*. Ad ogni iterazione elimina una colonna di $maxL$ e aggiunge ad E le righe differenti della $maxL$ corrente a cui concatena orizzontalmente degli zeri.

3. Ricalcola la $maxL$ originale e inizializza D come matrice vuota.

4. Se k è maggiore del numero di colonne di $maxL$, esegue un ciclo *for* che per ogni riga di $maxL$ inserisce in D n righe. Ciascuna di queste n righe è così strutturata: la riga di $maxL$ considerata a cui si concatena orizzontalmente l'indice di una transizione.

5. Se il numero di colonne di $maxL$ è maggiore di 1, costruisce il resto di D eseguendo il seguente ciclo *while*. Ad ogni iterazione elimina una colonna di $maxL$ e per ogni differente riga della $maxL$ corrente aggiunge a D delle righe così strutturate: la riga di $maxL$ considerata a cui si concatena orizzontalmente l'indice di una transizione disabilitata dopo lo scatto della sequenza corrispondente a questa riga di $maxL$.

6. Quando $maxL$ (quella ridotta dal ciclo *while* di cui al passo 5 o quella di partenza) ha una sola colonna, esegue uno o più cicli *while* che aggiungono a D delle righe aventi un solo elemento non nullo, il primo, di valore diverso da quelli contenuti nell'unica colonna di $maxL$ e comunque compreso tra 1 ed n.

Un esempio chiarirà il funzionamento del programma.

Esempio 6.3.1. *Anche qui consideriamo la rete in Figura 6.2. Come mostrato nell'Esempio 6.2.1 si potrebbero inserire i dati in ingresso su un apposito file che ho creato (*`inputdata_maxL_n_k2E_D`*), ma dato che gli ingressi sono stati salvati nel Workspace dalle precedenti computazioni, si può richiamare direttamente la funzione* `maxL_n_k2E_D`*. Ricordiamo che gli ingressi sono:*

$$maxL = \begin{bmatrix} 1 & 2 & 5 \\ 1 & 3 & 1 \end{bmatrix}, \quad n = 5 \quad e \quad k = 3.$$

Eseguiamo il programma in Matlab e riportiamo in Figura 6.4 il risultato visualizzato nella finestra dei comandi. Si può vedere come gli insiemi $\mathcal{E}$ e $\mathcal{D}$ ricavati siano rispettivamente:

$$\mathcal{E} = \{(\varepsilon, t_1), (t_1, t_2), (t_1, t_3), (t_1t_2, t_5), (t_1t_3, t_1)\}$$

```
>> [E,D]=maxL_n_k2E_D(maxL,n,k)

E =

     1   0   0
     1   2   0
     1   3   0
     1   2   5
     1   3   1


D =

     2   0   0
     3   0   0
     4   0   0
     5   0   0
     1   1   0
     1   4   0
     1   5   0
     1   2   1
     1   2   2
     1   2   3
     1   2   4
     1   3   2
     1   3   3
     1   3   4
     1   3   5
```

Figura 6.4: Finestra dei comandi di Matlab.

e

$$\mathcal{D} = \{(\varepsilon, t_2), (\varepsilon, t_3), (\varepsilon, t_4), (\varepsilon, t_5), (t_1, t_1), (t_1, t_4), (t_1, t_5), (t_1t_2, t_1), (t_1t_2, t_2), (t_1t_2, t_3), (t_1t_2, t_4), (t_1t_3, t_2), (t_1t_3, t_3), (t_1t_3, t_4), (t_1t_3, t_5)\}.$$

Osservando la Figura 6.4 si può vedere come l'algoritmo descritto sia stato correttamente applicato. Infatti, per quanto riguarda E, le ultime due righe coincidono con $maxL$ e le righe sopra sono state ottenute eliminando una colonna di $maxL$ ad ogni iterazione e aggiungendo degli zeri. Per quanto riguarda D, le sue righe sono formate da righe di E private dell'ultimo elemento non nullo, che è stato sostituito da tutte le transizioni disabilitate dopo lo scatto delle sequenze σ corrispondenti a queste righe di E. Le prime 4 righe con un solo elemento non nullo rappresentano tutte le transizioni disabilitate dalla marcatura iniziale. □

6.4 Costruzione del *Constraint Set* (CS)

Il programma che costruisce il CS è stato realizzato attraverso la funzione `E_D_pre_red2CS` che ha la seguente sintassi:

```
[q,A,b,ctype,lb,vartype]=E_D_pre_red2CS(E,D,pre_red).
```

Gli argomenti di ingresso sono le matrici E e D, che sono argomenti di uscita di `maxL_n_k2E_D`, e la matrice pre_red che rappresenta le informazioni sulla rete (se ne abbiamo) che consentono di effettuare una pre-riduzione dei posti. Quindi se non ci sono tali informazioni pre_red è vuota altrimenti è strutturata in questo modo: le righe con un solo elemento non nullo che vale i indicano che la transizione t_i ha solo un posto in ingresso, mentre le altre righe con più di un elemento non nullo, se esistono, indicano che i valori degli elementi non nulli corrispondono a transizioni in una relazione di scelta libera.

Gli argomenti di uscita sono l'intero q ossia il numero di posti della rete, la matrice A dei coefficienti del CS, il vettore colonna b dei termini noti dei vincoli, il vettore colonna di caratteri $ctype$ che contiene il senso di ogni vincolo (li impongo tutti '$\geq$'), il vettore lb che contiene il limite inferiore di ciascuna delle variabili incognite e il vettore colonna di caratteri $vartype$ che contiene la definizione del tipo di variabili (continue per un PPL). Gli argomenti di uscita $A, b, ctype, lb$ e $vartype$ definiscono il CS secondo la sintassi di GLPKMEX. Il limite superiore delle variabili non è stato specificato perché è infinito.

L'algoritmo implementato dalla funzione esegue i seguenti passi:

1. Esegue un ciclo *for* che calcola i vettori di scatto $\vec{\sigma}$ e $\vec{\sigma} + \vec{t}$ associati alle rispettive sequenze di scatto σ e (σ, t) in E e D.

2. Inizializza il numero di posti della rete pari alla cardinalità di $\mathcal{D}$.

3. Se son presenti informazioni sulla rete per poter applicare la pre-riduzione, esegue un ciclo *for* per ridurre il numero di posti ed eventualmente la cardinalità di $\mathcal{D}$ analizzando una alla volta le transizioni, a partire da t_1 sino a t_n.

   Se t_i ha un solo posto in ingresso e non è in relazione di scelta libera con altre transizioni, aggiorna il numero di posti che viene prima decrementato di una quantità pari al numero di $\vec{\sigma}$ o $\vec{\sigma} + \vec{t}$ corrispondenti a t_i e poi viene incrementato di un'unità. In pratica viene formato il blocco di una partizione ammissibile di $\mathcal{D}$ contenente le condizioni di disabilitazione per t_i (decremento di q) e, come sappiamo, viene assegnato un unico posto per disabilitare t_i dopo lo scatto di tutti i σ_i (incremento di q). Poi elimina, se ce ne sono, le coppie $(\vec{\sigma}, \vec{\sigma} + \vec{t})$ uguali all'interno del blocco che altrimenti genererebbero vincoli uguali nel CS modificato.

   Se t_i non ha un solo posto in ingresso effettua operazioni simili con qualche piccola differenza. Infatti, per queste transizioni non si forma un blocco contenente tutti i (σ, t_i) a cui assegnare un posto, ma si assegna un posto ad ogni (σ, t_i). In pratica si stanno formando blocchi di un solo elemento. Quindi se ci sono coppie $(\vec{\sigma}, \vec{\sigma} + \vec{t_i})$ uguali occorre eliminare queste coppie e i posti (o i blocchi) corrispondenti per non avere vincoli uguali nel problema.

   Se t_i è in relazione di scelta libera con una o più transizioni effettua nuovamente operazioni simili alle precedenti sempre con qualche differenza. Infatti questa volta non incrementa di 1 i posti per assegnare un posto al blocco formato dai (σ, t_i) perché il blocco dev'essere formato anche dai (σ, t) delle altre transizioni in relazione di scelta libera con t_i. L'aggiornamento dei posti relativo ai blocchi verrà fatto alla fine del ciclo *for*. Poi elimina le coppie $(\vec{\sigma}, \vec{\sigma} + \vec{t_i})$ uguali.

4. Dunque, terminato il ciclo *for*, se son presenti transizioni in relazione di scelta libera incrementa il numero di posti di una quantità pari al numero di blocchi corrispondenti a tali transizioni. Il numero di posti q è stato così calcolato.

5. Se non ci sono informazioni di pre-riduzione, esegue un ciclo *while* per eliminare le coppie $(\vec{\sigma}, \vec{\sigma} + \vec{t})$ uguali relative all'insieme $\mathcal{D}$ in modo da eliminare i posti ridondanti.

6. Esegue un ciclo *while* per eliminare le coppie $(\vec{\sigma}, \vec{\sigma} + \vec{t})$ uguali relative all'insieme $\mathcal{E}$ che altrimenti genererebbero vincoli uguali nel CS.

7. Definisce in forma sparsa una matrice identità di ordine q, chiamata $A1$, che, se non ci sono informazioni di pre-riduzione, è una sottomatrice di A contenente i coefficienti del CS relativi a $\mathcal{D}$ e associati all'incognita $M0$ del problema di identificazione. $A1$, come le altre sottomatrici che compongono A, è memorizzata in forma sparsa, ossia memorizzando solo gli elementi non nulli insieme ai loro indici, per consentire di ridurre il tempo di calcolo. Infatti A contiene molti zeri.

8. Esegue un ciclo *for* che restituisce le matrici sparse $A2, A3$ e $A5$ che sono sottomatrici di A contenenti i coefficienti del CS relativi ad $\mathcal{E}$ e associati rispettivamente alle incognite $M0, Post$ e Pre del problema. La dimensione di $A2$ è $nv \times q$ dove q, come già visto, è il numero di posti della rete da identificare ed nv è il numero dei vincoli corrispondente ad $\mathcal{E}$. $A3$ e $A5$ hanno dimensioni $nv \times q \cdot n$, dove n è il numero di transizioni.

9. Se ci sono informazioni di pre-riduzione, esegue un ciclo *for* per ricavare la sottomatrice $A1$ che, come già detto, non è la matrice identità definita nel passo 7. La sua dimensione è $sigm\_D \times q$, dove $sigm\_D$ è il numero dei vincoli corrispondente ad $\mathcal{D}$. $A1$ ha lo stesso significato datole nel passo 7.

10. Esegue un ciclo *for* che restituisce le matrici sparse $A4$ e $A6$ che sono sottomatrici di A contenenti i coefficienti del CS relativi a $\mathcal{D}$ e associati rispettivamente alle incognite $Post$ e Pre del problema. Entrambe hanno dimensioni $sigm\_D \times q \cdot n$.

11. Costruisce la matrice A disponendo le 6 sottomatrici $A1, A2, A3, A4, A5$ e $A6$ nel modo seguente:

$$A = \left|\begin{array}{c|c|c} A2 & A3 & -A5 \\ \hline -A1 & -A4 & A6 \end{array}\right|$$

12. Costruisce il vettore colonna b concatenando un vettore colonna di dimensione nv i cui elementi sono tutti nulli con un vettore colonna di dimensione $sigm\_D$ i cui elementi sono tutti pari a 1.

13. Costruisce il vettore colonna di caratteri $ctype$ di dimensione $nv + sigm\_D$ i cui elementi sono tutti pari ad L, dove L indica che il senso del vincolo è '$\geq$' secondo la sintassi di GLPKMEX.

14. Definisce il numero di incognite del problema e lo memorizza nella variabile xm: $xm = q + 2(q \cdot n)$.

15. Costruisce il vettore lb di dimensione xm i cui elementi sono tutti nulli.

16. Costruisce il vettore colonna di caratteri $vartype$ di dimensione xm i cui elementi sono tutti pari a C, dove C indica che le variabili del problema sono continue secondo la sintassi di GLPKMEX.

Facciamo un esempio.

Esempio 6.4.1. *Consideriamo ancora una volta la rete in Figura 6.2. Si potrebbero inserire tutti i dati in ingresso nel file* `inputdata_E_D_pre_red2CS`*, ma dato che E e D sono stati salvati nel Workspace dalle precedenti computazioni, basta inserire nel file solo l'ingresso pre_red non settando E e D. Ricordiamo che E e D sono quelle date in Figura 6.4. Supponiamo di utilizzare le informazioni per la pre-riduzione date dalla rete, in base alle quali le transizioni t_1, t_2, t_3 e t_5 hanno un posto in ingresso e t_2 e t_3 sono in relazione di scelta libera. Allora pre_red avrà la seguente struttura:*

$$pre\_red = \begin{bmatrix} 1 & 0 \\ 2 & 0 \\ 3 & 0 \\ 5 & 0 \\ 2 & 3 \end{bmatrix}.$$

Eseguiamo il programma in Matlab e riportiamo nelle Figure 6.5-6.10 il risultato visualizzato nella finestra dei comandi. Per motivi di brevità possiamo evitare di riportare nelle figure i valori di $ctype, lb$ e $vartype$ perché questi, come già detto negli ultimi passi dell'algoritmo, sono rispettivamente pari a L, 0, e C in ogni istanza di GLPKMEX.

Osservando la Figura 6.5 si può vedere che se c'è pre-riduzione il numero di posti q vale 7 invece che essere pari alla cardinalità di $\mathcal{D}$, cioè 15. Da notare che in quest'esempio non sono presenti coppie $(\vec{\sigma}, \vec{\sigma} + \vec{t})$ uguali. Osservando le Figure 6.5-6.10 si può notare come sia difficile inquadrare la struttura densa della matrice A guardando la sua forma sparsa.

```
>> inputdata_E_D_pre_red2CS
>> [q,A,b,ctype,lb,vartype]=E_D_pre_red2CS(E,D,pre_red)

q =

      7


A =

    (1,1)        1
    (8,1)        1
   (15,1)        1
   (22,1)        1
   (29,1)        1
   (36,1)       -1
   (37,1)       -1
    (2,2)        1
    (9,2)        1
   (16,2)        1
   (23,2)        1
   (30,2)        1
   (38,2)       -1
   (39,2)       -1
   (40,2)       -1
    (3,3)        1
   (10,3)        1
   (17,3)        1
   (24,3)        1
   (31,3)        1
   (41,3)       -1
   (42,3)       -1
   (43,3)       -1
   (44,3)       -1
   (45,3)       -1
   (46,3)       -1
    (4,4)        1
   (11,4)        1
   (18,4)        1
   (25,4)        1
   (32,4)        1
   (47,4)       -1
    (5,5)        1
   (12,5)        1
   (19,5)        1
   (26,5)        1
   (33,5)        1
   (48,5)       -1
    (6,6)        1
   (13,6)        1
   (20,6)        1
```

Figura 6.5: Finestra dei comandi di Matlab.

(27,6)	1
(34,6)	1
(49,6)	-1
(7,7)	1
(14,7)	1
(21,7)	1
(28,7)	1
(35,7)	1
(50,7)	-1
(8,8)	1
(15,8)	1
(22,8)	1
(29,8)	1
(36,8)	-1
(37,8)	-1
(9,9)	1
(16,9)	1
(23,9)	1
(30,9)	1
(39,9)	-1
(40,9)	-1
(10,10)	1
(17,10)	1
(24,10)	1
(31,10)	1
(42,10)	-1
(43,10)	-1
(45,10)	-1
(46,10)	-1
(11,11)	1
(18,11)	1
(25,11)	1
(32,11)	1
(12,12)	1
(19,12)	1
(26,12)	1
(33,12)	1
(48,12)	-1
(13,13)	1
(20,13)	1
(27,13)	1
(34,13)	1
(49,13)	-1
(14,14)	1
(21,14)	1
(28,14)	1
(35,14)	1
(50,14)	-1
(22,15)	1
(37,15)	-1
(23,16)	1

Figura 6.6: Finestra dei comandi di Matlab.

(24,17)	1
(42,17)	-1
(45,17)	-1
(25,18)	1
(26,19)	1
(27,20)	1
(49,20)	-1
(28,21)	1
(29,22)	1
(30,23)	1
(40,23)	-1
(31,24)	1
(43,24)	-1
(46,24)	-1
(32,25)	1
(33,26)	1
(34,27)	1
(35,28)	1
(50,28)	-1
(1,43)	-1
(8,43)	-1
(15,43)	-1
(22,43)	-1
(29,43)	-2
(36,43)	2
(37,43)	2
(2,44)	-1
(9,44)	-1
(16,44)	-1
(23,44)	-1
(30,44)	-2
(39,44)	1
(40,44)	1
(3,45)	-1
(10,45)	-1
(17,45)	-1
(24,45)	-1
(31,45)	-2
(42,45)	1
(43,45)	1
(45,45)	1
(46,45)	1
(4,46)	-1
(11,46)	-1
(18,46)	-1
(25,46)	-1
(32,46)	-2
(5,47)	-1
(12,47)	-1
(19,47)	-1
(26,47)	-1

Figura 6.7: Finestra dei comandi di Matlab.

(33,47)	-2
(48,47)	1
(6,48)	-1
(13,48)	-1
(20,48)	-1
(27,48)	-1
(34,48)	-2
(49,48)	1
(7,49)	-1
(14,49)	-1
(21,49)	-1
(28,49)	-1
(35,49)	-2
(50,49)	1
(8,50)	-1
(22,50)	-1
(37,50)	1
(9,51)	-1
(23,51)	-1
(10,52)	-1
(24,52)	-1
(41,52)	1
(42,52)	2
(43,52)	1
(45,52)	1
(11,53)	-1
(25,53)	-1
(12,54)	-1
(26,54)	-1
(13,55)	-1
(27,55)	-1
(49,55)	1
(14,56)	-1
(28,56)	-1
(15,57)	-1
(29,57)	-1
(16,58)	-1
(30,58)	-1
(40,58)	1
(17,59)	-1
(31,59)	-1
(43,59)	1
(44,59)	1
(45,59)	1
(46,59)	2
(18,60)	-1
(32,60)	-1
(19,61)	-1
(33,61)	-1
(20,62)	-1
(34,62)	-1

Figura 6.8: Finestra dei comandi di Matlab.

```
    (21,63)       -1
    (35,63)       -1
    (50,63)        1
    (47,67)        1
    (48,68)        1
    (49,69)        1
    (50,70)        1
    (22,71)       -1
    (23,72)       -1
    (38,72)        1
    (39,72)        1
    (40,72)        1
    (24,73)       -1
    (25,74)       -1
    (26,75)       -1
    (27,76)       -1
    (28,77)       -1

b =

     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
```

Figura 6.9: Finestra dei comandi di Matlab.

```
0
0
0
0
0
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

Figura 6.10: Finestra dei comandi di Matlab.

Solitamente le uscite di `E_D_pre_red2CS` *non vengono mostrate sullo schermo visto che spesso sono degli array di grandi dimensioni (specialmente la matrice A) e occupano molto spazio nella finestra dei comandi. Inoltre non sono molto interessanti a parte il numero di posti q. Per gli stessi motivi, talvolta neanche le uscite di* `maxL_k_n2E_D` *vengono mostrate. Le uscite di tutti gli altri programmi devono essere mostrate perché sono importanti.* □

6.5 Estrazione della rete P/T soluzione del PPL

Il programma che estrae la rete P/T risolvendo il PPL è stato realizzato attraverso la funzione `CS_infoPN2PN` che ha la seguente sintassi:

```
[M0,Post,Pre,fm,ex]=CS_infoPN2PN
(q,A,b,ctype,lb,vartype,P_inv,P_inc,P_dec,
T_inv,T_inc,T_dec,state_mach,mark_graph,
ord_PN,Gmec,msglev,dual,lpsolver).
```

Gli argomenti di ingresso sono tanti (19), per cui è più comodo organizzarli per gruppi in base al tipo di argomento:

1. L'intero q, la matrice A e i vettori $b, ctype, lb$ e $vartype$, che sono tutti e 6

argomenti di uscita di `E_D_pre_red2CS`.

2. Le matrici $P\_inv, P\_inc, P\_dec, T\_inv, T\_inc$ e T_dec, i vettori riga $state\_mach$ e $mark\_graph$ e la variabile ord_PN, che rappresentano tutti informazioni strutturali sulla rete (se ne abbiamo) che aggiungono dei vincoli al CS.

3. Il vettore riga $Gmec$ che rappresenta un'informazione comportamentale sulla rete (se l'abbiamo), cioè legata alla marcatura iniziale, che aggiunge un vincolo al CS.

4. Le variabili $msglev, dual$ ed $lpsolver$ che sono alcuni dei parametri di controllo di GLPK, già accennati nella sezione 6.1.

Dunque, se non ci sono informazioni sulla rete gli ingressi da P_inv a $Gmec$, secondo la lista della sintassi, sono vuoti. Altrimenti:

- P_inv, P_inc e P_dec sono matrici aventi un numero di colonne pari a q e le cui righe contengono rispettivamente i P-invarianti, i P-vettori crescenti e i P-vettori decrescenti.

- T_inv, T_inc e T_dec sono matrici aventi un numero di colonne pari a n e le cui righe contengono rispettivamente i T-invarianti, i T-vettori crescenti e i T-vettori decrescenti.

- $state\_mach$ ha tutti i suoi elementi pari ad 1 se la rete complessiva è una macchina di stato e alcuni elementi pari ad 1 ed altri pari a 0 se solamente una o più sottoreti sono macchine di stato. In quest'ultimo caso un elemento di indice i pari ad 1 corrisponde ad una transizione t_i facente parte di qualche macchina di stato, mentre un elemento nullo corrisponde ad una transizione non contenuta in alcuna macchina di stato.

- $mark\_graph$ ha la stessa struttura di $state\_mach$ nel caso in cui la rete complessiva o una o più sottoreti siano grafi marcati.

- ord_PN vale 1 se la rete è ordinaria e 0 se non lo è.

- $Gmec$ introduce nel CS un vincolo del tipo $\vec{w} \cdot M_0 \leq c$, chiamato GMEC. La sua dimensione è $q + 1$ dove i primi q elementi rappresentano il vettore $\vec{w}$ mentre l'ultimo elemento è la costante c.

Adesso vediamo qual è il significato dei parametri di GLPK.

- $msglev$ permette di scegliere il tipo di messaggio di uscita del risolutore (solo per il metodo del simplesso) e dev'essere settato a 1 se si desidera avere informazioni da GLPK solo in caso di errore (messaggi di errore), a 2 se si vuole un'uscita normale o a 3 se si vuole un'uscita completa. La differenza tra i messaggi di uscita verrà commentata quando mostrerò la finestra dei comandi di Matlab.

- $dual$ dev'essere settato a 0 se non si intende utilizzare il metodo del simplesso duale o a 1 se lo si vuole utilizzare quando la soluzione di base iniziale del simplesso revisionato è ammissibile duale.

- $lpsolver$ dev'essere settato a 1 se si intende risolvere il PPL col metodo del simplesso revisionato o a 2 se si vuole risolverlo col metodo del punto interno, che sono i due metodi implementati da GLPK.

Gli altri parametri definiti in GLPK sono meno interessanti e non è necessario citarli dato che non verranno modificati rispetto ai loro valori di default.

Gli argomenti di uscita sono:

- Il vettore colonna marcatura iniziale $M0$ di dimensione q e le matrici $Post$ e Pre di dimensioni $q \times n$ che rappresentano la rete identificata.

- La variabile fm che contiene il valore ottimo della funzione obiettivo.

- Una struttura dati, chiamata ex e coincidente con l'argomento $extra$ di GLPKMEX, contenente il numero di variabili duali e il numero di costi ridotti del problema di programmazione, il tempo (in secondi) impiegato per risolverlo e la memoria (in bytes) richiesta.

L'algoritmo implementato dalla funzione esegue i seguenti passi:

1. Se ci sono informazioni sui P-invarianti esegue un ciclo *for* per aggiungere dei vincoli al CS.

2. Se ci sono informazioni sui P-vettori crescenti esegue un ciclo *for* per aggiungere dei vincoli al CS.

3. Se ci sono informazioni sui P-vettori decrescenti esegue un ciclo *for* per aggiungere dei vincoli al CS.

4. Se ci sono informazioni sui T-invarianti esegue un ciclo *for* per aggiungere dei vincoli al CS.

5. Se ci sono informazioni sui T-vettori crescenti esegue un ciclo *for* per aggiungere dei vincoli al CS.

6. Se ci sono informazioni sui T-vettori decrescenti esegue un ciclo *for* per aggiungere dei vincoli al CS.

7. Se è presente l'informazione che la rete o qualche sua sottorete sia una macchina di stato aggiunge dei vincoli al CS.

8. Se è presente l'informazione che la rete o qualche sua sottorete sia un grafo marcato aggiunge dei vincoli al CS.

9. Se è definita una GMEC aggiunge un vincolo al CS.

10. Se è presente l'informazione che la rete sia ordinaria, ma non è noto se essa sia una macchina di stato o un grafo marcato, impone a 1 il limite superiore dei valori degli elementi di $Post$ e Pre. Il limite superiore degli elementi di M_0 è sempre infinito. Se la rete è una macchina di stato o un grafo marcato è anche ordinaria, quindi tale operazione è ridondante e non viene effettuata.

11. Chiama la funzione `glpkmex` per risolvere il PPL.

12. Estrae M_0 dall'argomento di uscita $xmin$ di `glpkmex`.

13. Esegue un ciclo *for* per estrarre il resto della soluzione, cioè $Post$ e Pre, sempre da $xmin$.

14. Assegna ad fm il valore dell'argomento di uscita $fmin$ di `glpkmex`.

15. Se la soluzione trovata non è intera esegue un ciclo *while* per ricavarne una intera equivalente.

16. Assegna ad ex il valore dell'argomento di uscita $extra$ di `glpkmex`.

Facciamo un esempio.

Esempio 6.5.1. *Consideriamo ancora una volta la rete in Figura 6.2. Anche qui si possono inserire i dati in ingresso nel file* `inputdata_CS_infoPN2PN`*, ma* $q, A, b, ctype, lb$ *e* $vartype$ *sono nel Workspace per cui non li dobbiamo settare nel file. Ricordiamo quali sono questi ingressi:* q *è pari a 7,* A *è una matrice sparsa di dimensioni* 50×77*,* b *è un vettore colonna di dimensione 50 e sono riportati nelle*

Figure 6.5-6.10. I vettori $ctype, lb$ e $vartype$ hanno rispettivamente dimensione 50, 77 e 77 e valori pari a L, 0, e C.

Supponiamo di non avere informazioni sulla rete perciò definiamo come vuoti tutti gli argomenti da P_inv a $Gmec$. Poniamo $msglev$ pari a 3 per avere un'uscita completa, $dual$ a 0 per non applicare la programmazione duale ed $lpsolver$ a 1 per risolvere il problema col metodo del simplesso revisionato.

Eseguiamo il programma in Matlab e riportiamo nelle Figure 6.11-6.12 il risultato visualizzato nella finestra dei comandi.

Subito dopo la chiamata alla funzione `CS_infoPN2PN` *compaiono alcune informazioni sulla soluzione.*

La chiamata alla routine *'lpx_write_cpxlp' di GLPK consente di scrivere i dati del problema nel file 'outpb.lp' della directory corrente. La chiamata alla* routine *'lpx_simplex' consente di chiamare il risolutore per risolvere il problema e di immagazzinare una soluzione di base ottenuta e altre informazioni sul problema in questione che vedremo tra poco. Il risolutore utilizza un pre-risolutore integrato che trasforma il problema originale (in questo caso 50×77) in un problema equivalente (in questo caso 49×59) che può essere più facile da risolvere col metodo del simplesso di quello originale e poi effettua un'altra trasformazione per ottenere la soluzione del problema originale. La chiamata a 'lpx_adv_basis' permette di costruire una base iniziale avanzata del problema. Le altre informazioni a cui ho accennato poc'anzi sono riportate nelle righe seguenti la chiamata a 'lpx_adv_basis'. Ogni riga contiene nell'ordine:*

- *Il numero dell'iterazione corrente del simplesso (se questo è preceduto da un asterisco il risolutore sta cercando una soluzione ottimale altrimenti sta cercando una soluzione ammissibile primaria).*
- *Il valore corrente della funzione obiettivo (preceduto da* objval*).*
- *La somma corrente delle inammissibilità primarie (preceduta da* infeas *e seguita dal numero corrente di variabili di base fisse).*

Per il caso dell'esempio abbiamo le seguenti informazioni: alla 12a iterazione il simplesso trova la soluzione ammissibile primaria e la funzione obiettivo vale 12; Alla 18a iterazione trova la soluzione ottimale con la funzione obiettivo che vale sempre 12. Dopo l'ultima riga compare il messaggio OPTIMAL SOLUTION FOUND che attesta l'ottimalità della soluzione trovata.

```
>>  inputdata_CS_infoPN2PN
>>  [M0,Post,Pre,fm,ex]=CS_infoPN2PN(q,A,b,ctype,lb,vartype,P_inv,P_inc,P_dec,T_inv,T_inc,
T_dec,state_mach,mark_graph,ord_PN,Gmec,msglev,dual,lpsolver)
lpx_write_cpxlp: writing problem data to `outpb.lp'...
lpx_simplex: original LP has 50 rows, 77 columns, 211 non-zeros
lpx_simplex: presolved LP has 49 rows, 59 columns, 209 non-zeros
lpx_adv_basis: size of triangular part = 49
       0:    objval =  4.000000000e+000      infeas =  1.000000000e+000 (0)
      12:    objval =  1.200000000e+001      infeas =  0.000000000e+000 (0)
*     12:    objval =  1.200000000e+001      infeas =  0.000000000e+000 (0)
*     18:    objval =  1.200000000e+001      infeas =  0.000000000e+000 (0)
OPTIMAL SOLUTION FOUND

M0  =

     1
     0
     0
     0
     0
     0
     0


Post  =

     0   0   1   0   0
     0   1   0   0   0
     1   0   0   0   0
     0   0   0   0   0
     0   0   0   0   0
     0   0   0   0   0
     0   0   0   0   0


Pre  =

     1   0   0   0   0
     0   0   0   0   1
     0   1   1   0   0
     0   0   0   1   0
     0   0   0   1   0
     0   0   0   1   0
     0   0   0   1   0


fm  =

     12
```

Figura 6.11: Finestra dei comandi di Matlab.

```
ex =

      lambda: [50x1 double]
    redcosts: [77x1 double]
        time: 0
      memory: 96100
```

Figura 6.12: Finestra dei comandi di Matlab.

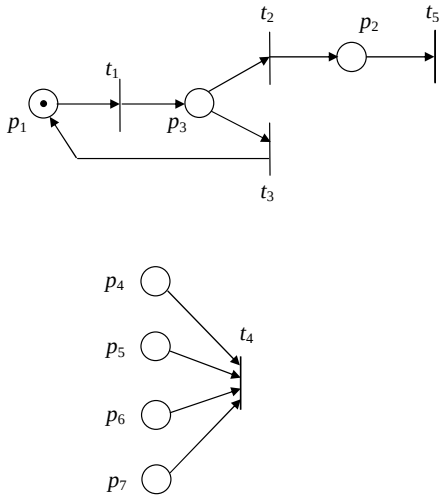

Figura 6.13: Il sistema di rete di Petri identificato.

```
>>  inputdata_CS_infoPN2PN
>>  [M0,Post,Pre,fm,ex]=CS_infoPN2PN(q,A,b,ctype,lb,vartype,P_inv,P_inc,P_dec,T_inv,T_inc,
T_dec,state_mach,mark_graph,ord_PN,Gmec,msglev,dual,lpsolver)
lpx_write_cpxlp: writing problem data to `outpb.lp'...
lpx_interior: original LP problem has 50 rows and 77 columns
lpx_interior: transformed LP problem has 50 rows and 127 columns
lpx_interior: A has 261 non-zeros
lpx_interior: S has 214 non-zeros (upper triangle)
lpx_interior: minimal degree ordering...
lpx_interior: computing Cholesky factorization...
lpx_interior: U has 214 non-zeros
lpx_interior: guessing initial point...
Optimization begins...
   0:  obj =  7.000273647e+001;  rpi = 1.3e+000;  rdi = 9.4e-001;  gap = 9.8e-001
   1:  obj =  3.778229397e+001;  rpi =  3.5e-001;  rdi = 1.7e-001;  gap = 7.3e-001
   2:  obj =  1.855758456e+001;  rpi =  7.1e-002;  rdi = 3.5e-002;  gap = 3.7e-001
   3:  obj =  1.271012017e+001;  rpi =  7.6e-003;  rdi = 3.6e-003;  gap = 5.7e-002
   4:  obj =  1.207112423e+001;  rpi =  7.6e-004;  rdi = 3.6e-004;  gap = 6.0e-003
   5:  obj =  1.200711254e+001;  rpi =  7.6e-005;  rdi = 3.6e-005;  gap = 6.0e-004
   6:  obj =  1.200071125e+001;  rpi =  7.6e-006;  rdi = 3.6e-006;  gap = 6.0e-005
   7:  obj =  1.200007113e+001;  rpi =  7.6e-007;  rdi = 3.6e-007;  gap = 6.0e-006
   8:  obj =  1.200000711e+001;  rpi =  7.6e-008;  rdi = 3.6e-008;  gap = 6.0e-007
   9:  obj =  1.200000071e+001;  rpi =  7.6e-009;  rdi = 3.6e-009;  gap = 6.0e-008
  10:  obj =  1.200000007e+001;  rpi =  7.6e-010;  rdi = 3.6e-010;  gap = 6.0e-009
OPTIMAL SOLUTION FOUND

M0  =

     1
     0
     0
     0
     0
     0
     0


Post  =

     0    0    1    0    0
     0    1    0    0    0
     1    0    0    0    0
     0    0    0    0    0
     0    0    0    0    0
     0    0    0    0    0
     0    0    0    0    0
```

Figura 6.14: Finestra dei comandi di Matlab.

```
Pre =

     1     0     0     0     0
     0     0     0     0     1
     0     1     1     0     0
     0     0     0     1     0
     0     0     0     1     0
     0     0     0     1     0
     0     0     0     1     0


fm =

    12


ex =

      lambda: [50x1 double]
    redcosts: [77x1 double]
        time: 0
      memory: 96100
```

Figura 6.15: Finestra dei comandi di Matlab.

Abbiamo identificato una rete con 7 posti, quindi facilmente rappresentabile a mano, che riportiamo in Figura 6.13. La rete identificata è una rete disconnessa in cui la parte contenente i posti p_1, p_2 e p_3 è sufficiente per generare il linguaggio massimale osservato $\{t_1t_2t_5, t_1t_3t_1\}$. Quindi l'altra parte di rete contiene dei posti ridondanti che possono essere eliminati applicando una post-riduzione. Vedremo nella prossima sezione cosa otteniamo. Da notare che la transizione t_4 non verrà mai abilitata dalla rete e infatti non compare nel linguaggio.

La struttura dati ex per l'esempio dice che ci sono 50 variabili duali e 77 costi ridotti, che il problema è stato risolto in 0 secondi (in realtà non è proprio 0, si parla di frazioni di secondo, però il dato 'time' fornisce solo valori interi) e che la memoria utilizzata è 96.1 kB.

Se avessimo posto $msglev$ a 2 per ottenere un'uscita normale, non avremmo visualizzato sullo schermo il messaggio OPTIMAL SOLUTION FOUND, le due righe che iniziano con $lpx\_simplex$ e la riga che inizia con lpx_adv_basis. Con $msglev$ a 1 non avremmo visualizzato nemmeno la lista delle iterazioni significative.

Ponendo $dual$ a 1 avremmo avuto quasi le stesse uscite al variare di $msglev$. Infatti le differenze sarebbero state le seguenti: prima del numero dell'iterazione corrente del simplesso avremmo visualizzato delle barrette verticali le quali indi-

cano che il problema viene risolto col simplesso duale; la somma corrente delle inammissibilità primarie sarebbe cambiata.

Ponendo $lpsolver$ a 2 per risolvere il problema col metodo del punto interno, otteniamo il risultato, visualizzato nella finestra dei comandi, riportato nelle Figure 6.14-6.15. Tale metodo è più fallace del metodo del simplesso come dimostreranno gli esempi del capitolo 7. In breve diciamo che nell'uscita del risolutore mostrata nelle Figure 6.14-6.15 la routine *$lpx\_interior$ chiama il risolutore per risolvere il problema di programmazione e visualizza l'informazione su ogni iterazione del punto interno.*

La soluzione ottenuta con le diverse tecniche di programmazione lineare è la stessa, come dovrebbe risultare sempre. Il risolutore del simplesso ha trovato la soluzione ottimale in 109 millesimi di secondo mentre il risolutore del punto interno ha impiegato più tempo, esattamente 172 millesimi. Comunque questa non è una regola generale, anzi, per problemi più complessi di questo ci si aspetta che il risolutore del punto interno sia più veloce. I tempi esatti sono stati calcolati mediante la sequenza di comandi

```
   tic;[M0,Post,Pre,fm,ex]=CS_infoPN2PN
(q,A,b,ctype,lb,vartype,P_inv,P_inc,P_dec,
 T_inv,T_inc,T_dec,state_mach,mark_graph,
   ord_PN,Gmec,msglev,dual,lpsolver);toc
```

che visualizza il numero di secondi richiesto per l'operazione con una precisione al millesimo di secondo. □

6.6 Estrazione della rete P/T ridotta

Il programma che estrae la rete P/T ridotta applicando una post-riduzione è stato realizzato attraverso la funzione `PN_D2post_red_PN` che ha la seguente sintassi:

```
[red_M0,red_Post,red_Pre]=PN_D2post_red_PN
          (M0,Post,Pre,D,msglev).
```

Gli argomenti di ingresso sono il vettore colonna $M0$ e le matrici $Post$ e Pre, che sono argomenti di uscita di `CS_infoPN2PN`, la matrice D, che è argomento

di uscita di `maxL_n_k2E_D` e argomento di ingresso di `E_D_pre_red2CS`, e la variabile $msglev$, argomento di ingresso di `CS_infoPN2PN`. Ricordiamo che D serve per calcolare i vettori caratteristici ed $msglev$ è utile per decidere come visualizzare l'uscita del risolutore.

Gli argomenti di uscita sono il vettore colonna marcatura iniziale red_M0 di dimensione p e le matrici red_Post e red_Pre di dimensioni $p \times n$ che rappresentano la rete ridotta. Ho scelto questi nomi per esplicitare la differenza tra la rete soluzione del PPL e la rete ridotta. Poi il nome diverso è utile quando visualizzeremo sullo schermo entrambe le reti, come risultato del programma che esegue l'intera procedura.

L'algoritmo implementato dalla funzione esegue i seguenti passi:

1. Esegue un ciclo *for* per calcolare i vettori caratteristici, i quali contengono i coefficienti del PPI che ricava il minimo *hitting set*. Tali vettori sono memorizzati come vettori riga e riuniti in una matrice sparsa y.
2. Setta i parametri di GLPKMEX in ingresso alla funzione `glpkmex`. Rispetto alla sintassi standard non ho inserito l'ingresso $lpsolver$ perché questo viene ignorato da GLPK nel caso in cui si debba risolvere un problema di programmazione misto-intera (MIP) o intera.
3. Chiama la funzione `glpkmex` per risolvere il PPI.
4. Esegue un ciclo *for* per estrarre la rete ridotta dall'argomento di uscita $xmin$ di `glpkmex`.

Facciamo ricorso ancora una volta ad un esempio chiarificatore.

Esempio 6.6.1. *Consideriamo sempre la rete in Figura 6.2. I dati in ingresso sono salvati nel Workspace e non è necessario assegnarli nel relativo file* `inputdata_PN_D2post_red_PN`*, a meno che non si voglia modificare il valore di* $msglev$*. Lasciamo* $msglev$ *a 3 per avere un'uscita completa. Ricordiamo quali sono gli altri ingressi: il vettore colonna* $M0$ *di dimensione 7 e le matrici* $Post$ *e* Pre *di dimensioni* 7×5 *riportati nelle Figure 6.11-6.12 e 6.14-6.15 e la matrice* D *riportata in Figura 6.4.*

Eseguiamo il programma in Matlab e riportiamo nella Figura 6.16 il risultato visualizzato nella finestra dei comandi.

In breve, nella Figura 6.16 la routine *'lpx_integer' chiama il risolutore per risolvere il PPI. In ingresso a questa* routine *si ottiene una soluzione ottimale (OP-*

```
>>  [red_M0,red_Post,red_Pre]=PN_D2post_red_PN(M0,Post,Pre,D,msglev)
lpx_simplex: original LP has 15 rows, 7 columns, 27 non-zeros
lpx_simplex: presolved LP has 4 rows, 4 columns, 16 non-zeros
lpx_adv_basis: size of triangular part = 4
        0:    objval =  3.000000000e+000    infeas =  1.000000000e+000  (0)
        1:    objval =  4.000000000e+000    infeas =  0.000000000e+000  (0)
*       1:    objval =  4.000000000e+000    infeas =  0.000000000e+000  (0)
*       2:    objval =  4.000000000e+000    infeas =  0.000000000e+000  (0)
OPTIMAL SOLUTION FOUND
Integer optimization begins...
Objective function is integral
+       2:  mip =           not found yet >=                          -inf        (1; 0)
+       2:  mip =  4.000000000e+000 >=  4.000000000e+000   0.0% (1; 0)
+       2:  mip =  4.000000000e+000 >=          tree is empty   0.0% (0; 1)
INTEGER OPTIMAL SOLUTION FOUND

red_M0  =

       1
       0
       0
       0


red_Post  =

       0    0    1    0    0
       0    1    0    0    0
       1    0    0    0    0
       0    0    0    0    0


red_Pre  =

       1    0    0    0    0
       0    0    0    0    1
       0    1    1    0    0
       0    0    0    1    0
```

Figura 6.16: Finestra dei comandi di Matlab.

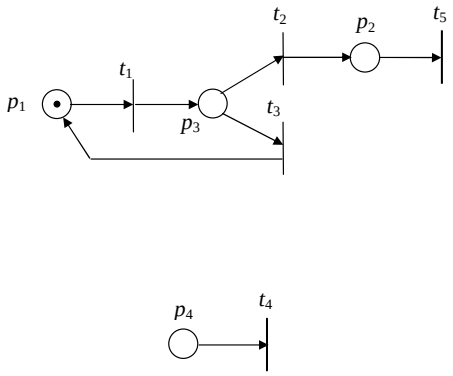

Figura 6.17: Il sistema di rete di Petri ridotto.

TIMAL SOLUTION FOUND) per il problema rilassato, per mezzo del risolutore basato sul simplesso. Quindi il metodo del branch-and-bound *ricava la soluzione ottimale finale(INTEGER OPTIMAL SOLUTION FOUND). Se* $msglev = 2$ *non compaiono le prime tre righe e i messaggi in stampatello mentre se* $msglev = 1$ *compare solo la seguente riga:*
$+ \quad 2: \quad mip = 4.000000000e + 000 >= \qquad tree\ is\ empty \quad 0.0\%\ (0;1).$

La rete ridotta ha 4 posti ed è rappresentata in Figura 6.17. Anch'essa, come la rete soluzione del PPL, è disconnessa. La post-riduzione ha eliminato i posti ridondanti p_5, p_6 *e* p_7*. Da notare che la parte della rete contenente i posti* p_1, p_2 *e* p_3 *ha una struttura simile alla sottorete* $p_1, p_2, p_3, t_1, t_2, t_3, t_4$ *della rete di partenza in Figura 6.2.* □

6.7 Esecuzione dell'intera procedura

Il programma che esegue l'intera procedura richiamando le funzioni descritte in questo capitolo è stato realizzato attraverso la funzione `identif_proced` che ha la seguente sintassi:

```
[red_M0,red_Post,red_Pre,maxL2]=identif_proced
     (maxL,n,k,pre_red,T_inv,T_inc,T_dec,
   state_mach,ord_PN,msglev,dual,lpsolver).
```

Gli argomenti di ingresso sono i seguenti:

- La matrice $maxL$ e l'intero n, che sono argomenti di uscita di `PN_k2maxL_n` e argomenti di ingresso di `maxL_n_k2E_D`.
- L'intero k, che è argomento di ingresso di `PN_k2maxL_n` e di `maxL_n_k2E_D`.
- La matrice pre_red, che è argomento di ingresso di `E_D_pre_red2CS`.
- Le matrici T_inv, T_inc e T_dec, il vettore riga $state\_mach$ e le variabili $ord\_PN, msglev, dual$ e $lpsolver$, che sono tutti argomenti di ingresso di `CS_infoPN2PN`.

Non ho inserito come ingressi le matrici E e D dato che si ricavano da `maxL_k_n2E_D`, stesso discorso per $q, A, b, ctype, lb$ e $vartype$, che sono uscite di `E_D_pre_red2CS`, e per la rete soluzione del PPL che si ricava come uscita di `CS_infoPN2PN`. Non ho inserito gli ingressi P_inv, P_inc, P_dec, $mark\_graph$ e $Gmec$ di `CS_infoPN2PN`, che hanno dimensioni dipendenti dal numero di posti q della rete soluzione del PPL, in quanto, nonostante non vengano ricavati da alcuna funzione, non si possono definire all'inizio dell'intera procedura perché ancora non è noto q, che sarà noto solo dopo la chiamata a `E_D_pre_red2CS`.

Gli argomenti di uscita sono il vettore colonna marcatura iniziale red_M0 e le matrici red_Post e red_Pre, che sono argomenti di uscita di `PN_D2post_red_PN`, e la matrice $maxL2$ che rappresenta il linguaggio L_k generato dalla rete ridotta. L'uscita $maxL2$ corrisponde all'uscita $maxL$ di `PN_k2maxL_n`. Nell'illustrazione dell'algoritmo spiegherò il motivo del cambiamento di nome.

Dunque, l'algoritmo implementato dalla funzione esegue i seguenti passi:

1. Richiama la funzione `maxL_k_n2E_D` che costruisce gli insiemi $\mathcal{E}$ e $\mathcal{D}$.
2. Richiama la funzione `E_D_pre_red2CS` che costruisce il CS.
3. Richiama il file dati `inputdata_CS_infoPN2PN` per settare gli ingressi della funzione `CS_infoPN2PN` dipendenti dal numero di posti che non possono essere dati in ingresso a `identif_proced`. Quest'operazione va comunque effettuata dall'utente prima di lanciare il programma `identif_proced`, definendo, in funzione di q, gli ingressi che dipendono dal numero di posti. Infatti la chiamata al file

`inputdata_CS_infoPN2PN` dentro il programma dev'essere trasparente all'utente che non può intervenire sul programma in esecuzione per settare gli ingressi.

4. Richiama la funzione `CS_infoPN2PN` che estrae la rete P/T soluzione del PPL e assegna la soluzione ricavata $M0, Post$ e Pre rispettivamente a $M0\_PPL, Post\_PPL$ e Pre_PPL. I tre assegnamenti sono necessari per mostrare sullo schermo la soluzione corretta visto che $M0, Post$ e Pre saranno sovrascritti da istruzioni successive altrettanto necessarie.

5. Richiama `PN_k2maxL_n` per verificare che la rete ottenuta generi il linguaggio di partenza $maxL$ e assegna $maxL$ a $maxL1$ perché $maxL$ verrà sovrascritto in seguito.

6. Richiama la funzione `PN_D2post_red_PN` che estrae la rete P/T ridotta soluzione finale della procedura completa e assegna red_M0, red_Post e red_Pre rispettivamente a $M0, Post$ e Pre per poter richiamare di nuovo `PN_k2maxL_n`.

7. Richiama `PN_k2maxL_n` per verificare che la rete ridotta generi il linguaggio di partenza $maxL$ e assegna $maxL$ a $maxL2$. Tale assegnamento, anche se non necessario, è stato fatto solo per distinguere, usando nomi diversi, il linguaggio osservato da quello generato dalla rete ridotta finale.

Come sempre ricorriamo ad un esempio per chiarire quanto appena illustrato.

Esempio 6.7.1. *Consideriamo sempre la rete in Figura 6.2. Nel file dati* `inputdata_identif_proced` *vengono settati tutti gli ingressi. Abbiamo:*

$$maxL = \begin{bmatrix} 1 & 2 & 5 \\ 1 & 3 & 1 \end{bmatrix}, \quad n = 5 \quad e \quad k = 3.$$

$maxL$ ed n sono stati ricavati da `PN_k2maxL_n`*, dato che era nota una struttura di rete da testare.*

Supponiamo ancora che si possa effettuare una pre-riduzione, quindi

$$pre\_red = \begin{bmatrix} 1 & 0 \\ 2 & 0 \\ 3 & 0 \\ 5 & 0 \\ 2 & 3 \end{bmatrix}.$$

Inoltre supponiamo sempre che non si abbiano informazioni strutturali sulla rete o GMEC e poniamo $msglev$ *a 3,* $dual$ *a 0 e* $lpsolver$ *a 1.*

Eseguiamo il programma in Matlab e riportiamo nelle Figure 6.18-6.20 il risultato visualizzato nella finestra dei comandi.

Illustriamo ciò che compare nella finestra dei comandi.

$size\_D$ *indica le dimensioni della matrice* D *delle condizioni di disabilitazione, poi compaiono il numero di posti* q *e* $size\_A$*, che indica le dimensioni della matrice* A *dei coefficienti dei vincoli. Ho voluto mostrare questi tre dati per avere delle indicazioni sulla complessità del problema. Inoltre* $size\_D$ *e* q *sono utili anche per un confronto diretto col numero di posti della rete ridotta.*

Seguono le informazioni sulla soluzione del PPL, le stesse di Figura 6.11, e le 5 uscite della funzione `CS_infoPN2PN` *che estrae la rete P/T, da* $M0\_PPL$ *a* ex*, coi nuovi nomi per* $M0, Post$ *e* Pre*. Poi compaiono il linguaggio* $maxL1$*, generato da* $M0\_PPL, Post\_PPL$ *e* Pre_PPL*, preceduto dalle sue dimensioni. Seguono poi le informazioni relative al PPI, riportate anche in Figura 6.16. Infine compaiono la rete ridotta* red_M0, red_Post *e* red_Pre *e il linguaggio* $maxL2$ *da essa generato.*

Si può notare che i due linguaggi $maxL1$ *e* $maxL2$ *sono uguali a quello di partenza per cui la procedura completa è stata eseguita con successo per* $k = 3$*.*

In pratica, sullo schermo vengono visualizzate le informazioni più importanti ai fini della correttezza e dell'efficienza della procedura.

```
>> inputdata_identif_proced
>> [red_M0,red_Post,red_Pre,maxL2]=identif_proced
(maxL,n,k,pre_red,T_inv,T_inc,T_dec,state_mach,ord_PN,msglev,dual,lpsolver)

size_D  =

     15     3


q  =

      7


size_A  =

     50    77

lpx_write_cpxlp: writing problem data to `outpb.lp'...
lpx_simplex: original LP has 50 rows, 77 columns, 211 non-zeros
lpx_simplex: presolved LP has 49 rows, 59 columns, 209 non-zeros
lpx_adv_basis: size of triangular part = 49
       0:    objval =  4.000000000e+000     infeas =  1.000000000e+000  (0)
      12:    objval =  1.200000000e+001     infeas =  0.000000000e+000  (0)
*     12:    objval =  1.200000000e+001     infeas =  0.000000000e+000  (0)
*     18:    objval =  1.200000000e+001     infeas =  0.000000000e+000  (0)
OPTIMAL SOLUTION FOUND

M0_PPL  =

      1
      0
      0
      0
      0
      0
      0


Post_PPL  =

      0     0     1     0     0
      0     1     0     0     0
      1     0     0     0     0
      0     0     0     0     0
      0     0     0     0     0
      0     0     0     0     0
      0     0     0     0     0
```

Figura 6.18: Finestra dei comandi di Matlab.

```
Pre_PPL  =

     1     0     0     0     0
     0     0     0     0     1
     0     1     1     0     0
     0     0     0     1     0
     0     0     0     1     0
     0     0     0     1     0
     0     0     0     1     0


fm  =

    12


ex  =

      lambda: [50x1 double]
    redcosts: [77x1 double]
        time: 0
      memory: 96100


size_maxL1  =

     2     3


maxL1  =

     1     2     5
     1     3     1

lpx_simplex: original LP has 15 rows, 7 columns, 27 non-zeros
lpx_simplex: presolved LP has 4 rows, 4 columns, 16 non-zeros
lpx_adv_basis: size of triangular part = 4
        0:   objval =  3.000000000e+000   infeas =  1.000000000e+000  (0)
        1:   objval =  4.000000000e+000   infeas =  0.000000000e+000  (0)
*       1:   objval =  4.000000000e+000   infeas =  0.000000000e+000  (0)
*       2:   objval =  4.000000000e+000   infeas =  0.000000000e+000  (0)
OPTIMAL SOLUTION FOUND
Integer optimization begins...
Objective function is integral
+       2:  mip =           not found yet >=                        -inf       (1; 0)
+       2:  mip =  4.000000000e+000 >=  4.000000000e+000   0.0% (1; 0)
+       2:  mip =  4.000000000e+000 >=          tree is empty   0.0% (0; 1)
INTEGER OPTIMAL SOLUTION FOUND
```

Figura 6.19: Finestra dei comandi di Matlab.

```
red_M0 =

     1
     0
     0
     0


red_Post =

     0   0   1   0   0
     0   1   0   0   0
     1   0   0   0   0
     0   0   0   0   0


red_Pre =

     1   0   0   0   0
     0   0   0   0   1
     0   1   1   0   0
     0   0   0   1   0


maxL2 =

     1   2   5
     1   3   1
```

Figura 6.20: Finestra dei comandi di Matlab.

Capitolo 7

Testing di reti P/T: esempi e risultati

7.1 Presentazione generale

Una volta realizzati i programmi ho testato la loro funzionalità su varie reti, con differenti caratteristiche e al variare di determinati parametri come l'intero k, che in questo capitolo considero sempre uguale alla lunghezza della stringa più lunga del linguaggio, il numero di gettoni della marcatura iniziale, i pesi degli archi e le dimensioni della rete da testare.

Quindi, ho una rete da testare di cui si conoscono la struttura e la marcatura iniziale e da cui si può ricavare un prefisso finito del suo linguaggio al variare dei parametri. Voglio applicare la procedura di identificazione descritta nei capitoli precedenti per vedere se si riesce ad ottenere una rete, probabilmente diversa da quella che sto testando, che genera lo stesso linguaggio. E poi sperare di ottenere una rete di piccole dimensioni che genera sempre lo stesso linguaggio, magari simile alla rete di partenza, con la tecnica di post-riduzione.

Ho applicato le diverse tecniche di programmazione lineare implementate da GLPK: il metodo del simplesso revisionato, il metodo del simplesso duale e il metodo del punto interno. Qualche esempio l'ho già trattato, anche se non completamente, nei capitoli 4, 5 e 6. In questo capitolo presenterò 4 esempi, dei quali il primo è relativo ad una rete qualsiasi e gli altri 3 a reti che vengono usate per rappresentare sistemi tratti da diversi campi applicativi (sistemi operativi e protocolli di comunicazione).

Ho effettuato i test su un PC Fujitsu Siemens con un processore Pentium M da 2

GHz e una memoria RAM da 1 GB.

I dati più interessanti e più importanti, ai fini del buon funzionamento e dell'efficienza dei programmi, della correttezza della procedura e dell'utilizzabilità di Matlab e GLPK, che ho ricavato dai test sono i seguenti:

- Limiti di affidabilità del risolutore di GLPK basato sul metodo del simplesso revisionato.
- Limiti di affidabilità del risolutore di GLPK basato sul metodo del punto interno.
- Limiti di affidabilità del risolutore di GLPK basato sul metodo del *branch-and-bound*.
- Linguaggio L_k generato dalla rete soluzione del problema di programmazione lineare (PPL), che chiamo $maxL1$ per distinguerlo dal linguaggio osservato che chiamo $maxL$; comunque $maxL$ e $maxL1$ coincidono se la procedura è corretta. Il prefisso max indica che si tratta esattamente del linguaggio massimale, cioè il linguaggio delle stringhe di massima lunghezza: infatti è questo il linguaggio matriciale in ingresso alla procedura, come abbiamo visto nel capitolo 6.
- Linguaggio L_k generato dalla rete ridotta con una post-riduzione, che chiamo $maxL2$ per distinguerlo sia da $maxL$ che da $maxL1$; i tre linguaggi coincidono se la procedura è corretta.
- Cardinalità dell'insieme $\mathcal{D}$ ($card\_D$).
- Dimensioni della matrice A dei coefficienti dei vincoli ($size\_A$); ricordiamo che il numero di righe è il numero di vincoli e il numero di colonne è il numero di variabili.
- Numero di posti della rete soluzione del PPL (q).
- Numero di posti della rete ridotta con una post-riduzione (p).
- Tempo di risoluzione del PPL (t_{PPL}).
- Tempo di esecuzione dell'intera procedura (t_{proc}), che comprende anche i tempi necessari per calcolare $maxL1$ e $maxL2$.
- Limiti di applicabilità della procedura dal punto di vista della memoria utilizzata da Matlab.

Altri dati ricavabili dai test, come la matrice E delle condizioni di abilitazione, gli elementi delle matrici D ed A, il valore ottimo della funzione obiettivo, il tempo di risoluzione del PPI che calcola gli *hitting set* e i tempi di esecuzione dei singoli programmi, rivestono meno importanza e sono meno utili dei dati elencati sopra. Quindi non ne prenderò nota e non farò considerazioni su di essi, anche per non appesantire troppo la trattazione, tranne fare qualche riflessione sui tempi di esecuzione dei singoli programmi.

Per comodità di lettura riporto dentro tabelle la cardinalità di $maxL$ ($card\_maxL$) e D, le dimensioni di A, il numero di posti e i tempi di computazione. Non riporto per esteso i linguaggi per ovvii motivi di spazio, ma mi limito a segnalare se $maxL1$ e $maxL2$ sono uguali o meno al linguaggio osservato. Per quanto riguarda i limiti di affidabilità dei risolutori di GLPK è sufficiente evidenziare gli eventuali casi in cui non ottengo soluzioni ottimali o GLPK dà errore. Neanche per i limiti di memoria di Matlab utilizzo tabelle, ma indico solamente per quali valori dei parametri la procedura si blocca per insufficienza di memoria. Se non viene specificato diversamente, i risolutori di GLPK troveranno sempre la soluzione ottimale, naturalmente indipendente dal metodo applicato, e Matlab non darà problemi di memoria.

Tutti i dati tranne i linguaggi e la cardinalità di $\mathcal{D}$ possono assumere valori diversi in base alla presenza o meno di informazioni aggiuntive sulla rete (pre-riduzione, P-vettori, T-vettori, classe di appartenenza, GMEC). I limiti di affidabilità del risolutore del *branch-and-bound* e il numero di posti dipendono dalla presenza di informazioni che consentono di applicare la pre-riduzione. Nei test non considero l'influenza delle altre eventuali informazioni strutturali o di Gmec sui valori di t_{proc} e sui limiti di memoria di Matlab perché è irrilevante per la variazione di essi. L'influenza su A è irrilevante per problemi complessi, ma può essere rilevante per problemi semplici, però non ne tengo conto per semplicità. I tempi di risoluzione del PPL potrebbero subire variazioni percentuali importanti, ma anche qui non dirò niente di più per non appesantire la trattazione.

Invece la presenza di altre informazioni sulla rete (comunque non di pre-riduzione) può condizionare fortemente l'affidabilità dei risolutori del simplesso e del punto interno, nel senso che l'inserimento di tali informazioni, sopratutto se non viene fatto con criterio, potrebbe impedire al risolutore di trovare una soluzione corretta. In altri termini, inserire informazioni casuali sulla rete senza conoscerne a fondo proprietà e caratteristiche può portare a soluzioni del PPL non ammissibili. Talvolta questa situazione si può verificare perfino se si conosce bene la rete che si sta testando e si aggiungono informazioni appropriate. Generalmente, come hanno dimostrato i test eseguiti sulle 4 reti trattate in questo capitolo, non si

hanno problemi se si impongono T-vettori o si aggiungono i vincoli che la rete complessiva sia una macchina di stato o un grafo marcato o una rete ordinaria. Invece occorre prestare molta attenzione quando si impongono P-vettori, Gmec o si aggiungono i vincoli che una o più sottoreti siano macchine di stato o grafi marcati: in questi casi è più facile che i risolutori ricavino soluzioni non ammissibili perché questi sono vincoli più difficili da imporre o da soddisfare. Nella sottosezione 7.2.3 farò due esempi di inserimento di informazioni aggiuntive, uno andato a buon fine e l'altro fallito.

I tempi di computazione potrebbero anche cambiare a seconda del metodo di risoluzione del PPL.

Inoltre i tempi potrebbero cambiare al variare della temperatura corrente del processore, che dipende da diversi fattori. Quindi i dati sui tempi non hanno la massima attendibilità perché può capitare di ottenere valori differenti ripetendo lo stesso test. Per problemi più complessi, con tempi di esecuzione di almeno qualche decina di minuti, le variazioni riscontrate non sono significative (variazioni percentuali inferiori al 10%) mentre possono essere rilevanti per tempi più bassi.

I dati si ricavano tutti lanciando il programma che esegue l'intera procedura. Il tempo di esecuzione della procedura completa si calcola mediante la sequenza di comandi

```
tic;[red_M0,red_Post,red_Pre,maxL2]=identif_proced
        (maxL,n,k,pre_red,T_inv,T_inc,T_dec,
    state_mach,ord_PN,msglev,dual,lpsolver);toc
```

che visualizza il tempo con una precisione al millesimo di secondo. Una precisione simile è importante per tempi di esecuzione molto brevi, altrimenti si può anche trascurare e considerare solo il numero intero di secondi.

Se interessa un singolo dato è sufficiente richiamare i programmi specifici. Di solito interessano tutti i dati, per cui per un test globale è più comodo richiamare la funzione `identif_proced` piuttosto che richiamare a una a una tutte le altre funzioni.

Per ogni rete e per ogni parametro rispetto al quale effettuerò i test, mi porrò come obiettivi limite quei valori del parametro o dei parametri considerati che bloccano l'esecuzione di `identif_proced` per insufficienza di memoria o che richiedono un tempo di esecuzione dell'intera procedura di almeno 2 ore.

Esattamente, per quanto riguarda il limite di memoria, effettuerò i test lanciando il programma `identif_proced` sino a quando non compare un messaggio di "Out of memory" nella finestra dei comandi di Matlab. Ciò significa che non c'è più spazio in memoria per allocare nuove variabili. Se compare il messaggio "***SEVERE CRITICAL ERROR*** from GLPK" dopo la chiamata alla funzione *glpkmex* significa che è presente anche in questo caso un problema di allocazione di memoria, ma è legato ai limiti di affidabilità dell'implementazione di qualche risolutore di GLPK. Se questi problemi si verificano quando non viene applicata alcuna pre-riduzione, potrebbero essere risolti proprio con la pre-riduzione. Comunque il problema non viene risolto definitivamente ed evitato in ogni caso, ma è solamente ritardato nel senso che si verificherà per un valore più grande del parametro considerato.

Per quanto riguarda il limite di tempo, effettuerò i test lanciando `identif_proced` sino a quando il tempo di computazione di questo programma non raggiunge le 2 ore. In quel caso prendo nota dei dati che sono stati calcolati nelle 2 ore e non eseguo altri test (anche se non si sono verificati problemi di memoria) per valori più grandi rispetto a quello considerato se sono sicuro che il tempo sia superiore, altrimenti devo continuare ad effettuare i test. Anche in questo caso si può ritardare il verificarsi di una tale situazione applicando una pre-riduzione. Il limite temporale solitamente viene imposto quando il tempo a disposizione per effettuare simulazioni è limitato. Inoltre non sarebbe molto utile imporre degli ingressi rispetto ai quali un programma impieghi molte ore prima di restituire le uscite.

I risultati dei test si ottengono dal confronto tra dati omogenei o dalle considerazioni su un singolo dato.

7.2 Esempi di testing

In questa sezione vengono illustrati 4 esempi di testing effettuati su reti P/T. Il primo esempio riguarda una rete che non rappresenta un sistema fisico. Gli altri 3 sono esempi di reti che hanno un significato fisico. Riporterò i dati principali ricavati dai test sulle reti e farò le considerazioni a riguardo. Comunque non riporterò tutti i dati per tutti gli esempi sopratutto per motivi di brevità. Alla fine dell'ultimo esempio saremo in grado di trarre le conclusioni del lavoro svolto.

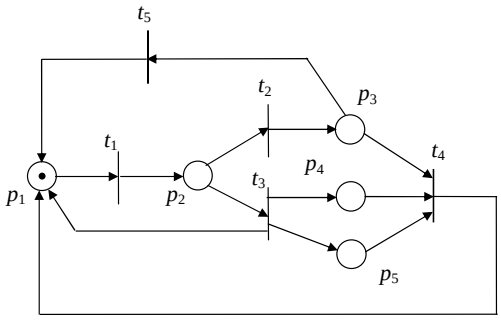

Figura 7.1: Il sistema di rete di Petri dell'Esempio 1.

7.2.1 Esempio 1

Consideriamo la rete rappresentata in Figura 7.1, già utilizzata nel capitolo 6. Considero la struttura della rete fissa, quindi effettuo i test per diversi valori di k. Ricordiamo che la struttura algebrica della rete è la seguente:

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Ricordiamo inoltre che le informazioni per la pre-riduzione sono le seguenti: le transizioni t_1 e t_5 hanno un solo posto in ingresso mentre le transizioni t_2 e t_3 sono in relazione di scelta libera. La rete è l'unica delle quattro trattate in questo capitolo che contiene transizioni in relazione di scelta libera.

Riporto la cardinalità di $maxL$ e D e le dimensioni di A, il numero di posti, i tempi di risoluzione del PPL e i tempi di esecuzione dell'intera procedura in 4 distinte tabelle. Riporto i tempi in secondi; uso una precisione al millesimo di secondo per valori di t_{proc} inferiori ai 10 secondi mentre indico questo dato con un numero intero per tempi superiori ai 10 secondi; visualizzo (t_{PPL} sempre come intero così come viene riportato da GLPK. Nelle caselle in cui compaiono sigle e non dati numerici significa che non è stato possibile ottenere il relativo dato a causa dei seguenti problemi:

- O.o.m.: "Out of memory", problema di memoria di Matlab.

- S.C.E.G.: "***SEVERE CRITICAL ERROR*** from GLPK", errore di GLPK.
- O.o.t.: "Out of time", raggiunto il tempo limite di 2 ore.
- N.I.: "NUMERIC INSTABILITY", instabilità numerica del metodo del punto interno.

Per semplicità riporto solamente i dati per alcuni valori significativi di k.

Il pedice *p-r* significa che il dato è ricavato applicando una pre-riduzione. Il pedice *duale* significa che il dato è ricavato applicando il metodo del simplesso duale. Applicherò tale metodo in un solo caso (per esempio quando eseguo una pre-riduzione), infatti l'unica possibile differenza rispetto al metodo del simplesso standard è nei tempi di computazione ed è sufficiente confrontare i due metodi senza bisogno di distinguere tra applicazione o meno della pre-riduzione nell'ambito di un determinato metodo. Il pedice *MPI* significa che il dato è ricavato applicando il metodo del punto interno. Anche questo metodo verrà applicato solo in caso di pre-riduzione sia perché l'unica possibile differenza rispetto al metodo del simplesso è nei tempi di computazione sia perché la sua implementazione in GLPK manca di importanti caratteristiche (poi spiegherò quali quando individuerò i limiti di applicabilità). Cioè, ai fini dei risultati dei test questo metodo mi interessa solamente per vedere se è più veloce del simplesso come potrebbe essere e quanto è affidabile. I nomi non contenenti alcun pedice sono legati a dati che si intendono ricavati applicando il metodo del simplesso revisionato e non avendo a disposizione alcun tipo di informazione aggiuntiva sulla rete.

Adesso analizziamo i dati riportati nelle tabelle e facciamo le dovute considerazioni sui risultati ottenuti che ovviamente sono parziali dovendo testare ancora le altre tre reti.

La procedura è stata eseguita sino a $k = 25$ (con la pre-riduzione), poi mi son fermato perché il tempo di esecuzione ha superato le 2 ore come si può notare nella corrispondente riga in Tabella 7.4. Non applicando alcuna pre-riduzione mi sarei bloccato per $k = 22$ per problemi di memoria ("Out of memory").

Come si può notare in Tabella 7.1, il numero di stringhe di $maxL$ aumenta più velocemente al crescere di k, ma ad un tasso inferiore rispetto a quello esponenziale. Inoltre i linguaggi maxL1 e maxL2 sono uguali al linguaggio osservato per tutti i casi tranne uno, quando viene applicato il metodo del punto interno, come vedremo tra poco. Anche il numero di righe di D ed A aumenta più velocemente al crescere di k. $maxL, D$ ed A più direttamente sono degli indicatori della com-

k	$card\_maxL$	$card\_D$	dim_A	dim_A_{p-r}
3	2	15	90×165	50×77
4	3	22	198×242	94×99
5	5	32	448×352	175×121
8	14	107	3060×935	925×264
12	68	493	19110×2310	4980×583
15	229	1600	56564×3938	13860×946
18	766	5323	142128×6204	33696×1452
20	1719	11930	243688×8074	56673×1859
21	2575	17950	314418×9174	72650×2101
22	3846	26979	O.o.m.	92103×2354
23	5872	40337	O.o.t.	115775×2629
24	8750	60947	O.o.t.	144020×2926
25	13248	O.o.t.	O.o.t.	O.o.t.

Tabella 7.1: Cardinalità di $maxL$ e D e dimensioni di A.

k	$card\_D$	q	q_{p-r}	p	p_{p-r}
3	15	15	7	4	4
4	22	22	9	4	4
5	32	32	11	5	5
8	107	85	24	4	4
12	493	210	53	4	4
15	1600	358	86	4	4
18	5323	564	132	4	4
20	11930	734	169	4	4
21	17950	834	191	4	4
22	26979	O.o.m.	214	O.o.m.	4
23	40337	O.o.t.	239	O.o.t.	4
24	60947	O.o.t.	266	O.o.t.	4
25	O.o.t.	O.o.t.	O.o.t.	O.o.t.	O.o.t.

Tabella 7.2: Numero di posti.

plessità del PPL. Il numero di variabili del PPL, cioè il numero di colonne di A, aumenta più velocemente al crescere di k, ma il fattore di crescita è sicuramente inferiore, e di molto per i valori di k più elevati, rispetto a quello che sarebbe se dovessimo risolvere un PPI. Ricordo infatti che il numero di variabili di un PPI aumenta con k di un fattore n^k, cioè esponenzialmente. La pre-riduzione non dà

k	t_{PPL}	t_{PPL_p-r}	$t_{PPL\_duale\_p-r}$	$t_{PPL\_MPI\_p-r}$
3	0	0	0	0
4	0	0	0	0
5	0	0	0	N.I.
8	1	0	0	0
12	3	0	0	N.I.
15	21	1	1	N.I.
18	91	6	4	N.I.
20	230	13	9	N.I.
21	562	39	27	N.I.
22	O.o.m.	30	19	N.I.
23	O.o.t.	48	29	N.I.
24	O.o.t.	70	41	N.I.
25	O.o.t.	O.o.t.	O.o.t.	O.o.t.

Tabella 7.3: Tempi di risoluzione del PPL.

k	t_{proc}	t_{proc_p-r}	$t_{proc\_duale\_p-r}$	$t_{proc\_MPI\_p-r}$
3	0.045	0.108	0.124	0.171
4	0.202	0.108	0.171	0.171
5	0.311	0.202	0.186	N.I.
8	1.23	0.48	0.42	0.623
12	5.869	1.119	1.025	N.I.
15	55	5.881	5.428	N.I.
18	195	39	37	N.I.
20	608	173	165	N.I.
21	2065	686	654	N.I.
22	O.o.m.	1503	847	N.I.
23	O.o.t.	1988	1783	N.I.
24	O.o.t.	4553	4279	N.I.
25	O.o.t.	O.o.t.	O.o.t.	O.o.t.

Tabella 7.4: Tempi di esecuzione dell'intera procedura.

grandi vantaggi per valori di k di poche unità mentre riesce a ridurre notevolmente la complessità per valori di k dell'ordine delle decine.

In Tabella 7.2 si può vedere come la pre-riduzione offra dei notevoli vantaggi sopratutto per grandi valori di k. Il numero di posti in entrambi i casi aumenta

gradualmente al crescere di k. Ma ciò che è più importante e che salta all'occhio subito è che la post-riduzione consente di ottenere una rete con 4 posti in quasi tutti i casi e quindi facilmente rappresentabile anche a mano. Quindi si è rivelata sicuramente efficace per questa rete. Confrontando la cardinalità di $\mathcal{D}$ col numero di posti q calcolato senza pre-riduzione, si nota come l'eliminazione coppie $(\vec{\sigma}, \vec{\sigma} + \vec{t})$ uguali relative a $\mathcal{D}$ sia fondamentale per ridurre la complessità della rete e impedire che subentrino troppo presto problemi di memoria. Anche qui il vantaggio è nettamente più significativo per i valori di k più alti.

Osservando le Tabelle 7.3 e 7.4 si nota subito che il metodo del punto interno fornisce soluzioni ottimali per 2 soli valori di k, quindi rivelandosi poco affidabile. In tutti gli altri casi ha portato ad instabilità numerica. Per $k = 8$ è riuscito a trovare una soluzione ottimale, ma con valori non nulli di $M0, Post$ e Pre dell'ordine di 10^{19}-10^{20} e comunque la rete identificata non generava il linguaggio osservato.

Dalla Tabella 7.3 si evince che il PPL viene risolto abbastanza velocemente ed è altresì notevole il vantaggio offerto dalla pre-riduzione: per esempio, notare che per $k = 21$ il PPL viene risolto in circa 40 secondi invece che in più di 9 minuti. Se utilizzo il metodo del simplesso duale risparmio dei secondi rispetto al simplesso standard.

Dalla tabella 7.4 ricaviamo risultati simili. La differenza tra il simplesso normale e quello duale è minima se pesata su tempi di computazione più grandi. Osservando ogni singola colonna ci si accorge che i tempi di computazione aumentano anch'essi più velocemente al crescere di k.

Rappresentiamo le reti P/T ottenute per $k = 3$ e $k = 4$. Non lo faccio per gli altri valori di k per motivi di brevità e per ovvii motivi di spazio.

Per $k = 3$ abbiamo:

Rete soluzione del PPL senza pre-riduzione ($q = 15$):

$$M_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Rete ridotta con una post-riduzione ($p = 4$):

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Rete soluzione del PPL con pre-riduzione ($q_{p-r} = 7$):

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Rete ridotta con una post-riduzione ($p_{p-r} = 4$):

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Per $k = 4$ abbiamo:

Rete soluzione del PPL senza pre-riduzione ($q = 22$):

$$M_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Rete ridotta con una post-riduzione ($p = 4$):

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Rete soluzione del PPL con pre-riduzione ($q_{p-r} = 9$):

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Rete ridotta con una post-riduzione ($p_{p-r} = 4$):

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Le reti ottenute per $k = 3$ hanno una struttura simile a quelle ottenute per $k = 4$, indipendentemente dal numero di posti.

Per un dato valore di k, la rete identificata risolvendo il PPL con una pre-riduzione è composta da parti della struttura della rete identificata senza la pre-riduzione. Le reti ridotte sono uguali e sono in qualche modo simili alla struttura di rete nota in partenza.

Anche per valori di k più alti valgono le stesse considerazioni.

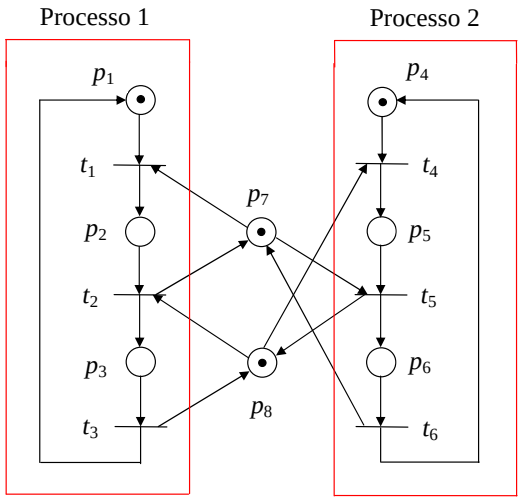

Figura 7.2: Il problema dei filosofi a tavola.

7.2.2 Esempio 2: il problema dei filosofi a tavola

La rete dell'Esempio 2 rappresenta un ben noto processo alimentare bloccante, meglio conosciuto come problema dei filosofi a tavola.

Il problema dei filosofi esemplifica come possa nascere uno stallo (deadlock) quando più processi usano le stesse risorse in ordine diverso. La rete in Figura 7.2 rappresenta due processi che lavorano in parallelo. Il primo processo (1) richiede due operazioni, per la prima delle quali ha bisogno della risorsa $R1$ (posto p_7) e per la seconda delle quali ha bisogno della risorsa $R2$ (posto p_8). Il secondo processo (2) richiede due operazioni, per la prima delle quali ha bisogno della risorsa $R2$ e per la seconda delle quali ha bisogno della risorsa $R1$. In pratica il primo filosofo $F1$ prende prima la forchetta $R1$ e poi la $R2$ per mangiare. Invece il secondo filosofo $F2$ usa prima $R2$ e poi $R1$. Dunque se $F1$ ha preso $R1$ e $F2$ ha preso $R2$, ciascuno è bloccato attendendo che l'altro liberi la risorsa (forchetta) occupata.

La rete non è parametrica nella struttura e nella marcatura iniziale, quindi effettuo i test per diversi valori significativi di k. La struttura algebrica della rete è riportata nel seguito:

k	$card\_maxL$	$card\_D$	dim_A	dim_A_{p-r}
3	2	24	168×312	132×234
4	4	32	352×416	252×286
7	8	88	2100×1092	1428×728
12	16	360	14640×3120	9960×2106
16	64	872	39552×4992	26292×3302
20	256	2280	99372×7644	64428×4940
21	128	3048	113568×8736	76272×5850
23	512	4584	166656×9984	108336×6474
25	512	7144	211500×11700	140364×7748
26	1024	9192	263412×12636	171612×8216
27	512	12264	O.o.m.	196020×9386
28	1024	14312	O.o.m.	216300×9646
30	1024	24552	O.o.m.	292380×11466
31	2048	28648	O.o.m.	O.o.m.

Tabella 7.5: Cardinalità di $maxL$ e D e dimensioni di A.

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

Supponiamo di avere le seguenti informazioni per la pre-riduzione: le transizioni t_3 e t_6 hanno un solo posto in ingresso e non ci sono transizioni in relazione di scelta libera.

Riporto i dati su $maxL, D, A, q, p, t_{PPL}$ e t_{proc} in 4 distinte tabelle, strutturate come nell'Esempio 1.

Analizziamo i dati riportati nelle tabelle e facciamo le dovute considerazioni sui risultati ottenuti.

Ho riscontrato problemi di memoria ("Out of memory") per i seguenti valori di k e nei seguenti casi:

k	$card\_D$	q	q_{p-r}	p	p_{p-r}
3	24	24	18	5	5
4	32	32	22	5	5
7	88	84	56	5	5
12	360	240	162	5	5
16	872	384	254	5	5
20	2280	588	380	7	7
21	3048	672	450	5	5
23	4584	768	498	7	7
25	7144	900	596	5	5
26	9192	972	632	7	7
27	12264	O.o.m.	722	O.o.m.	5
28	14312	O.o.m.	742	O.o.m.	5
30	24552	O.o.m.	882	O.o.m.	5
31	28648	O.o.m.	O.o.m.	O.o.m.	O.o.m.

Tabella 7.6: Numero di posti.

k	t_{PPL}	t_{PPL_p-r}	$t_{PPL\_duale\_p-r}$	$t_{PPL\_MPI\_p-r}$
3	0	0	0	N.I.
4	0	0	0	0
7	0	0	0	N.I.
12	7	5	2	N.I.
16	28	7	7	N.I.
20	54	25	24	N.I.
21	90	46	40	N.I.
23	145	64	57	N.I.
25	240	114	104	N.I.
26	303	139	135	N.I.
27	O.o.m.	411	363	N.I.
28	O.o.m.	410	216	N.I.
30	O.o.m.	816	O.o.m.	N.I.
31	O.o.m.	O.o.m.	O.o.m.	O.o.m.

Tabella 7.7: Tempi di risoluzione del PPL.

- $k = 27$, senza pre-riduzione.
- $k = 28$, senza pre-riduzione.

k	t_{proc}	t_{proc_p-r}	$t_{proc\_duale\_p-r}$	$t_{proc\_MPI\_p-r}$
3	0.155	0.14	0.108	N.I.
4	0.232	0.201	0.186	0.42
7	0.996	0.683	0.652	N.I.
12	12	7.412	6.662	N.I.
16	52	13	12	N.I.
20	106	60	53	N.I.
21	169	93	90	N.I.
23	297	143	134	N.I.
25	484	260	246	N.I.
26	700	361	340	N.I.
27	O.o.m.	936	900	N.I.
28	O.o.m.	1098	614	N.I.
30	O.o.m.	2420	O.o.m.	N.I.
31	O.o.m.	O.o.m.	O.o.m.	O.o.m.

Tabella 7.8: Tempi di esecuzione dell'intera procedura.

- $k = 30$, con pre-riduzione e col simplesso duale.
- $k = 31$, in tutti i casi delle tabelle.

Quindi l'applicazione della pre-riduzione ha ritardato la comparsa del messaggio di memoria piena di 3 e 4 unità di k nei due casi.

Come si può notare in Tabella 7.5, il numero di stringhe del linguaggio massimale è una potenza di due al variare di k e non presenta il tasso di crescita della rete dell'Esempio 1, anzi, in certi casi diminuisce al crescere di k. Stavolta $maxL1$ e $maxL2$ non sono sempre uguali al linguaggio osservato in ogni istanza di applicazione del metodo del simplesso, ma sono state riscontrate le seguenti anomalie:

- Simplesso normale: per $k = 20, k = 23$ e $k = 26$ viene trovata una soluzione ottimale, ma con valori degli elementi elevatissimi, dell'ordine di 10^{15}-10^{19}, analogamente a quanto visto in un caso dell'Esempio 1 applicando il metodo del punto interno. $maxL1$ non risulta uguale a $maxL$. $maxL2$ non è uguale a $maxL$ solo per $k = 23$ con pre-riduzione.
- Simplesso duale: per $k = 23$ e $k = 26$ vengono ancora trovate soluzioni

ottimali con dei valori molto alti. $maxL1$ e $maxL2$ sono diversi da $maxL$ solo per $k = 23$.

Le dimensioni di D ed A aumentano più velocemente al crescere di k, ma non in maniera esponenziale: è una situazione simile a quella dell'Esempio 1. La pre-riduzione non dà vantaggi rilevanti neanche per i più grandi valori di k, a differenza dell'Esempio 1.

In Tabella 7.6 si può vedere come questa volta la pre-riduzione non riduce significativamente il numero di posti come nell'Esempio 1 in cui per elevati valori di k tale numero veniva ridotto di un fattore 4 circa. Invece in quest'esempio non viene neanche dimezzato. Viceversa, il numero di posti aumenta sempre in modo graduale come nell'Esempio 1. D'altra parte, la post-riduzione si è rivelata ancora efficace visto che riusciamo ad ottenere una semplice rete con 5 posti, qualche volta con 7. Inoltre, ancora una volta l'introduzione della variante alla procedura di base che consente di eliminare le coppie $(\vec{\sigma}, \vec{\sigma} + \vec{t})$ uguali riduce abbastanza il numero di posti in modo molto più evidente per $k > 20$.

Dalle tabelle dei tempi si vede che il metodo del punto interno fornisce soluzioni ottimali con valori dell'ordine di $10^{17} - 10^{19}$ solo per $k = 4$, quindi rivelandosi ancora meno affidabile rispetto al primo esempio. In tutti gli altri casi ha portato ad instabilità numerica. Questa situazione si è verificata anche per altre reti, quindi c'è un denominatore comune: questo metodo in qualche caso ricava una soluzione ottimale, per piccoli valori di k come abbiamo visto, ma spesso conduce all'instabilità non appena, aumentando k, il problema incomincia a complicarsi. Perciò nei prossimi esempi non riporterò più il dato sul metodo del punto interno.

Dalla Tabella 7.7 si evince che il PPL viene risolto al massimo in pochi minuti, anche se non si ottengono grandi risparmi con la pre-riduzione. Anche in quest'esempio la differenza tra il simplesso normale e quello duale è minima a vantaggio del duale, anzi, percentualmente si va assottigliando sempre più al crescere di k. Anche su altre reti è stato dimostrato questo, quindi si può concludere che i due metodi conducono a risultati simili, talvolta anche molto simili, per quanto riguarda i tempi di computazione. In virtù di ciò e per questioni di tempo, nei prossimi esempi non ricorrerò più al simplesso normale, ma continuerò a utilizzare solamente il metodo del simplesso duale.

In generale i tempi di computazione aumentano gradualmente e più lentamente rispetto all'Esempio 1.

Per semplicità, riporto solamente le reti ridotte con una post-riduzione per $k = 3$ e $k = 4$:

Per $k = 3$ abbiamo:

Rete ridotta senza aver applicato una pre-riduzione ($p = 4$):

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0\,0 \end{bmatrix}, Post = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, Pre = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Rete ridotta avendo applicato anche una pre-riduzione ($p_{p-r} = 4$):

$$M_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, Post = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, Pre = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Per $k = 4$ abbiamo:

Rete ridotta senza aver applicato una pre-riduzione ($p = 4$):

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0\,0 \end{bmatrix}, Post = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, Pre = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Rete ridotta avendo applicato anche una pre-riduzione ($p_{p-r} = 4$):

$$M_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, Post = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, Pre = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Le reti identificate per $k = 3$ e $k = 4$ sono molto simili. Le 2 reti identificate per $k = 3$ hanno la stessa struttura, quelle ottenute per $k = 4$ differiscono solo per la presenza di una arco. Inoltre la post-riduzione è riuscita a ricostruire una parte della rete iniziale. Queste caratteristiche si ritrovano anche per valori maggiori di k.

7.2.3 Esempio 3: il protocollo di comunicazione

La rete dell'Esempio 3 rappresenta un processo emittente-ricevente basato su un protocollo di comunicazione.

Consideriamo il sistema di rete di Petri in Figura 7.3 che consiste di $2(s + 1)$ posti e $2s$ transizioni. Esso modella il processo emittente-ricevente, un sistema di comunicazione basato su un assegnato protocollo. In particolare, i posti p_i, con $i = 1, ..., s$, modellano il processo emittente. I posti p_i, con $i = s + 1, ..., 2s$, modellano il processo ricevente. I posti p_i, con $i = 2s + 1, 2s + 2$, corrispondono ai canali di comunicazione tra i due processi. Quando il posto p_1 è marcato l'emittente è pronto a trasmettere un messaggio. Dopo lo scatto di t_1 il messaggio è sul canale, pronto per essere ricevuto, a patto che sia pronto anche il ricevente, cioè che il posto p_{s+1} sia marcato. Adesso le transizioni $t_2, ..., t_{s-1}, t_{s+1}, ..., t_{2s}$ possono scattare. In particolare lo scatto di t_{2s} corrisponde all'invio della conferma della ricezione da parte del ricevente. La ricezione della conferma da parte dell'emittente è modellata dalla transizione t_s.

Se il numero di gettoni $m0$ dei posti p_1 e p_s è inizialmente maggiore di uno, ci sono più emittenti (in numero pari a $m0$) e più riceventi (in numero pari a $m0$) che stanno comunicando tra loro.

La rete è parametrica nella marcatura iniziale (p_1 e p_{s+1} hanno rispettivamente un numero di gettoni iniziale pari a $m0$) e nelle dimensioni. Infatti il numero di posti m e il numero di transizioni n della rete da testare dipendono dal parametro s come ho specificato all'inizio di questa sottosezione. Abbiamo $m = 2s + 2$ e $n = 2s$. Quindi al crescere di s aumentano le dimensioni della rete che devo testare perché aumentano il numero di posti e il numero di transizioni. Al contrario, negli altri esempi testo sempre una rete che in partenza ha sempre lo stesso numero di posti e lo stesso numero di transizioni. Quindi effettuo i test facendo variare i parametri $m0$ ed s e riportando i dati per alcuni valori significativi dei parametri. Il parametro k in quest'esempio è pari a $2s$. In questo modo si può riottenere la marcatura iniziale dopo lo scatto delle sequenze di lunghezza k (se-

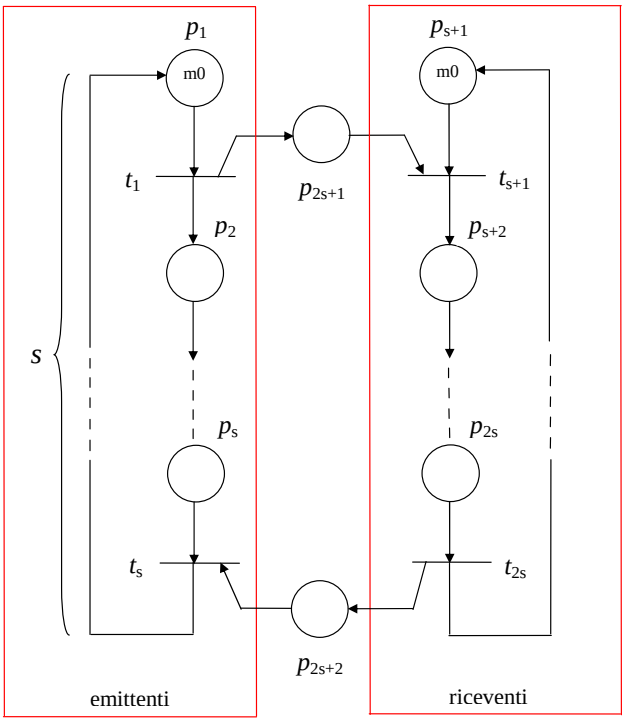

Figura 7.3: Il processo emittente-ricevente.

s	k	$m0$	$card\_maxL$	$card\_D$	$size\_A$	$size\_A_{p-r}$
2	4	1	1	12	60×108	44×72
2	4	5	10	15	210×126	168×99
2	4	15	10	15	210×126	168×99
3	6	1	4	72	546×546	270×247
3	6	10	274	302	9306×1222	4504×585
3	6	20	274	302	9306×1222	4504×585
4	8	1	15	378	2600×1768	920×578
4	8	10	7635	8718	211248×8313	65570×2567
4	8	12	7635	8718	211248×8313	65570×2567
5	10	1	56	1835	8610×4410	2330×1113
5	10	2	25008	78727	O.o.t.	O.o.t.
6	12	1	210	8503	22692×9300	4932×1900
6	12	2	O.o.t.	O.o.t.	O.o.t.	O.o.t.
7	14	1	792	38248	51170×17458	9254×2987
8	16	1	3003	168598	O.o.t.	O.o.t.

Tabella 7.9: Cardinalità di $maxL$ e D e dimensioni di A.

quenze ripetitive e stazionarie), così che il comportamento della rete possa essere ciclico per $k > 2s$.

Supponiamo di avere le seguenti informazioni per la pre-riduzione: le transizioni $t_1, ..., t_{s-1}, t_{s+2}, ..., t_{2s}$ hanno un solo posto in ingresso e non ci sono transizioni in relazione di scelta libera. Inoltre, poiché la rete ammette un T-invariante di elementi tutti pari a 1, supponiamo di avere in ingresso tale informazione e aggiungiamo i relativi vincoli al CS. Se non avessimo questa informazione non riusciremmo ad ottenere una rete ripetitiva (o ciclica), come è la rete del protocollo che stiamo testando, in cui le sequenze di lunghezza k o $2s$ possano esser ripetute per multipli di $2s$ (sequenze ripetitive e stazionarie). Oppure, visto che la rete è un grafo marcato (ogni posto ha un arco in ingresso e un arco in uscita), si poteva anche supporre di avere tale informazione in ingresso.

Anche qui riporto i dati su $maxL, D, A, q, p, t_{PPL}$ e t_{proc} in 4 tabelle. Ricordo che verrà utilizzato solamente il metodo del simplesso duale.

Ancora una volta, procediamo con l'analisi dei dati contenuti nelle tabelle e con le relative considerazioni.

Ho effettuato i test sino a $s = 8$ ed $m0 = 1$, poi mi son fermato perché il tempo di

s	k	$m0$	$card\_D$	q	q_{p-r}	p	p_{p-r}
2	4	1	12	12	8	4	4
2	4	5	15	14	11	3	3
2	4	15	15	14	11	3	3
3	6	1	72	42	19	7	7
3	6	10	302	94	45	6	6
3	6	20	302	94	45	6	6
4	8	1	378	104	34	9	9
4	8	10	8718	489	151	8	8
4	8	12	8718	489	151	8	8
5	10	1	1835	210	53	11	11
5	10	2	78727	O.o.t.	O.o.t.	O.o.t.	O.o.t.
6	12	1	8503	372	76	13	13
6	12	2	O.o.t.	O.o.t.	O.o.t.	O.o.t.	O.o.t.
7	14	1	38248	602	103	15	15
8	16	1	168598	O.o.t.	O.o.t.	O.o.t.	O.o.t.

Tabella 7.10: Numero di posti.

s	k	$m0$	t_{PPL_duale}	$t_{PPL\_duale\_p-r}$
2	4	1	0	0
2	4	5	0	0
2	4	15	0	0
3	6	1	0	0
3	6	10	0	0
3	6	20	0	0
4	8	1	0	0
4	8	10	102	6
4	8	12	186	26
5	10	1	6	0
5	10	2	O.o.t.	O.o.t.
6	12	1	60	2
6	12	2	O.o.t.	O.o.t.
7	14	1	46	5
8	16	1	O.o.t.	O.o.t.

Tabella 7.11: Tempi di risoluzione del PPL.

s	k	$m0$	t_{proc_duale}	$t_{proc\_duale\_p-r}$
2	4	1	0.078	0.047
2	4	5	0.063	0.062
2	4	15	0.062	0.046
3	6	1	0.485	0.406
3	6	10	2.11	1.063
3	6	20	1.922	1.063
4	8	1	1.172	0.344
4	8	10	828	418
4	8	12	1453	722
5	10	1	25	5.14
5	10	2	O.o.t.	O.o.t.
6	12	1	262	49
6	12	2	O.o.t.	O.o.t.
7	14	1	1933	467
8	16	1	O.o.t.	O.o.t.

Tabella 7.12: Tempi di esecuzione dell'intera procedura.

esecuzione della procedura ha superato le 2 ore. Il limite di tempo è stato superato anche per $s = 5$ ed $s = 6$, con $m0 = 2$. Stavolta la pre-riduzione non ha ritardato il problema.

Dalla Tabella 7.9 si vede come il numero di stringhe di $maxL$ aumenti in maniera esponenziale al crescere di s, fissato $m0$. Mantenendo fisso s e aumentando $m0$, si nota un'iniziale crescita, graduale sino ad $s = 3$ o $s = 4$ ed esponenziale per valori più alti, poi oltre determinati valori di $m0$ il linguaggio calcolato rimane invariato. $maxL1$ e $maxL2$ sono risultati sempre uguali a $maxL$. Le dimensioni di D e il numero di vincoli (o righe di A) aumentano in maniera esponenziale al crescere di s. Invece il numero di variabili (o colonne di A) aumenta gradualmente, come già visto negli altri esempi. La pre-riduzione è molto vantaggiosa per i più grandi valori di k, come nell'Esempio 1.

Dalla Tabella 7.10 viene confermato un risultato già ottenuto negli altri esempi: la pre-riduzione è vantaggiosa sopratutto per grandi valori di s (o k). Il numero di posti in generale aumenta gradualmente con s mentre per valori di s maggiori di 3 o 4 si nota un aumento sensibile al crescere di $m0$. Oltre un certo valore di $m0$ il numero di posti si stabilizza ad un valore fisso. La post-riduzione si dimostra ancora efficace riuscendo ad estrarre reti di piccole dimensioni. Il numero di posti della rete ridotta aumenta di 2 unità al crescere di s e fissato $m0$. Invece, dato

s, si nota che per un determinato valore di $m0$ la post-riduzione elimina un altro posto ridondante. Come nei primi 2 esempi, la differenza tra $card\_D$ e q si va accentuando al crescere di s; addiritura per $s = 7$ ed $m0 = 1$ il numero di posti viene ridotto da più di 38000 unità a 600 circa.

Nella Tabella 7.11 si nota subito che la pre-riduzione riesce a ridurre nettamente il t_{PPL} per $s \geq 4$. I valori assoluti dei tempi aumentano più velocemente al crescere di q ed $m0$.

La sensibile riduzione vista per il t_{PPL} non si estende al t_{proc} come si evince dalla Tabella 7.12, infatti il vantaggio della pre-riduzione è un pò meno evidente. I tempi aumentano in maniera esponenziale.

In sintesi, osservando le 4 Tabelle 7.9-7.12 si nota come la complessità del problema di identificazione aumenti in maniera spropositata a partire da $s = 5$ e in modo particolare quando si passa da $m0 = 1$ a $m0 = 2$.

Per concludere, ecco un esempio di inserimento di informazione aggiuntiva non andato a buon fine: provando ad inserire come vincolo aggiuntivo la GMEC $\vec{1}^T \cdot M_0 \leq 1$ per $s = 3$ ed $m0 = 2$, cioè imponendo che il numero di gettoni nella marcatura iniziale non sia superiore a 1, ottengo una soluzione non ammissibile ($M_0, Post$ e Pre hanno elementi tutti nulli) perchè sto imponendo un vincolo che la rete non può soddisfare.

Per semplicità, sotto riporto solamente le reti ridotte per $s = 2(k = 4, m = 6, n = 4)$, con $m0 = 1$ ed $m0 = 5$, e per $s = 3(k = 6, m = 8, n = 6)$, con $m0 = 1$, nel caso in cui viene applicata anche la pre-riduzione. Per $s = 2$ ed $m0 = 1$ riporto anche la rete che si otterrebbe senza l'aggiunta dei vincoli legati al T-invariante di elementi tutti pari ad 1 (e/o al grafo marcato).

Per $s = 2$ ed $m0 = 1$ senza il T-invariante abbiamo ($p_{p-r} = 4$):

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Per $s = 2$ ed $m0 = 1$ col T-invariante abbiamo ($p_{p-r} = 4$):

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

L'aggiunta dei vincoli legati al T-invariante o al grafo marcato consente di ottenere una rete che sia proprio un grafo marcato, come risulta essere il protocollo di comunicazione. Infatti ogni riga delle matrici $Post$ e Pre ha un elemento pari a 1 e tutti gli altri a 0, che significa che ogni posto ha un arco in ingresso e un arco in uscita (grafo marcato). Senza l'aggiunta di questi vincoli non avremmo ottenuto un grafo marcato, visto che la prima riga di $Post$ ha elementi tutti nulli, cioè il posto p_1 non ha un arco in ingresso.

Per $s = 2$ ed $m0 = 5$ abbiamo ($p_{p-r} = 3$):

$$M_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Il passaggio da $m0 = 1$ ad $m0 = 5$ riduce la rete di un posto, mantenendo una struttura simile.

Per $s = 3$ ed $m0 = 1$ abbiamo ($p_{p-r} = 7$):

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, Post = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, Pre = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Per $s = 3$ ed $m0$ si può notare che il numero di posti della rete identificata è stato ridotto di un'unità rispetto alla rete di partenza.

Le considerazioni precedenti valgono anche per valori maggiori di s ed $m0$.

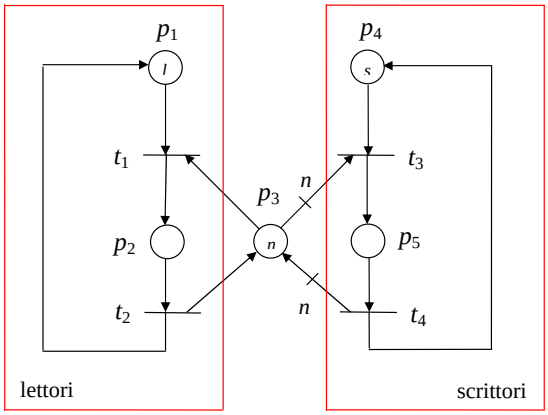

Figura 7.4: Il processo lettori-scrittori.

7.2.4 Esempio 4: il processo lettori-scrittori

La rete dell'Esempio 4 rappresenta il processo lettori-scrittori. Quest'esempio verrà trattato più velocemente dei precedenti visto che molti risultati già ottenuti si possono generalizzare per molti tipi di rete. Nel campo dei sistemi operativi, un problema classico è quello di conservare l'integrità dei dati in presenza di vari processi che leggono ("lettori") e modificano ("scrittori") i dati. Si supponga che vi siano l lettori e s scrittori. Ogni lettore può essere inattivo o nello stato di lettura. Ogni scrittore può essere inattivo o nello stato di scrittura. Tale processo deve rispettare le due specifiche:

- Più lettori possono leggere contemporaneamente, sino a un massimo di n.
- Se uno scrittore ha accesso ai dati nessun altro scrittore o lettore può avere accesso ai dati.

Il processo complessivo è descritto dalla rete in Figura 7.4. Il posto p_3 impone il soddisfacimento dei vincoli sopra descritti. Infatti dalla marcatura in figura se uno scrittore accede ai dati (scatto della transizione t_3) il posto p_3 si svuota e nessun altro lettore o scrittore può accedere ai dati sin quando non termina l'operazione di scrittura (scatto della transizione t_4) che restituisce le n marche al posto p_3. Se viceversa dalla marcatura in figura un lettore accede ai dati (scatto della transizione t_1), il posto p_3 perde una marca (contenendone $n-1$) e nessuno scrittore potrà accedere ai dati, mentre ancora altri $n-1$ lettori potranno accedervi.

La rete è parametrica nella struttura (2 archi di peso n) e nella marcatura iniziale (p_1, p_4 e p_3 hanno rispettivamente un numero di gettoni iniziale pari a l, s ed n), quindi effettuo i test per diversi valori significativi sia di k sia di l, s ed n. Per semplicità, per questi ultimi tre parametri non considero valori superiori alle poche decine, al massimo 50, e comunque non ho effettuato i test per tutte le loro possibili combinazioni. Gli unici vincoli da rispettare obbligatoriamente nella scelta dei tre valori sono che l non sia inferiore nè a s, nè a n. La rappresentazione algebrica della rete in funzione di l, s ed n è la seguente:

$$M_0 = \begin{bmatrix} l \\ 0 \\ n \\ s \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & n \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & n & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Supponiamo di non avere alcun tipo di informazione aggiuntiva sulla struttura della rete.

Riporto i dati su $maxL, D, q, p, t_{PPL}$ e t_{proc} sempre in 4 tabelle. Poiché le variazioni delle dimensioni di A valutate negli esempi 1,2 e 3 sono tipiche di molte altre reti compreso il processo lettori-scrittori, in quest'esempio non le riporterò in tabella per motivi di brevità. In ogni caso si possono ricavare delle indicazioni sulla complessità del problema dalla cardinalità di D e dal numero di posti. Ricordo che anche in questo esempio utilizzerò solamente il metodo del simplesso duale.

Ancora una volta, analizziamo i dati riportati nelle tabelle e facciamo le dovute considerazioni.

Considerando il limite di 50 per i valori di l, s ed n, il primo valore di k per il quale ho bloccato l'esecuzione del programma è stato 15 perché per $l = 35, s = 25$ ed $n = 30$ il tempo di esecuzione della procedura ha superato le 2 ore. Ho effettuato i test sino a $k = 26$ ed $l = s = n = 1$. Sono sopraggiunti problemi di memoria ("Out of memory") per $k = 25$ e $k = 26$. Per $k = 16, l = 6, s = n = 5$ GLPK ha dato un errore ("***SEVERE CRITICAL ERROR FROM GLPK***") di allocazione di memoria. Tengo a precisare che i valori limite dei diversi parametri spesso sono solo indicativi, se avessi voluto la massima precisione avrei dovuto effettuare i test per tutti i loro possibili valori, la qual cosa avrebbe richiesto tempi molto lunghi.

k	l	s	n	$card\_maxL$	$card\_D$
3	5	1	3	6	13
3	5	5	5	6	13
3	10	5	8	6	13
5	5	1	3	18	46
5	5	5	5	20	47
5	10	5	8	20	47
10	5	1	3	243	847
10	5	5	5	396	1081
10	10	5	8	460	1113
13	5	5	5	2704	6393
13	20	10	15	3432	7450
13	50	30	40	3432	7450
15	5	5	5	9232	21825
15	20	10	15	12870	27613
15	50	40	45	12870	27613
16	5	3	4	12087	33151
16	5	5	5	15760	42993
16	6	5	5	15760	42993
16	10	5	8	22950	53032
17	5	1	3	13122	36082
17	6	2	4	21794	59705
18	4	1	2	6765	26068
18	4	1	3	19683	68887
19	4	1	2	10946	42182
19	4	1	3	39366	O.o.t.
20	2	1	1	1024	8184
20	2	1	2	17711	68255
21	2	1	1	2048	10232
23	2	1	1	4096	20472
25	1	1	1	8192	40952
25	2	1	1	8192	40952
26	1	1	1	8192	65528

Tabella 7.13: Cardinalità di $maxL$ e D.

Dalla Tabella 7.13 si vede che il numero di stringhe di $maxL$ aumenta in maniera graduale al crescere di k e fissati l, s ed n. Mantenendo fisso k e variando gli altri parametri, l'aumento è meno evidente, e oltre determinati valori il linguaggio

k	l	s	n	$card\_D$	q	p
3	5	1	3	13	13	3
3	5	5	5	13	13	3
3	10	5	8	13	13	3
5	5	1	3	46	36	3
5	5	5	5	47	37	3
5	10	5	8	47	37	3
10	5	1	3	847	155	3
10	5	5	5	1081	175	3
10	10	5	8	1113	184	3
13	5	5	5	6393	293	3
13	20	10	15	7450	343	3
13	50	30	40	7450	343	3
15	5	5	5	21825	397	3
15	20	10	15	27613	492	3
15	50	40	45	27613	492	3
16	5	3	4	33151	427	3
16	5	5	5	42993	469	3
16	6	5	5	42993	469	S.C.E.G.
16	10	5	8	53032	O.o.t.	O.o.t.
17	5	1	3	36082	426	3
17	6	2	4	59705	O.o.t.	O.o.t.
18	4	1	2	26068	423	3
18	4	1	3	68887	O.o.t.	O.o.t.
19	4	1	2	42182	470	3
19	4	1	3	O.o.t.	O.o.t.	O.o.t.
20	2	1	1	8184	440	3
20	2	1	2	68255	O.o.t.	O.o.t.
21	2	1	1	10232	462	3
23	2	1	1	20472	552	3
25	1	1	1	40952	650	3
25	2	1	1	40952	650	3
26	1	1	1	65528	728	3

Tabella 7.14: Numero di posti.

calcolato non cambia più, come succedeva nel caso del protocollo di comunicazione. $maxL2$ è sempre uguale a $maxL$, ma non si può dire la stessa cosa per quanto riguarda $maxL1$. Infatti, anche in questo esempio, come nell'Esempio

k	l	s	n	t_{PPL_duale}
3	5	1	3	0
3	5	5	5	0
3	10	5	8	0
5	5	1	3	0
5	5	5	5	0
5	10	5	8	0
10	5	1	3	3
10	5	5	5	7
10	10	5	8	2
13	5	5	5	17
13	20	10	15	35
13	50	30	40	68
15	5	5	5	52
15	20	10	15	139
15	50	40	45	242
16	5	3	4	61
16	5	5	5	127
16	6	5	5	S.C.E.G.
16	10	5	8	O.o.t.
17	5	1	3	50
17	6	2	4	O.o.t.
18	4	1	2	38
18	4	1	3	O.o.t.
19	4	1	2	63
19	4	1	3	O.o.t.
20	2	1	1	38
20	2	1	2	O.o.t.
21	2	1	1	39
23	2	1	1	87
25	1	1	1	151
25	2	1	1	148
26	1	1	1	216

Tabella 7.15: Tempi di risoluzione del PPL.

2, il risolutore può trovare soluzioni ottimali con valori molto elevati, dell'ordine di 10^{15}-10^{19}. Tali anomalie si verificano più frequentemente in questo esempio, esattamente per valori di k maggiori di 10, indipendentemente da l, s ed n, e

k	l	s	n	t_{proc_duale}
3	5	1	3	0.14
3	5	5	5	0.187
3	10	5	8	0.187
5	5	1	3	0.391
5	5	5	5	0.39
5	10	5	8	0.437
10	5	1	3	14
10	5	5	5	24
10	10	5	8	12
13	5	5	5	157
13	20	10	15	263
13	50	30	40	555
15	5	5	5	2123
15	20	10	15	3749
15	50	40	45	6214
16	5	3	4	4080
16	5	5	5	O.o.t.
16	6	5	5	S.C.E.G.
16	10	5	8	O.o.t.
17	5	1	3	4304
17	6	2	4	O.o.t.
18	4	1	2	2467
18	4	1	3	O.o.t.
19	4	1	2	4265
19	4	1	3	O.o.t.
20	2	1	1	159
20	2	1	2	O.o.t.
21	2	1	1	297
23	2	1	1	986
25	1	1	1	O.o.m.
25	2	1	1	O.o.m.
26	1	1	1	O.o.m.

Tabella 7.16: Tempi di esecuzione dell'intera procedura.

quando non si applica la pre-riduzione. Il numero di elementi di D aumenta più sensibilmente rispetto a $maxL$ al crescere dei parametri e oltre determinati valori di l, s ed n rimane invariato. Quindi si può ritenere che il numero di vincoli,

e la complessità del problema, aumentIno ad un tasso inferiore rispetto a quello esponenziale.

Dalla Tabella 7.14 si nota come il numero di posti non aumenta sensibilmente con k e, come per il linguaggio, oltre determinati valori di l, s ed n rimane fisso, analogamente all'Esempio 3. Quest'ultimo risultato era scontato poiché, se il linguaggio non cambia, allora verranno costruite le stesse matrici E e D e lo stesso CS, e quindi nemmeno il numero di posti cambia. Ancora una volta, la riduzione dei posti dovuta all'eliminazione di coppie $(\vec{\sigma}, \vec{\sigma} + \vec{t})$ uguali è notevole per grandi valori di k. E ancora una volta la post-riduzione riesce ad ottenere una rete semplice, in questo caso contenente 3 o 4 posti.

Dalla Tabella 7.15 si evince che il PPL viene risolto molto velocemente.

Dalla Tabella 7.16 si nota che il tempo di computazione aumenta più o meno gradualmente all'aumentare dei parametri, ma un pò più velocemente per valori più alti di questi.

Riporto le reti ridotte per $k = 3$, con $l = 5, s = 1, n = 3$ e $l = 5, s = 5, n = 5$, e per $k = 5$ con $l = 5, s = 1, n = 3$.

Per $k = 3$ ed $l = 5, s = 1, n = 3$ abbiamo ($p = 3$):

$$M_0 = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 1 & 0 & 3 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 1 & 0 & 3 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}.$$

Per $k = 3$ ed $l = s = n = 5$ abbiamo ($p = 3$):

$$M_0 = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 1 & 0 & 3 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 1 & 0 & 3 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}.$$

Per $k = 5$ ed $l = 5, s = 1, n = 3$ abbiamo ($p = 3$):

$$M_0 = \begin{bmatrix} 27 \\ 0 \\ 0 \end{bmatrix}, \quad Post = \begin{bmatrix} 0 & 9 & 0 & 27 \\ 18 & 0 & 0 & 0 \\ 0 & 0 & 18 & 0 \end{bmatrix}, \quad Pre = \begin{bmatrix} 9 & 0 & 27 & 0 \\ 0 & 18 & 0 & 0 \\ 0 & 0 & 0 & 18 \end{bmatrix}.$$

Per $k = 5$ il risolutore del simplesso non è affidabile. Tra poco spiegherò il motivo.

7.3 Risultati finali del testing

In questa sezione vengono riportati i risultati e le considerazioni conclusive del testing sulle reti P/T.

Limiti di affidabilità del risolutore di GLPK basato sul metodo del simplesso.

Nella stragrande maggioranza dei casi il risolutore del simplesso trova soluzioni ottimali e corrette nel senso che la rete soluzione del PPL genera il linguaggio osservato. In alcuni casi, quasi sempre per valori di k di qualche decina e per un numero di vincoli del PPL dell'ordine di 10^5 e senza pre-riduzione, sono state riscontrate le seguenti anomalie:

1. La soluzione trovata è ottimale e i linguaggi generati dalle reti ridotta e non ridotta coincidono col linguaggio osservato, ma ci sono valori degli elementi di $M_0, Post$ e Pre dell'ordine di 10^{15}-10^{20}.

2. La soluzione trovata è ottimale con valori elevatissimi, ma i linguaggi generati differiscono dal linguaggio osservato.

3. Compare un messaggio di errore di GLPK ("***SEVERE CRITICAL ERROR*** from GLPK!") che indica allocazione di memoria fallita.

Le anomalie di cui ai punti 1 e 2 nascondono il seguente problema: GLPK talvolta ricava soluzioni razionali contenenti valori negativi, nonostante sia imposto il limite inferiore delle variabili correttamente a 0, che in seguito all'arrotondamento assumono i valori indicati sopra.

La spiegazione di queste anomalie sta nella non totale affidabilità dell'implementazione della versione 4.9 di GLPK per PPL con più di 100.000 vincoli e in qualche caso anche per PPL di complessità inferiore (è l'ultimo caso dell'esempio del processo lettori-scrittori, per $k = 5$), come riportato nel manuale di GLPK.

Limiti di affidabilità del risolutore di GLPK basato sul metodo del punto interno.

Il risolutore del punto interno trova raramente una soluzione ottimale e la trova solo per valori di k di poche unità. In tutti gli altri casi conduce ad instabilità numerica. Il motivo di ciò sta nella scarsa affidabilità del risolutore di GLPK. Infatti in questa versione di GLPK, nonostante sia la più evoluta, il risolutore del punto interno non include molte importanti caratteristiche, in particolare:

- Non è capace di elaborare colonne dense. Questo problema viene comunque evitato nel momento in cui sto definendo la matrice dei vincoli in forma sparsa.
- Non ha caratteristiche contro l'instabilità numerica. Questo è stato ampiamente verificato dai test.
- Non è capace di identificare la base ottimale, che corrisponde alla soluzione del punto interno trovata.

Limiti di affidabilità del risolutore di GLPK basato sul metodo del *branch-and-bound*.

Il risolutore del *branch-and-bound* trova sempre una soluzione ottimale con il limite temporale fissato a 2 ore. Infatti se avessi aumentato questo limite, in qualche caso sarebbe potuto comparire il messaggio di errore di GLPK ("***SEVERE CRITICAL ERROR*** from GLPK!") che indica problemi di memoria. Anche qui la spiegazione sta nella non totale affidabilità del risolutore di GLPK che è incapace talvolta di trattare problemi con più di 200 variabili intere, come riportato nel manuale di GLPK.

Variazione del numero di stringhe del linguaggio osservato.

Generalmente il numero di stringhe del linguaggio aumenta più velocemente al crescere di k; talvolta aumenta in maniera esponenziale. Per particolari strutture di rete parametriche, come abbiamo visto negli esempi, può succedere che aumentando i valori dei parametri il linguaggio rimanga invariato, cioè la rete non genera più nuove sequenze di scatto, o addiritura il numero di stringhe diminuisca.

Complessità del PPL.

La complessità del PPL, indicata in maniera più diretta dalle dimensioni della matrice A, di solito aumenta più velocemente al crescere di k. In particolare il numero di vincoli (numero di righe di A) aumenta ad una tasso superiore, talvolta esponenzialmente, rispetto al numero di variabili (numero di colonne di A).

Variazione del numero di posti.

Il numero di posti, a differenza del linguaggio, non aumenta sensibilmente se non in qualche caso, ma lo fa in maniera graduale, quasi lineare, al crescere di k.

La tecnica di pre-riduzione riduce il numero di posti a volte sensibilmente di qualche centinaio di unità, altre volte in modo meno evidente. La sua efficacia, quindi, dipende fortemente dal sistema di rete ed è tanto più evidente quante più transizioni con un solo posto in ingresso e in relazione di scelta libera sono presenti nella rete.

D'altra parte la tecnica di post-riduzione si è rivelata molto efficace poiché spesso estrae una rete contenente un numero di posti di poche unità, indipendentemente dall'applicazione della pre-riduzione. Quindi la pre-riduzione non è determinante per ricavare una rete finale con pochi posti, ma è utile per semplificare la risoluzione del problema di identificazione. Se è dato un sistema di rete da testare, la post-riduzione riesce ad estrarre una rete avente una struttura in qualche modo simile o molto simile a quella di partenza, anche se non esattamente la stessa.

Tempo di risoluzione del PPL.

In linea di massima il tempo di risoluzione del PPL non aumenta sensibilmente al crescere di k se non in qualche caso, anzi, talvolta aumenta molto lentamente con crescite dell'ordine dei secondi. In assoluto i tempi di risoluzione del PPL assumono valori non più grandi dell'ordine del minuto per qualsiasi valore di k, e si parla di pochi secondi per valori di k dell'ordine delle unità. Occorre sottolineare che i dati sui tempi sono relativi al PC su cui sono stati effettuati i test, quindi i valori cambiano in base alle prestazioni della macchina su cui vengono eseguiti. La pre-riduzione consente spesso di trarre notevoli risparmi di tempo: nei casi più fortunati può succedere che un problema che viene risolto in pochi minuti senza la pre-riduzione venga risolto in pochi secondi applicando una pre-riduzione. La differenza tra i due metodi del simplesso, revisionato e duale, è al massimo dell'ordine dei minuti e va a favore del simplesso duale. Non è stato possibile fare un confronto ragionevole col metodo del punto interno data la scarsa affidabilità del risolutore del punto interno.

Tempo di esecuzione dell'intera procedura.

Generalmente il tempo di esecuzione della procedura di identificazione completa aumenta più velocemente al crescere di k, talvolta esponenzialmente. Il valore assoluto dei tempi di esecuzione è fortemente legato al tipo di sistema di rete e naturalmente alla macchina su cui vengono effettuati i test. Può capitare di

eseguire la procedura in meno di 2 ore anche per valori di k dell'ordine delle decine o di dover attendere delle ore prima di visualizzare l'output anche per valori ad esempio attorno a 10-20. La pre-riduzione fa risparmiare tempo ovviamente e la sua incidenza è legata al sistema di rete. Quindi talvolta consente notevoli vantaggi con riduzioni dell'ordine dei minuti senza andare oltre comunque, altre volte i vantaggi sono meno evidenti e si parla di guadagni inferiori al minuto. Anche qui la differenza tra i due metodi del simplesso è minima a vantaggio del simplesso duale e inoltre non abbiamo dati sufficienti per effettuare un confronto col metodo del punto interno.

Nonostante non siano stati ricavati i dati sui tempi di esecuzione dei singoli programmi per motivi di brevità, ritengo utile dare almeno un'indicazione, anche se approssimativa, sulla loro incidenza percentuale sul tempo di esecuzione dell'intera procedura. In ordine crescente di efficienza, dal programma più lento a quello più veloce, abbiamo la seguente classificazione, che comunque può variare in qualche posizione dipendentemente dal tipo di rete:

1. `maxL_n_k2E_D` (programma che costruisce $\mathcal{E}$ e $\mathcal{D}$): generalmente incide per circa il 55-60% sul tempo totale.
2. `E_D_pre_red2CS` (programma che costruisce l'insieme dei vincoli del PPL): generalmente incide per circa il 25-30%.
3. `PN_D2post_red_PN` (programma che estrae la rete ridotta con una post-riduzione): generalmente incide per circa il 15-20%.
4. `PN_k2maxL_n` (programma che calcola il linguaggio): generalmente incide al massimo l'1-2%.
5. `CS_infoPN2PN` (programma che estrae la rete soluzione del PPL): generalmente incide al massimo l'1%.

Programmazione lineare vs. programmazione intera.

Se avessi utilizzato la programmazione intera per identificare la rete avrei sicuramente avuto tempi di risoluzione del PPL molto più alti, si sarebbe parlato di minuti per i più bassi valori di k e di decine di minuti, se non di ore, per problemi più complessi.

Per quanto riguarda il tempo di esecuzione dell'intera procedura in [12] è stata analizzata la complessità computazionale della procedura di identificazione basata sulla programmazione intera per la rete del protocollo di comunicazione. In

questo lavoro non si parla di post-riduzione, quindi mi riferisco alla procedura di base illustrata nel capitolo 4. La soluzione ottimale è stata ricavata in un'ora solo per $s = 2$ e per $s = 3$ (in [12] s corrisponde a q). In tutti gli altri casi il numero di vincoli era troppo alto e un'ora non è stata sufficiente per determinare la soluzione ottimale. Invece, se osserviamo la Tabella 7.12 dello stesso esempio riportato nel capitolo 7, notiamo che i tempi di computazione per $s = 2$ e $s = 3$ utilizzando la programmazione lineare sono attestati attorno a valori bassissimi, si parla infatti di un secondo circa o anche meno, mentre l'ora di tempo viene raggiunta per $s = 5$ e $s = 6$. Ricordo che in questo caso si parla di procedura completa comprendente quindi anche la post-riduzione, ad ogni modo il confronto tra le due tecniche di programmazione è attendibile. In quest'esempio si è dimostrato quanto ci si aspettava e si può generalizzare il risultato ottenuto. Dunque, la programmazione intera ha una complessità computazionale elevata essenzialmente perché il numero di variabili del problema di programmazione aumenta esponenzialemnte con k. Se il sistema di rete è complesso, come è il caso del protocollo di comunicazione, il problema si complica notevolmente già per valori di k di poche unità. D'altra parte, la programmazione lineare riduce drasticamente la complessità computazionale e consente di gestire senza troppe difficoltà il problema di programmazione sino a valori di k di qualche decina.

Limiti di applicabilità della procedura dal punto di vista della memoria utilizzata da Matlab.

È naturale che al crescere della complessità del problema anche la memoria richiesta da Matlab per allocare le variabili cresca di pari passo. Per valori di k tra i 10 e 30, dipendentemente dal sistema di rete, Matlab dà problemi di memoria e sullo schermo compare un messaggio di "Out of memory" che indica insufficienza di memoria, cioè l'impossibilità di allocare nuove variabili. Il motivo principale dell'overflow di Matlab è legato alle dimensioni della matrice A dei coefficienti dei vincoli che, nonostante venga memorizzata in forma sparsa, per problemi più complessi richiede una memoria dell'ordine delle decine di MB. La pre-riduzione consente di ritardare il verificarsi del problema di qualche valore di k, sempre in maniera dipendente dal tipo di sistema di rete.

Capitolo 8

Conclusioni

In questa tesi è stato creato un toolbox in Matlab per l'identificazione di una rete posto/transizione (P/T) basato sulla programmazione lineare. L'utilizzo di tecniche di programmazione lineare è un vantaggio rilevante dal punto di vista della complessità computazionale rispetto a precedenti approcci che erano basati sulla programmazione intera.

La procedura di identificazione prende in ingresso un linguaggio $\mathcal{L}$ finito chiuso per prefisso e un intero k maggiore o uguale alla lunghezza della stringa più lunga in esso contenuta, e fornisce in uscita una rete il cui linguaggio delle parole generate di lunghezza non superiore a k coincide con $\mathcal{L}$.

La procedura è stata eseguita nel seguente modo:

1. A partire da $\mathcal{L}$ costruisce gli insiemi delle condizioni di abilitazione $\mathcal{E}$ e di disabilitazione $\mathcal{D}$.
2. Da questi estrae un insieme di vincoli lineari, che è stato chiamato *Constraint Set*, le cui incognite sono gli elementi della marcatura iniziale M_0 e delle matrici $Post$ e Pre. Il numero di posti della rete è inizialmente pari alla cardinalità di $\mathcal{D}$ o, eventualmente, minore se sono presenti coppie $(\vec{\sigma}, \vec{\sigma} + \vec{t})$ uguali, relative agli elementi di $\mathcal{D}$. Esso può essere ridotto in base ad informazioni aggiuntive sulla struttura della rete, esattamente nel caso in cui ci siano transizioni con un solo posto in ingresso e transizioni in relazione di scelta libera (tecnica di pre-riduzione).
3. All'insieme di vincoli così costruito viene associata una funzione obiettivo che minimizza la somma dei gettoni nella marcatura iniziale e dei pesi degli

archi. Il risultante problema di programmazione lineare (PPL) viene risolto, con l'ausilio della libreria software GLPK, determinando la rete con marcatura iniziale M_0 e struttura $Post$ e Pre che genera il linguaggio dato $\mathcal{L}$. Inoltre è stato anche considerato il problema di sintetizzare la rete quando sono date altre informazioni strutturali su di essa (P-vettori, T-vettori, classe di appartenenza) o GMEC, che consentono di aggiungere nuovi vincoli al CS.

4. Il numero di posti della rete risultante può non essere minimo, perciò i posti ridondanti possono essere rimossi (tecnica di post-riduzione) senza modificare il linguaggio generato.

Le funzioni Matlab che costituiscono lo strumento software eseguono automaticamente i vari passaggi sopra descritti. La funzionalità dei programmi e la correttezza della procedura sono state testate su varie reti, in particolare su tre reti tratte da diversi campi applicativi (sistemi operativi, sistemi di comunicazione): il problema dei filosofi a tavola (deadlock tra processi che utilizzano risorse condivise), il processo lettori-scrittori e il processo emittente-ricevente.

I dati più importanti ricavati dai test riguardano:

- Limiti di affidabilità dei risolutori di GLPK.
- Linguaggi generati dalla rete soluzione del PPL e dalla rete ridotta con una post-riduzione.
- Complessità del PPL.
- Numero di posti della rete soluzione del PPL e della rete ridotta.
- Tempo di risoluzione del PPL.
- Tempo di esecuzione dell'intera procedura.
- Limiti di applicabilità della procedura dal punto di vista della memoria utilizzata da Matlab.

In sintesi, i risultati fondamentali ricavati dai test sono i seguenti:

- Nella stragrande maggioranza dei casi il risolutore di GLPK basato sul metodo del simplesso trova soluzioni ottimali e corrette nel senso che la rete soluzione del PPL genera il linguaggio osservato. Tale risolutore non è totalmente affidabile per PPL con più di 100.000 vincoli.

- Il risolutore di GLPK basato sul metodo del punto interno trova raramente una soluzione ottimale e la trova solo per valori di k di poche unità. In tutti gli altri casi conduce ad instabilità numerica.
- La complessità del PPL aumenta più velocemente man mano che cresce k, generalmente ad un tasso inferiore rispetto a quello esponenziale.
- L'efficacia della tecnica di pre-riduzione dipende fortemente dal tipo di rete. Questa tecnica è utile per semplificare il problema di identificazione. La tecnica di post-riduzione si è rivelata molto efficace poiché spesso estrae una rete contenente un numero di posti di poche unità, indipendentemente dall'applicazione della pre-riduzione. Se è data una rete da testare, la post-riduzione riesce ad estrarre una rete avente una struttura simile a quella di partenza.
- Generalmente il tempo di esecuzione dell'intera procedura aumenta più velocemente al crescere di k. Il valore assoluto dei tempi di esecuzione è fortemente legato al tipo di rete. La differenza tra i due metodi del simplesso, revisionato e duale, è minima a vantaggio del simplesso duale.
- La programmazione lineare riduce drasticamente la complessità computazionale rispetto alla programmazione intera e consente di gestire senza troppe difficoltà il problema di identificazione sino a valori di k di qualche decina.

Come possibile linea guida per la ricerca futura, si può considerare l'estensione di questa procedura a casi più generali, come quelli basati sulla conoscenza dell'albero o grafo di raggiungibilità/copertura della rete da identificare.

Appendice A

Listati dei programmi

A.1 Costruzione della rete P/T del protocollo di comunicazione

```
function [M0,Post,Pre]=com_prot_PN(m0,s)
% Questa funzione crea la rete posto/transizione che
% modella un protocollo di comunicazione in funzione del
% numero di posti e del numero iniziale di gettoni
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
%                      ## SINTASSI ##
%            [M0,Post,Pre]=com_prot_PN(m0,s)
%                 ARGOMENTI DI INGRESSO:
%  m0: intero positivo che fornisce il numero iniziale
%      di gettoni
%  s: intero maggiore di 1 che definisce il numero di
%     posti (2*s+2) e anche il numero di transizioni (2*s)
%                 ARGOMENTI DI USCITA:
%  M0: vettore colonna marcatura iniziale di lunghezza m
%      in cui ciascuna componente indica il numero di
%      gettoni contenuti in un posto
%  Post: matrice detta funzione di post-incidenza in cui
%        ciascun elemento Post(p,t) indica quanti archi
%        vanno dalla transizione t al posto p;
%        la dimensione è m x n
%  Pre: matrice detta funzione di pre-incidenza in cui
%       ciascun elemento Pre(p,t) indica quanti archi
```

```
%       vanno dal posto p alla transizione t;
%       la dimensione è m x n
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
% Copyright 2008 by Pierandrea Secchi. Rev.: 15 luglio 2008

nargin=2;
% numero di argomenti di ingresso della funzione
nargout=3;
% numero di argomenti di uscita della funzione
M0=[m0 zeros(s-1,1)' m0 zeros(s+1,1)']';
% solo M0 dipende da m0

P1=[zeros(s-1,1)' 1 zeros(s,1)'];
P2=eye(2*s);
P2(s,s)=0;
P2(s,2*s)=1;
P2(2*s,1)=1;
P2(2*s,2*s)=0;
Post=[P1;P2;zeros(2*s-1,1)' 1];

Pre=[eye(2*s);P2(s+1,:);P1];
% Post e Pre dipendono solo da s
```

A.2 Costruzione della rete P/T del processo lettori-scrittori

```
function [M0,Post,Pre]=read_writ_proc_PN(l,n,s)
% Questa funzione crea la rete posto/transizione che
% modella un processo lettori-scrittori noti il numero di
% lettori e di scrittori e il numero massimo di lettori
% che possono leggere contemporaneamente
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
%                     ## SINTASSI ##
%         [M0,Post,Pre]=read_writ_proc_PN(l,n,s)
%                 ARGOMENTI DI INGRESSO:
%  l: numero di lettori
%  n: numero massimo di lettori che possono leggere
%     contemporaneamente
%  s: numero di scrittori
%                 ARGOMENTI DI USCITA:
```

```
%  M0: vettore colonna marcatura iniziale di lunghezza m
%      in cui ciascuna componente indica il numero di
%      gettoni contenuti in un posto
%  Post: matrice detta funzione di post-incidenza in cui
%        ciascun elemento Post(p,t) indica quanti archi
%        vanno dalla transizione t al posto p;
%        la dimensione è m x n
%  Pre: matrice detta funzione di pre-incidenza in cui
%       ciascun elemento Pre(p,t) indica quanti archi
%       vanno dal posto p alla transizione t;
%       la dimensione è m x n
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
% Copyright 2008 by Pierandrea Secchi. Rev.: 15 luglio 2008

nargin=3;
% numero di argomenti di ingresso della funzione
nargout=3;
% numero di argomenti di uscita della funzione
M0=[l 0 n s 0]';
% solo M0 dipende da l ed s

Post=[0 1 0 0;1 0 0 0;0 1 0 n;0 0 0 1;0 0 1 0];

Pre=[1 0 0 0;0 1 0 0;1 0 n 0;0 0 1 0;0 0 0 1];
% Post e Pre dipendono solo da n
```

A.3 Calcolo del linguaggio massimale

```
function [maxL,n]=PN_k2maxL_n(M0,Post,Pre,k)
% Questa funzione ricava il linguaggio massimale finito
% chiuso per prefisso di una rete posto/transizione,
% costituito da sequenze di scatto di pari lunghezza,
% minore o uguale a k, e il numero di transizioni a
% partire dalla struttura della rete
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
%                     ## SINTASSI ##
%          [maxL,n]=PN_k2maxL_n(M0,Post,Pre,k)
%                ARGOMENTI DI INGRESSO:
%  M0: vettore colonna marcatura iniziale di lunghezza m
%      in cui ciascuna componente indica il numero di
%      gettoni contenuti in un posto
```

```
%  Post: matrice detta funzione di post-incidenza in cui
%        ciascun elemento Post(p,t) indica quanti archi
%        vanno dalla transizione t al posto p;
%        la dimensione è m x n
%  Pre: matrice detta funzione di pre-incidenza in cui
%       ciascun elemento Pre(p,t) indica quanti archi
%       vanno dal posto p alla transizione t;
%       la dimensione è m x n
%  k: intero naturale maggiore o uguale alla lunghezza
%     delle stringhe del linguaggio massimale
%                 ARGOMENTI DI USCITA:
%  maxL: matrice delle sequenze di scatto aventi la stessa
%        lunghezza, la più grande possibile minore o uguale
%        a k, dove ogni riga contiene gli indici i delle
%        transizioni t_i che compongono la generica
%        sequenza di scatto sigma, disposte nell'ordine in
%        cui compaiono in tale sequenza;
%        la dimensione è numRows x numCol
%  n: numero di transizioni
%  * * * * * * * * * * * * * * * * * * * * * * * * * * *
% Copyright 2008 by Pierandrea Secchi. Rev.: 15 luglio 2008

nargin=4;
% numero di argomenti di ingresso della funzione
nargout=2;
% numero di argomenti di uscita della funzione
n=size(Post,2);
% size restituisce il numero di colonne di Post (e Pre)
% come una variabile separata

if k==0
   maxL=[];
% se k=0 maxL contiene la sola sequenza vuota
else maxL=[];
     C=Post-Pre;
% C è la matrice di incidenza
     count=0;
     for i=1:n
         if M0>=Pre(:,i)
            maxL=[maxL;i];
            count=count+1;
            M(:,count)=M0+C(:,i);
         end
```

```
     end
% il ciclo for calcola le sequenze di lunghezza unitaria
     for j=2:k
         maxL=[maxL zeros(count,1)];
% la variabile count memorizza il numero di
% sequenze di lunghezza j-1
         [numRows,j]=size(maxL);
         newcount=0;
% la variabile newcount memorizza il numero di
% nuove sequenze di lunghezza j
         for h=1:count
           for i=1:n
             if M(:,h)>=Pre(:,i)
                newcount=newcount+1;
                newM(:,newcount)=M(:,h)+C(:,i);
                maxL=[maxL;maxL(numRows-count+h,1:j-1) i];
             end
           end
         end
% il ciclo for,per ogni marcatura raggiunta dallo scatto
% di una determinata sequenza di lunghezza j-1, conta il
% numero di nuove sequenze abilitate, calcola le nuove
% marcature raggiunte e le nuove sequenze di lunghezza j
         if newcount==0
            maxL(:,j)=[];
            break
% se non ci sono nuove sequenze abilitate viene eliminata
% l'ultima colonna di maxL e il programma termina
         else maxL(1:count,:)=[];
              count=newcount;
              M=newM;
         end
     end
end
```

A.4 Costruzione degli insiemi $\mathcal{E}$ e $\mathcal{D}$

```
function [E,D]=maxL_n_k2E_D(maxL,n,k)
% Questa funzione ricava gli insiemi delle condizioni di
% abilitazione e di disabilitazione a partire dal
% linguaggio massimale finito chiuso per prefisso di una
```

```
% rete posto/transizione, definito su un alfabeto T,
% insieme di n transizioni, e costituito da sequenze
% di scatto di pari lunghezza, minore o uguale a k
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
%                     ## SINTASSI ##
%              [E,D]=maxL_n_k2E_D(maxL,n,k)
%                 ARGOMENTI DI INGRESSO:
%  maxL: matrice delle sequenze di scatto aventi la stessa
%        lunghezza, la più grande possibile minore o uguale
%        a k, dove ogni riga contiene gli indici i delle
%        transizioni t_i che compongono la generica
%        sequenza di scatto sigma, disposte nell'ordine
%        in cui compaiono in tale sequenza;
%        la dimensione è numRows x numCol
%  n: numero di transizioni
%  k: intero naturale maggiore o uguale alla lunghezza
%     delle stringhe del linguaggio massimale
%                 ARGOMENTI DI USCITA:
%  E: matrice delle condizioni di abilitazione dove ogni
%     riga contiene gli indici i delle transizioni t_i
%     nella condizione (sigma,t), disposte nell'ordine in
%     cui compaiono in tale condizione;
%     la sua dimensione è card_E x numCol_E
%  D: matrice delle condizioni di disabilitazione dove ogni
%     riga contiene gli indici i delle transizioni t_i
%     nella condizione (sigma,t), disposte nell'ordine in
%     cui compaiono in tale condizione;
%     la sua dimensione è card_D x numCol_D
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
% Copyright 2008 by Pierandrea Secchi. Rev.: 15 luglio 2008

nargin=3;
% numero di argomenti di ingresso della funzione
nargout=2;
% numero di argomenti di uscita della funzione
numCol=size(maxL,2);
% size restituisce il numero di colonne di maxL
% come una variabile separata
E=maxL;
count=0;
% la variabile count fornisce la dimensione di un vettore
% di zeri che aumenta di un'unità ad ogni iterazione
% del ciclo while successivo
```

```
while numCol>1
      count=count+1;
      maxL(:,numCol)=[];
      [numRows,numCol]=size(maxL);
      while numRows>1
            for i=1:numRows-1
                if maxL(i,:)==maxL(numRows,:)
                   numRows=i;
                   break
                end
            end
% il ciclo for scandisce le righe della matrice maxL
% corrente finchè non ne trova una uguale all'ultima e la
% definisce come nuova ultima riga prima di terminare
            if numRows==1
               break
            else E=[maxL(numRows,:) zeros(1,count);E];
                 numRows=numRows-1;
            end
      end
% il ciclo while consente di ripetere il ciclo for di cui
% sopra aggiungendo ad E ad ogni iterazione l'ultima riga
% corrente finchè non rimane una sola riga da esaminare
      E=[maxL(1,:) zeros(1,count);E];
end
% il ciclo while itera il procedimento dopo aver
% eliminato una colonna di maxL

[card_E,numCol_E]=size(E);
cntzer=0;
while E(cntzer+1,numCol_E)==0
      cntzer=cntzer+1;
end
maxL=E(cntzer+1:card_E,:);
[numRows,numCol]=size(maxL);
% il ciclo while recupera la dimensione originale della
% maxL d'ingresso, che coincide con le ultime card_E-cntzer
% righe di E, per estrarre la matrice D
D=[];
count=0;
if k>numCol
   D=zeros(numRows*n,k);
   for i=1:numRows
```

```
       for j=1:n
           D((i-1)*n+j,:)=[maxL(i,:) j];
       end
   end
   count=1;
end
% i cicli for aggiungono a D un numero di righe pari
% al numero di righe di maxL per il numero di transizioni
% solo nel caso in cui k sia maggiore
% del numero di colonne di maxL
while numCol>1
  for i=1:numRows
    if maxL(i,1:numCol-1)==maxL(numRows,1:numCol-1)
      j=n;
      while j>maxL(numRows,numCol)
          D=[maxL(numRows,1:numCol-1) j zeros(1,count);D];
          j=j-1;
      end
      j=numRows;
      while j>i
        if maxL(j,numCol)-maxL(j-1,numCol)>1
         j1=maxL(j,numCol)-1;
         while j1>maxL(j-1,numCol)
          D=[maxL(numRows,1:numCol-1) j1 zeros(1,count);D];
          j1=j1-1;
         end
        end
        j=j-1;
      end
      j=maxL(i,numCol);
      while j>1
            D=[maxL(i,1:numCol-1) j-1 zeros(1,count);D];
            j=j-1;
      end
      break
    end
  end
% il ciclo for, appena trova una riga di maxL uguale
% all'ultima, aggiunge a D delle righe uguali alle due
% messe a confronto eccetto per l'ultimo elemento; il
% numero delle righe aggiunte è pari al numero di ultimi
% elementi mancanti tra quelli compresi fra 1 ed n
      numRows=i;
```

```
      if numRows==1
         count=count+1;
         maxL(:,numCol)=[];
         [numRows,numCol]=size(maxL);
      else numRows=numRows-1;
      end
end
% il ciclo while ad ogni iterazione ripete le operazioni
% di cui sopra finchè non si arriva a confrontare
% la prima riga della matrice maxL corrente;
% dopodichè maxL ad ogni passo viene ridotta di una colonna
j=n;
while j>maxL(numRows,1)
      D=[j zeros(1,count);D];
      j=j-1;
end
j=numRows;
while j>1
      if maxL(j,1)-maxL(j-1,1)>1
         j1=maxL(j,1)-1;
         while j1>maxL(j-1,1)
               D=[j1 zeros(1,count);D];
               j1=j1-1;
         end
      end
      j=j-1;
end
j=maxL(1,1);
while j>1
      D=[j-1 zeros(1, count);D];
      j=j-1;
end
% i cicli while, quando maxL ridotta ha una sola colonna,
% aggiungono a D delle righe aventi un solo elemento non
% nullo, il primo, di valore diverso da quelli contenuti
% nell'unica colonna di maxL e comunque compreso tra 1 ed n
```

A.5 Costruzione del *Constraint Set* (CS)

```
function [q,A,b,ctype,lb,vartype]=...
 E_D_pre_red2CS(E,D,pre_red)
```

```
% Questa funzione ricava la struttura del Constraint
% Set(CS), il sistema di equazioni dei vincoli nel problema
% di programmazione lineare la cui soluzione identifica una
% rete posto/transizione, a partire dagli insiemi delle
% condizioni di abilitazione e di disabilitazione e da
% alcune informazioni aggiuntive (opzionali) sulla rete che
% consentono di ridurre il numero di posti
% (pre-riduzione dei posti)
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * *
%                     ## SINTASSI ##
%   [q,A,b,ctype,lb,vartype]=E_D_pre_red2CS(E,D,pre_red)
%                  ARGOMENTI DI INGRESSO:
%  E: matrice delle condizioni di abilitazione dove ogni
%     riga contiene gli indici i delle transizioni t_i
%     nella condizione (sigma,t), disposte nell'ordine in
%     cui compaiono in tale condizione;
%     la sua dimensione è card_E x numCol_E
%  D: matrice delle condizioni di disabilitazione dove ogni
%     riga contiene gli indici i delle transizioni t_i
%     nella condizione (sigma,t), disposte nell'ordine in
%     cui compaiono in tale condizione;
%     la sua dimensione è card_D x numCol_D
%  pre_red: matrice vuota se non ci sono informazioni
%           aggiuntive che consentono di scrivere il CS in
%           una forma modificata usando un numero ridotto
%           di posti (pre-riduzione dei posti) o matrice
%           non vuota se son presenti tali informazioni; in
%           quest'ultimo caso le righe con un solo elemento
%           non nullo, che vale i indicano che la
%           transizione i ha solo un posto in ingresso
%           mentre le altre righe, se esistono, indicano
%           che i valori degli elementi non nulli
%           corrispondono a transizioni
%           in una relazione di scelta libera
%                  ARGOMENTI DI USCITA:
%  q: variabile che memorizza il numero di blocchi di una
%     partizione ammissibile dell'insieme D e rappresenta
%     il numero di posti della rete risolvente
%     il problema di identificazione
%  A: matrice sparsa contenente i coefficienti dei vincoli;
%     la sua dimensione è (nv+sigm_D) x xm
%     o (nv+sigm_D) x (q*(2n+1))
%  b: vettore colonna di lunghezza nv+sigm_D contenente il
```

```
%     termine noto per ogni vincolo
%     nella matrice dei vincoli
%  ctype: vettore colonna di lunghezza nv+sigm_D contenente
%         il senso di ogni vincolo
%  lb: vettore di lunghezza xm contenente il limite
%      inferiore di ciascuna delle variabili
%      incognite del problema di identificazione
%  vartype: vettore colonna di lunghezza xm contenente la
%           definizione del tipo di variabili (continue
%           per un problema di programmazione lineare)
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * *
% Copyright 2008 by Pierandrea Secchi. Rev.: 15 luglio 2008

nargin=3;
% numero di argomenti di ingresso della funzione
nargout=6;
% numero di argomenti di uscita della funzione
nE=max(E);
nE2=max(nE);
nD=max(D);
nD2=max(nD);
n=max(nE2,nD2);
% n è il numero di transizioni
T=eye(n);
[card_E,numCol_E]=size(E);
[card_D,numCol_D]=size(D);
% size restituisce il numero di righe di E e D
% come variabili separate
if numCol_E==numCol_D
   ED=[E;D];
elseif numCol_E<numCol_D
       ED=[E zeros(card_E,1);D];
end
% E e D vengono riunite in un'unica grande matrice ED
card_ED=card_E+card_D;
global j;
j=zeros(card_ED,1);
global sigma;
% le variabili j e sigma sono dichiarate globali perchè
% verranno riutilizzate nella funzione PN_D2post_red_PN
sigma=zeros(card_ED,n);
t=zeros(card_ED,n);
sigma_plus_t=zeros(card_ED,n);
```

```
for i=1:card_ED
    k1=find(ED(i,:)~=0);
    j(i)=k1(end);
    for p=1:j(i)-1
        sigma(i,ED(i,p))=sigma(i,ED(i,p))+1;
    end
    t(i,:)=T(ED(i,j(i)),:);
    sigma_plus_t(i,:)=sigma(i,:)+t(i,:);
end
% il ciclo for esterno restituisce l'indice dell'ultimo
% elemento non nullo in ogni riga di E e D, cioé la
% posizione della transizione t della generica condizione
% (sigma,t); i cicli for interni restituiscono i vettori di
% scatto sigma(i,1:n), associati alle rispettive sequenze
% di scatto ED(:,1:j(i)-1), i vettori t della generica
% condizione (sigma,t), che hanno l'i-esima componente
% corrispondente alla transizione t-i pari a 1 mentre le
% altre componenti sono nulle,
% e i vettori sigma+t del problema
q=card_D;
if isempty(pre_red)==0
 [numRows_pre_red,numCol_pre_red]=size(pre_red);
 if numCol_pre_red==1
    pred=length(pre_red);
 else pred=0;
      while pre_red(pred+1,2)==0
            pred=pred+1;
      end
 end
% la variabile pred memorizza il numero di righe di pre_red
% aventi un solo elemento non nullo
 nelbl=[];
 nel=0;
 sigma_D1=[];
 sigma_plus_t_D1=[];
 sigma_D2=[];
 sigma_plus_t_D2=[];
 sigma_D3=[];
 sigma_plus_t_D3=[];
 m=numRows_pre_red-pred;
% la variabile m memorizza il numero di righe
% aventi più di un elemento non nullo
 nelbl1=zeros(1,m);
```

```
 for i=1:n
  [i1,i2]=find(pre_red==i);
  if length(i1)==1
   sigma1=[];
   sigma_plus_t1=[];
   count=0;
   for j1=card_E+1:card_ED
       if ED(j1,j(j1))==i
          sigma1=[sigma1;sigma(j1,:)];
          sigma_plus_t1=[sigma_plus_t1;sigma_plus_t(j1,:)];
          count=count+1;
       end
   end
% il ciclo for memorizza nelle matrici sigma1 e
% sigma_plus_t1 i vettori sigma(j1,:) e sigma_plus_t(j1,:)
% corrispondenti a condizioni di disabilitazione che
% disabilitano la transizione t_i che ha un solo posto in
% ingresso e non è in relazione di scelta libera con
% altre transizioni; il numero di righe delle matrici
% è memorizzato nella variabile count
   q=q-count+1;
   sigm1=size(sigma1,1);
   count=0;
   while sigm1>=1
      sigma_D1=[sigma_D1;sigma1(1,:)];
      sigma_plus_t_D1=[sigma_plus_t_D1;sigma_plus_t1(1,:)];
      count=count+1;
      r=1;
      while r<=sigm1-1
            if sigma1(1+r,:)==sigma1(1,:)
               sigma1(1+r,:)=[];
               sigma_plus_t1(1+r,:)=[];
               sigm1=size(sigma1,1);
            else r=r+1;
            end
      end
% il ciclo while elimina le coppie
% sigma1(i,:)-sigma_plus_t1(i,:) uguali, all'interno di un
% blocco associato ad una transizione che ha un solo posto
% in ingresso e non è in relazione di scelta libera con
% altre transizioni, che altrimenti genererebbero
% vincoli uguali nel CS modificato
      sigma1(1,:)=[];
```

```
      sigma_plus_t1(1,:)=[];
      sigm1=size(sigma1,1);
   end
% il ciclo while memorizza nelle matrici sigma_D1 e
% sigma_plus_t_D1 le differenti coppie
% sigma1(i,:)-sigma_plus_t1(i,:); il numero di coppie
% è memorizzato nella variabile count
   nelbl=[nelbl count];
% nelbl è un vettore riga in cui ciascuna componente
% rappresenta il numero di elementi, memorizzato nella
% variabile count ad ogni iterazione del ciclo for,
% contenuti in un blocco di una partizione ammissibile di D
% nel caso in cui ci siano transizioni che hanno solo
% un posto in ingresso e non sono in
% relazione di scelta libera con altre transizioni
  elseif length(i1)==0
   sigma2=[];
   sigma_plus_t2=[];
   for j1=card_E+1:card_ED
       if ED(j1,j(j1))==i
          sigma2=[sigma2;sigma(j1,:)];
          sigma_plus_t2=[sigma_plus_t2;sigma_plus_t(j1,:)];
       end
   end
% il ciclo for memorizza nelle matrici sigma2 e
% sigma_plus_t2 i vettori sigma(j1,:) e sigma_plus_t(j1,:)
% corrispondenti a condizioni di disabilitazione
% che disabilitano la transizione t_i
% che non ha un solo posto in ingresso
   sigm2=size(sigma2,1);
   while sigm2>=1
      sigma_D2=[sigma_D2;sigma2(1,:)];
      sigma_plus_t_D2=[sigma_plus_t_D2;sigma_plus_t2(1,:)];
      r=1;
      while r<=sigm2-1
            if sigma2(1+r,:)==sigma2(1,:)
               q=q-1;
% ad ogni ciclo while interno il numero di posti q
% viene decrementato quando viene rilevata
% una coppia sigma2(i,:)-sigma_plus_t2(i,:) uguale
               sigma2(1+r,:)=[];
               sigma_plus_t2(1+r,:)=[];
               sigm2=size(sigma2,1);
```

```
            else r=r+1;
            end
      end
      nel=nel+1;
% la variabile nel memorizza il numero di blocchi nel caso
% in cui ci siano transizioni che non hanno
% un solo posto in ingresso; vengono considerati
% solo i blocchi relativi a quelle transizioni
      sigma2(1,:)=[];
      sigma_plus_t2(1,:)=[];
      sigm2=size(sigma2,1);
   end
% il ciclo while memorizza nelle matrici
% sigma_D2 e sigma_plus_t_D2 le differenti
% coppie sigma2(i,:)-sigma_plus_t2(i,:)
  else sigma3=[];
   sigma_plus_t3=[];
   count=0;
   for j1=card_E+1:card_ED
       if ED(j1,j(j1))==i
          sigma3=[sigma3;sigma(j1,:)];
          sigma_plus_t3=[sigma_plus_t3;sigma_plus_t(j1,:)];
          count=count+1;
       end
   end
% il ciclo for memorizza nelle matrici sigma3 e
% sigma_plus_t3 i vettori sigma(j1,:) e sigma_plus_t(j1,:)
% corrispondenti a condizioni di disabilitazione che
% disabilitano la transizione t_i che è in relazione di
% scelta libera con altre transizioni, cioé esiste un posto
% che è l'unico posto in ingresso per quelle transizioni;
% il numero di righe delle matrici
% è memorizzato nella variabile count
   q=q-count;
   sigm3=size(sigma3,1);
   count=0;
   while sigm3>=1
      sigma_D3=[sigma_D3;sigma3(1,:)];
      sigma_plus_t_D3=[sigma_plus_t_D3;sigma_plus_t3(1,:)];
      count=count+1;
      r=1;
      while r<=sigm3-1
            if sigma3(1+r,:)==sigma3(1,:)
```

```
% il ciclo while elimina le coppie
% sigma3(i,:)-sigma_plus_t3(i,:) uguali, all'interno di
% un blocco associato ad una transizione che è in
% relazione di scelta libera con altre transizioni, che
% altrimenti genererebbero vincoli uguali nel CS modificato
               sigma3(1+r,:)=[];
               sigma_plus_t3(1+r,:)=[];
               sigm3=size(sigma3,1);
            else r=r+1;
            end
      end
      sigma3(1,:)=[];
      sigma_plus_t3(1,:)=[];
      sigm3=size(sigma3,1);
   end
% il ciclo while memorizza nelle matrici sigma_D3 e
% sigma_plus_t_D3 le differenti coppie
% sigma3(i,:)-sigma_plus_t3(i,:); il numero di coppie
% è memorizzato nella variabile count
   nelbl1(i1(2)-pred)=nelbl1(i1(2)-pred)+count;
% nelbl1 è un vettore riga in cui ciascuna componente
% rappresenta il numero di elementi contenuti in un blocco
% della partizione ammissibile nel caso in cui ci siano due
% o più transizioni in relazione di scelta libera;
% vengono considerati solo i blocchi relativi
% alle transizioni in relazione di scelta libera
  end
 end
% il ciclo for riduce il numero di posti q ed eventualmente
% l'insieme D se son presenti informazioni di pre-riduzione
 if numCol_pre_red>1
    q=q+m;
 end
 sigma_D=[sigma_D1;sigma_D3;sigma_D2];
 sigma_plus_t_D=...
  [sigma_plus_t_D1;sigma_plus_t_D3;sigma_plus_t_D2];
 nelbl=[nelbl nelbl1];
% nelbl1 è un vettore riga in cui ciascuna componente
% rappresenta il numero di elementi contenuti in un blocco
% della partizione ammissibile nel caso in cui ci siano
% transizioni che hanno un solo posto in ingresso;
% vengono considerati solo i blocchi relativi
% alle transizioni con un solo posto in ingresso
```

```
else sigmaD=sigma(card_E+1:card_ED,:);
     sigma_plus_tD=sigma_plus_t(card_E+1:card_ED,:);
     sig_D=card_D;
     sigma_D=[];
     sigma_plus_t_D=[];
     while sig_D>=1
        sigma_D=[sigma_D;sigmaD(1,:)];
        sigma_plus_t_D=[sigma_plus_t_D;sigma_plus_tD(1,:)];
        i=1;
        while i<=sig_D-1
              if sigmaD(1+i,:)==sigmaD(1,:) &...
                sigma_plus_tD(1+i,:)==sigma_plus_tD(1,:)
                 q=q-1;
                 sigmaD(1+i,:)=[];
                 sigma_plus_tD(1+i,:)=[];
                 sig_D=size(sigmaD,1);
              else i=i+1;
              end
        end
        sigmaD(1,:)=[];
        sigma_plus_tD(1,:)=[];
        sig_D=size(sigmaD,1);
     end
% il ciclo while elimina le coppie
% sigma(i,:)-sigma_plus_t(i,:) uguali relative all'insieme
% D in modo da eliminare i posti ridondanti, nel caso
% in cui non si abbiano informazioni sulla rete
% per poter applicare la pre-riduzione
end
sigm_D=size(sigma_D,1);
% sigm_D è il numero dei vincoli corrispondente
% all'insieme delle condizioni di disabilitazione

sig_E=card_E;
sigma_E=[];
sigma_plus_t_E=[];
while sig_E>=1
      sigma_E=[sigma_E;sigma(1,:)];
      sigma_plus_t_E=[sigma_plus_t_E;sigma_plus_t(1,:)];
      i=1;
      while i<=sig_E-1
            if sigma(1+i,:)==sigma(1,:) &...
                sigma_plus_t(1+i,:)==sigma_plus_t(1,:)
```

```
                sigma(1+i,:)=[];
                sigma_plus_t(1+i,:)=[];
                sigm=size(sigma,1);
                sig_E=sigm-card_D;
            else i=i+1;
            end
        end
        sigma(1,:)=[];
        sigma_plus_t(1,:)=[];
        sigm=size(sigma,1);
        sig_E=sigm-card_D;
end
% il ciclo while elimina le coppie
% sigma(i,:)-sigma_plus_t(i,:) uguali relative all'insieme
% E che altrimenti genererebbero vincoli uguali nel CS
sigm_E=size(sigma_E,1);
nv=sigm_E*q;
% nv è il numero dei vincoli corrispondente
% all'insieme delle condizioni di abilitazione
A1=speye(q);
% A1 è una matrice identità sparsa q x q; se pre_red è
% vuota, è una sottomatrice di A contenente i coefficienti
% del Constraint Set relativi a D e associati
% all'incognita M0 del problema di identificazione
A2=[];
A3=[];
A5=[];
for i=1:sigm_E
    a3=[];
    a5=[];
    for r=1:n
        a3=[a3 sigma_E(i,r)*A1];
        a5=[a5 sigma_plus_t_E(i,r)*A1];
    end
    A2=[A2;A1];
    A3=[A3;a3];
    A5=[A5;a5];
end
% i cicli for restituiscono le matrici sparse A2, A3 e A5,
% che sono sottomatrici di A contenenti i coefficienti del
% Constraint Set relativi ad E e associati rispettivamente
% alle incognite M0, Post e Pre del problema di
% identificazione; la dimensione di A2 è nv x q,
```

```
% la dimensione di A3 e A5 è nv x q*n
if isempty(pre_red)==0
   A1_1=[];
   for i=1:length(nelbl)
       A1_1=[A1_1;repmat(A1(i,:),nelbl(i),1)];
   end
   for i=1:nel
       A1_1=[A1_1;A1(i+length(nelbl),:)];
   end
   A1=A1_1;
end
% se pre_red non è vuota, i due cicli for restituiscono la
% matrice A1 che è una sottomatrice  di A contenente i
% coefficienti del Constraint Set relativi a D e associati
% all'incognita M0 del problema di identificazione
A4=[];
A6=[];
for r=1:n
    a4=[];
    a6=[];
    for i=1:sigm_D
        a4=[a4;sigma_D(i,r)*A1(i,:)];
        a6=[a6;sigma_plus_t_D(i,r)*A1(i,:)];
    end
    A4=[A4 a4];
    A6=[A6 a6];
end
% i cicli for restituiscono le matrici sparse A4 e A6, che
% sono sottomatrici di A contenenti i coefficienti del
% Constraint Set relativi ad D e associati rispettivamente
% alle incognite Post e Pre del problema di
% identificazione; la dimensione di A3 e A5 è sigm_D x q*n
A=[A2 A3 -A5;-A1 -A4 A6];
% A è memorizzata in forma compatta, ossia memorizzando
% solo gli elementi non nulli insieme ai loro indici;
% ciò consente di ridurre il tempo di calcolo eliminando
% operazioni su elementi nulli

b=[zeros(nv,1);ones(sigm_D,1)];

ct='L';
ctype=ct(:,ones(1,nv+sigm_D)).';
ctype=ctype(:);
```

```
xm=q+2*q*n;
% xm è il numero di variabili del problema
lb=zeros(xm,1);

vart='C';
vartype=vart(:,ones(1,xm)).';
vartype=vartype(:);
```

A.6 Estrazione della rete P/T

```
function [M0,Post,Pre,fm,ex]=CS_infoPN2PN(q,A,b,...
 ctype,lb,vartype,P_inv,P_inc,P_dec,T_inv,T_inc,T_dec,...
 state_mach,mark_graph,ord_PN,Gmec,msglev,dual,lpsolver)
% Questa funzione identifica una rete Posto/Transizione
% attraverso la risoluzione di un problema di
% programmazione lineare, a partire dal Constraint Set(CS),
% il sistema di equazioni dei vincoli, e da altre
% informazioni (opzionali) di natura strutturale e
% comportamentale sulla rete
% che aggiungono dei vincoli al CS
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
%                     ## SINTASSI ##
%   [M0,Post,Pre,fm,ex]=CS_infoPN2PN(q,A,b,ctype,lb,
%    vartype,P_inv,P_inc,P_dec,T_inv,T_inc,T_dec,
%    state_mach,mark_graph,ord_PN,
%    Gmec,msglev,dual,lpsolver)
%                 ARGOMENTI DI INGRESSO:
%  q: variabile che memorizza il numero di blocchi di una
%     partizione ammissibile dell'insieme D e rappresenta
%     il numero di posti della rete risolvente
%     il problema di identificazione
%  A: matrice sparsa contenente i coefficienti dei vincoli;
%     la sua dimensione è
%     (nv+sigm_D) x xm o (nv+sigm_D) x (q*(2n+1))
%  b: vettore colonna di lunghezza nv+sigm_D contenente
%     il termine noto per ogni vincolo
%     nella matrice dei vincoli
%  ctype: vettore colonna di lunghezza nv+sigm_D
%         contenente il senso di ogni vincolo
%  lb: vettore di lunghezza xm contenente il limite
```

```
%      inferiore di ciascuna delle variabili
%      incognite del problema di identificazione
%  vartype: vettore colonna di lunghezza xm contenente la
%           definizione del tipo di variabili (continue
%           per un problema di programmazione lineare)
%  P_inv: matrice che, se non vuota, ha q colonne
%         e le righe sono i P-invarianti
%         i quali aggiungono dei vincoli al CS
%  P_inc: matrice che, se non vuota, ha q colonne
%         e le righe sono i P-vettori crescenti
%         i quali aggiungono dei vincoli al CS
%  P_dec: matrice che, se non vuota, ha q colonne
%         e le righe sono i P-vettori decrescenti
%         i quali aggiungono dei vincoli al CS
%  T_inv: matrice che, se non vuota, ha n colonne
%         e le righe sono i T-invarianti
%         i quali aggiungono dei vincoli al CS
%  T_inc: matrice che, se non vuota, ha n colonne
%         e le righe sono i T-vettori crescenti
%         i quali aggiungono dei vincoli al CS
%  T_dec: matrice che, se non vuota, ha n colonne
%         e le righe sono i T-vettori decrescenti
%         i quali aggiungono dei vincoli al CS
%  state_mach: vettore riga di lunghezza n che, se non
%              vuoto, aggiunge dei vincoli al CS; il
%              vettore ha tutti i suoi elementi pari ad 1
%              se la rete complessiva è una macchina di
%              stato e alcuni elementi pari ad 1 ed altri
%              pari a 0 se solamente una o più sottoreti
%              sono macchine di stato; in quest'ultimo caso
%              un elemento di indice i pari ad 1
%              corrisponde ad una transizione facente parte
%              di qualche macchina di stato mentre un
%              elemento nullo corrisponde ad una
%              transizione non contenuta
%              in alcuna macchina di stato
%  mark_graph: vettore riga di lunghezza q che, se non
%              vuoto, aggiunge dei vincoli al CS; il
%              vettore ha tutti i suoi elementi pari ad 1
%              se la rete complessiva è un grafo marcato e
%              alcuni elementi pari ad 1 ed altri pari a 0
%              se solamente una o più sottoreti sono grafi
%              marcati; in quest'ultimo caso un elemento di
```

```
%              indice i pari ad 1 corrisponde ad una
%              transizione facente parte di qualche grafo
%              marcato mentre un elemento nullo corrisponde
%              ad una transizione non contenuta
%              in alcun grafo marcato
%  ord_PN: variabile che, se non vuota, vale 1 se la rete è
%          ordinaria (aggiunge dei vincoli al CS se la rete
%          non è nè una macchina di stato nè un grafo
%          marcato) e 0 se non lo è
%  Gmec: vettore riga che, se non vuoto, introduce nel CS
%        un vincolo sulla marcatura iniziale del tipo
%        w*M0<=c chiamato GMEC; la sua dimensione è q+1
%        dove i primi q elementi rappresentano il vettore
%        w mentre l'ultimo elemento è la costante c
%  msglev: variabile intera che permette di scegliere il
%          tipo del messaggio di uscita del risolutore: 1
%          per soli messaggi di errore, 2 per un'uscita
%          normale e 3 per un'uscita completa
%          (include messaggi informativi)
%  dual: variabile intera pari a 0 se non si intende
%        utilizzare il metodo del simplesso duale o pari a
%        1 se lo si vuole utilizzare nel caso in cui la
%        soluzione di base iniziale sia ammissibile duale
%  lpsolver: variabile intera pari a 1 se si intende
%            risolvere il problema col metodo del simplesso
%            revisionato o pari a 2 se si vuole
%            risolverlo col metodo del punto interno
%                 ARGOMENTI DI USCITA:
%  M0: vettore colonna marcatura iniziale di lunghezza q
%      in cui ciascuna componente indica
%      il numero di gettoni contenuti in un posto
%  Post: matrice detta funzione di post-incidenza in cui
%        ciascun elemento Post(p,t) indica quanti archi
%        vanno dalla transizione t al posto p;
%        la dimensione è q x n
%  Pre: matrice detta funzione di pre-incidenza in cui
%       ciascun elemento Pre(p,t) indica quanti archi
%       vanno dal posto p alla transizione t;
%       la dimensione è q x n
%  fm: valore ottimo della funzione obiettivo
%  ex: struttura dati contenente variabili duali,
%      costi ridotti, tempo(in secondi) e
%      memoria(in bytes) usati per risolvere il problema
```

```
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
% Copyright 2008 by Pierandrea Secchi. Rev.: 15 luglio 2008

nargin=19;
% numero di argomenti di ingresso della funzione
nargout=5;
% numero di argomenti di uscita della funzione
sense=1;
% problema di minimizzazione
xm=size(A,2);
% size restituisce il numero di colonne di A
% come una variabile separata
n=(xm/q-1)/2;
% n è il numero di transizioni
c=ones(xm,1);
% vettore colonna di lunghezza xm contenente
% i coefficienti della funzione obiettivo
add_A=[];
add_b=[];
add_ctype=[];
if isempty(P_inv)==0
   q1=size(P_inv,1);
   for i=1:q1
       add=zeros(n,q*n);
       for j=1:n
           add(j,q*(j-1)+1:q*(j-1)+q)=P_inv(i,:);
       end
       add=[zeros(n,q) add -add];
       add_A=[add_A;sparse(add)];
       add_b=[add_b;zeros(n,1)];
       adct='S';
       adctyp=adct(:,ones(1,n)).';
       adctyp=adctyp(:);
       add_ctype=[add_ctype;adctyp];
   end
end
% il ciclo for aggiunge dei vincoli al CS
% associati ai P-invarianti
if isempty(P_inc)==0
   q1=size(P_inc,1);
   for i=1:q1
       add=zeros(n,q*n);
       for j=1:n
```

```
            add(j,q*(j-1)+1:q*(j-1)+q)=P_inc(i,:);
        end
        add=[(zeros(n,q)) add -add];
        add1=[zeros(1,q)...
            repmat(P_inc(i,:),1,n) -repmat(P_inc(i,:),1,n)];
% repmat crea una matrice consistente di
% 1 x n copie di P_inc(i,:) ad ogni ciclo
        add_A=[add_A;sparse(add);sparse(add1)];
        add_b=[add_b;zeros(n,1);1];
        adct='L';
        adctyp=adct(:,ones(1,n+1)).';
        adctyp=adctyp(:);
        add_ctype=[add_ctype;adctyp];
    end
end
% il ciclo for aggiunge dei vincoli al CS
% associati ai P-vettori crescenti
if isempty(P_dec)==0
    q1=size(P_dec,1);
    for i=1:q1
        add=zeros(n,q*n);
        for j=1:n
            add(j,q*(j-1)+1:q*(j-1)+q)=P_dec(i,:);
        end
        add=[(zeros(n,q)) -add add];
        add1=[zeros(1,q)...
            -repmat(P_dec(i,:),1,n) repmat(P_dec(i,:),1,n)];
        add_A=[add_A;sparse(add);sparse(add1)];
        add_b=[add_b;zeros(n,1);1];
        adct='L';
        adctyp=adct(:,ones(1,n+1)).';
        adctyp=adctyp(:);
        add_ctype=[add_ctype;adctyp];
    end
end
% il ciclo for aggiunge dei vincoli al CS
% associati ai P-vettori decrescenti
if isempty(T_inv)==0
    n1=size(T_inv,1);
    add=eye(q);
    for i=1:n1
        add1=[];
        for j=1:n
```

```
            add1=[add1 T_inv(i,j)*add];
        end
        add1=[(zeros(q)) add1 -add1];
        add_A=[add_A;sparse(add1)];
        add_b=[add_b;zeros(q,1)];
        adct='S';
        adctyp=adct(:,ones(1,q)).';
        adctyp=adctyp(:);
        add_ctype=[add_ctype;adctyp];
    end
end
% il ciclo for aggiunge dei vincoli al CS
% associati ai T-invarianti
if isempty(T_inc)==0
    n1=size(T_inc,1);
    add=eye(q);
    for i=1:n1
        add1=[];
        for j=1:n
            add1=[add1 T_inc(i,j)*add];
        end
        add1=[zeros(q) add1 -add1];
        Tin_i=T_inc(i,:);
        add2=Tin_i(ones(1,q),:);
        add2=add2(:).';
        add2=[zeros(1,q) add2 -add2];
        add_A=[add_A;sparse(add1);sparse(add2)];
        add_b=[add_b;zeros(q,1);1];
        adct='L';
        adctyp=adct(:,ones(1,q+1)).';
        adctyp=adctyp(:);
        add_ctype=[add_ctype;adctyp];
    end
end
% il ciclo for aggiunge dei vincoli al CS
% associati ai T-vettori crescenti
if isempty(T_dec)==0
    n1=size(T_dec,1);
    add=eye(q);
    for i=1:n1
        add1=[];
        for j=1:n
            add1=[add1 T_dec(i,j)*add];
```

```
       end
       add1=[zeros(q) -add1 add1];
       Tde_i=T_dec(i,:);
       add2=Tde_i(ones(1,q),:);
       add2=add2(:).';
       add2=[zeros(1,q) -add2 add2];
       add_A=[add_A;sparse(add1);sparse(add2)];
       add_b=[add_b;zeros(q,1);1];
       adct='L';
       adctyp=adct(:,ones(1,q+1)).';
       adctyp=adctyp(:);
       add_ctype=[add_ctype;adctyp];
   end
end
% il ciclo for aggiunge dei vincoli al CS
% associati ai T-vettori decrescenti
if isempty(state_mach)==0
 if isequal(state_mach,ones(1,n))==1
  for i=1:2*n
      add=[zeros(1,i*q) ones(1,q) zeros(1,(2*n-i)*q)];
      add_A=[add_A;sparse(add)];
  end
  add_b=[add_b;ones(2*n,1)];
  adct='S';
  adctyp=adct(:,ones(1,2*n)).';
  adctyp=adctyp(:);
  add_ctype=[add_ctype;adctyp];
 else n1=find(state_mach==1);
  len=length(n1);
  for i=1:len
   add=[zeros(1,q*n1(i)) ones(1,q) zeros(1,q*(2*n-n1(i)))];
   add1=...
     [zeros(1,q*(n+n1(i))) ones(1,q) zeros(1,q*(n-n1(i)))];
   add_A=[add_A;sparse(add);sparse(add1)];
  end
  add_b=[add_b;ones(2*len,1)];
  adct='S';
  adctyp=adct(:,ones(1,2*len)).';
  adctyp=adctyp(:);
  add_ctype=[add_ctype;adctyp];
 end
end
% i cicli for aggiungono dei vincoli al CS se la rete
```

```
% o una o più sottoreti sono macchine di stato
if isempty(mark_graph)==0
   if isequal(mark_graph,ones(1,q))==1
      add=repmat(eye(q),1,n);
      add=[zeros(2*q,q) blkdiag(add,add)];
% blkdiag esegue una concatenazione diagonale
% a blocchi di add con sè stesso
      add_A=[add_A;sparse(add)];
      add_b=[add_b;ones(2*q,1)];
      adct='S';
      adctyp=adct(:,ones(1,2*q)).';
      adctyp=adctyp(:);
      add_ctype=[add_ctype;adctyp];
   else q1=find(mark_graph==1);
        len=length(q1);
        add=zeros(2*len,xm);
        for i=1:len
            for j=1:n
                add(i,q*j+q1(i))=1;
                add(i+len,q*(j+n)+q1(i))=1;
            end
        end
        add_A=[add_A;sparse(add)];
        add_b=[add_b;ones(2*len,1)];
        adct='S';
        adctyp=adct(:,ones(1,2*len)).';
        adctyp=adctyp(:);
        add_ctype=[add_ctype;adctyp];
   end
end
% i cicli for aggiungono dei vincoli al CS se
% la rete o una o più sottoreti sono grafi marcati
if isempty(Gmec)==0
   add=[Gmec(1:q) zeros(1,2*q*n)];
   add_A=[add_A;sparse(add)];
   add_b=[add_b;Gmec(q+1)];
   add_ctype=[add_ctype;'U'];
end
% viene aggiunto un vincolo al CS se è definita una GMEC
A=[A;add_A];
b=[b;add_b];
ctype=[ctype;add_ctype];
ub=[];
```

```
% vettore di lunghezza xm contenente il limite superiore
% di ciascuna delle variabili
if ord_PN==1 & isequal(state_mach,ones(1,n))==0 &...
    isequal(mark_graph,ones(1,q))==0
   ub=zeros(xm,1);
   ub(1:q)=exp(1000);
% exp(1000) produce l'infinito positivo da un overflow
   ub(q+1:end)=1;
end
% vengono aggiunti dei vincoli al CS se la rete è ordinaria
% e non è nè una macchina di stato nè un grafo marcato
if msglev==1
   param.msglev=1;
elseif msglev==2
       param.msglev=2;
else param.msglev=3;
end
% il parametro param.msglev ha lo stesso significato
% della variabile msglev
if dual==0
   param.dual=0;
else param.dual=1;
end
% il parametro param.dual ha lo stesso significato
% della variabile dual
save=1;
% quando l'argomento save assume un qualsiasi valore
% diverso da zero viene salvata una copia del
% problema nella directory corrente
[xmin,fmin,status,extra]=glpkmex...
 (sense,c,A,b,ctype,lb,ub,vartype,param,lpsolver,save);
% sintassi chiamante di glpkmex per risolvere
% il problema di programmazione lineare
M0=xmin(1:q);

Post=zeros(q,n);
Pre=zeros(q,n);
for i=1:n
    Post(:,i)=xmin(q*i+1:q*(i+1));

    Pre(:,i)=xmin(q*(n+i)+1:q*(n+i+1));
end
% M0,Post e Pre rappresentano la soluzione intera o
```

```
% razionale del problema di identificazione

fm=fmin;

if any(M0(:)~=round(M0(:))) |...
  any(Post(:)~=round(Post(:))) | any(Pre(:)~=round(Pre(:)))
% l'if controlla se la soluzione trovata
% è intera o razionale
   Soluz=[M0 Post Pre];
   [num,den]=rat(Soluz);
% rat restituisce due matrici intere in modo tale che n/d
% sia vicina a Soluz nel senso che
% abs(n./d-Soluz)<=tol*abs(Soluz);
% tol=10^-6*norm(Soluz(:),1) è il valore di default
   if isequal(den,ones(q,2*n+1))==1
% l'if controlla l'uguaglianza tra
% le due matrici den e ones(q,2*n+1)
      M0=num(:,1);

      Post=num(:,2:n+1);

      Pre=num(:,n+2:2*n+1);
   else alpha1=max(den);
        alpha=max(alpha1);
        nd=num./den;
% num./den divide num per den elemento per elemento
        Soluz=alpha*nd;
        while any(Soluz(:)~=round(Soluz(:)))
              alpha=2*alpha;
              Soluz=alpha*Soluz;
        end
        M0=Soluz(:,1);

        Post=Soluz(:,2:n+1);

        Pre=Soluz(:,n+2:2*n+1);
   end
   fm=sum(M0)+sum(sum(Post+Pre));
end
% Se la soluzione del CS è razionale possiamo sempre
% trovare una soluzione intera semplicemente moltiplicando
% M0, Post e Pre per un opportuno intero non negativo alpha
% che è esattamente il minimo comune multiplo
```

```
% degli elementi della matrice Soluz;
% ovviamente se la soluzione di CS è intera alpha=1

ex=extra;
```

A.7 Estrazione della rete P/T ridotta

```
function [red_M0,red_Post,red_Pre]=...
 PN_D2post_red_PN(M0,Post,Pre,D,msglev)
% Questa funzione riduce il numero di posti della rete di
% Petri Posto/Transizione,ricavata risolvendo un
% problema di programmazione lineare,
% mediante l'approccio della post-riduzione
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
%                     ## SINTASSI ##
%    [red_M0,red_Post,red_Pre]=
%     PN_D2post_red_PN(M0,Post,Pre,D,msglev)
%                ARGOMENTI DI INGRESSO:
%  M0: vettore colonna marcatura iniziale di lunghezza q
%      in cui ciascuna componente indica il numero di
%      gettoni contenuti in un posto
%  Post: matrice detta funzione di post-incidenza in cui
%        ciascun elemento Post(p,t) indica quanti archi
%        vanno dalla transizione t al posto p;
%        la dimensione è q x n
%  Pre: matrice detta funzione di pre-incidenza in cui
%       ciascun elemento Pre(p,t) indica quanti archi
%       vanno dal posto p alla transizione t;
%       la dimensione è q x n
%  D: matrice delle condizioni di disabilitazione dove ogni
%     riga contiene gli indici i delle transizioni t_i
%     nella condizione (sigma,t), disposte nell'ordine in
%     cui compaiono in tale condizione;
%     la sua dimensione è card_D x numCol_D
%  msglev: variabile intera che permette di scegliere il
%          tipo del messaggio di uscita del risolutore: 1
%          per soli messaggi di errore, 2 per un'uscita
%          normale e 3 per un'uscita completa
%          (include messaggi informativi)
%                 ARGOMENTI DI USCITA:
%  red_M0: vettore colonna M0 ridotto
```

```
%          la cui lunghezza è pari ad p
%  red_Post: matrice Post ridotta di dimensioni p x n
%  red_Pre: matrice Pre ridotta di dimensioni p x n
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
% Copyright 2008 by Pierandrea Secchi. Rev.: 15 luglio 2008

nargin=5;
% number of function input arguments
nargout=3;
% number of function output arguments
[q,n]=size(Post);
% size restituisce il numero di righe e colonne di
% Post(and Pre) come separate variabili di uscita
card_D=size(D,1);
global j;
j=j(end-card_D+1:end);
C=Post-Pre;
% C è la matrice di incidenza
y=zeros(card_D,q);
global sigma;
for i=1:card_D
    for i1=1:q
       if M0(i1)+sum(C(i1,:).*sigma(i,:))<Pre(i1,D(i,j(i)))
          y(i,i1)=1;
       end
    end
end
sense=1;
% problema di minimizzazione
c=ones(q,1);
% vettore colonna di lunghezza q contenente
% i coefficienti della funzione obiettivo
A=sparse(y);
% matrice sparsa contenente i coefficienti dei vincoli;
% la sua dimensione è card_D x q
b=ones(card_D,1);
% vettore colonna di lunghezza card_D contenente il
% termine noto per ogni vincolo nella matrice dei vincoli
ct='L';
ctype=ct(:,ones(1,card_D)).';
ctype=ctype(:);
% vettore colonna di lunghezza card_D contenente
% il senso di ogni vincolo
```

```
lb=zeros(q,1);
% vettore di lunghezza q contenente il limite inferiore
% di ciascuna delle variabili
% incognite del problema di programmazione intera
ub=ones(q,1);
% vettore di lunghezza q contenente il limite superiore
% di ciascuna delle variabili incognite
vart='I';
vartype=vart(:,ones(1,q)).';
vartype=vartype(:);
% vettore colonna di lunghezza q contenente
% la definizione del tipo di variabili
% (intere per un problema di programmazione intera)
if msglev==1
   param.msglev=1;
elseif msglev==2
       param.msglev=2;
else param.msglev=3;
end
save=1;
% quando l'argomento save assume un qualsiasi
% valore diverso da zero viene salvata
% una copia del problema nella directory corrente
[xmin,fmin,status,extra]=glpkmex...
 (sense,c,A,b,ctype,lb,ub,vartype,param,save);
% sintassi chiamante di glpkmex per risolvere
% il problema di programmazione
% in modo da determinare il minimo hitting set
red_M0=[];
red_Post=[];
red_Pre=[];
for i=1:q
    if xmin(i)==1
       red_M0=[red_M0;M0(i)];

       red_Post=[red_Post;Post(i,:)];

       red_Pre=[red_Pre;Pre(i,:)];
    end
end
% red_M0,red_Post e red_Pre rappresentano la
% soluzione finale del problema di identificazione
```

A.8 Esecuzione dell'intera procedura

```
function [red_M0,red_Post,red_Pre,maxL2]=identif_proced...
 (maxL,n,k,pre_red,T_inv,T_inc,T_dec,...
 state_mach,ord_PN,msglev,dual,lpsolver)
% Questa funzione esegue la procedura completa di
% identificazione di una rete posto/transizione richiamando
% tutte le altre funzioni create che rappresentano
% i singoli moduli della procedura
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * *
%                    ## SINTASSI ##
%   [red_M0,red_Post,red_Pre,maxL2]=identif_proced
%    (maxL,n,k,pre_red,T_inv,T_inc,T_dec,
%     state_mach,ord_PN,msglev,dual,lpsolver)
%                ARGOMENTI DI INGRESSO:
%  maxL: matrice delle sequenze di scatto aventi la stessa
%        lunghezza, la più grande possibile minore o uguale
%        a k, dove ogni riga contiene gli indici i delle
%        transizioni t_i che compongono la generica
%        sequenza di scatto sigma, disposte nell'ordine
%        in cui compaiono in tale sequenza;
%        la dimensione è numRows x numCol
%  n: numero di transizioni
%  k: intero naturale maggiore o uguale alla lunghezza
%     delle stringhe del linguaggio massimale
%  pre_red: matrice vuota se non ci sono informazioni
%           aggiuntive che consentono di scrivere il CS in
%           una forma modificata usando un numero ridotto
%           di posti (pre-riduzione dei posti) o matrice
%           non vuota se son presenti tali informazioni; in
%           quest'ultimo caso le righe con un solo elemento
%           non nullo, che vale i indicano che la
%           transizione i ha solo un posto in ingresso
%           mentre le altre righe, se esistono, indicano
%           che i valori degli elementi non nulli
%           corrispondono a transizioni
%           in una relazione di scelta libera
%  T_inv: matrice che, se non vuota, ha n colonne
%         e le righe sono i T-invarianti
%         i quali aggiungono dei vincoli al CS
%  T_inc: matrice che, se non vuota, ha n colonne
%         e le righe sono i T-vettori crescenti
```

```
%         i quali aggiungono dei vincoli al CS
%  T_dec: matrice che, se non vuota, ha n colonne
%         e le righe sono i T-vettori decrescenti
%         i quali aggiungono dei vincoli al CS
%  state_mach: vettore riga di lunghezza n che, se non
%              vuoto, aggiunge dei vincoli al CS; il
%              vettore ha tutti i suoi elementi pari ad 1
%              se la rete complessiva è una macchina di
%              stato e alcuni elementi pari ad 1 ed altri
%              pari a 0 se solamente una o più sottoreti
%              sono macchine di stato; in quest'ultimo caso
%              un elemento di indice i pari ad 1
%              corrisponde ad una transizione facente parte
%              di qualche macchina di stato mentre un
%              elemento nullo corrisponde ad una
%              transizione non contenuta in alcuna
%              macchina di stato
%  ord_PN: variabile che,se non vuota,vale 1 se la rete è
%          ordinaria (aggiunge dei vincoli al CS se la rete
%          non è nè una macchina di stato nè un grafo
%          marcato) e 0 se non lo è
%  msglev: variabile intera che permette di scegliere il
%          tipo del messaggio di uscita del risolutore: 1
%          per soli messaggi di errore, 2 per un'uscita
%          normale e 3 per un'uscita completa
%          (include messaggi informativi)
%  dual: variabile intera pari a 0 se non si intende
%        utilizzare il metodo del simplesso duale o pari a
%        1 se lo si vuole utilizzare nel caso in cui la
%        soluzione di base iniziale sia ammissibile duale
%  lpsolver: variabile intera pari a 1 se si intende
%            risolvere il problema col metodo del
%            simplesso revisionato o pari a 2 se si vuole
%            risolverlo col metodo del punto interno
%                 ARGOMENTI DI USCITA:
%  red_M0: vettore colonna M0 ridotto
%          la cui lunghezza è pari ad p
%  red_Post: matrice Post ridotta di dimensioni p x n
%  red_Pre: matrice Pre ridotta di dimensioni p x n
%  maxL2: matrice delle sequenze di scatto aventi la
%         stessa lunghezza, la più grande possibile minore
%         o uguale a k, dove ogni riga contiene gli indici
%         i delle transizioni t_i che compongono la
```

```
%         generica sequenza di scatto sigma, disposte
%         nell'ordine in cui compaiono in tale sequenza;
%         dev'essere uguale a maxL e a maxL1
%         se la rete è identificata correttamente
%  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
% Copyright 2008 by Pierandrea Secchi. Rev.: 15 luglio 2008

nargin=12;
% numero di argomenti di ingresso della funzione
nargout=4;
% numero di argomenti di uscita della funzione
[E,D]=maxL_n_k2E_D(maxL,n,k);
size_D=size(D)
% le dimensioni di D vengono mostrate sullo schermo
[q,A,b,ctype,lb,vartype]=E_D_pre_red2CS(E,D,pre_red);
q
size_A=size(A)
% le dimensioni di A vengono mostrate sullo schermo
inputdata_CS_infoPN2PN;
% inputdata_CS_infoPN2PN viene richiamato per settare gli
% ingressi di CS_infoPN2PN dipendenti dal numero di posti q
% che non possono essere dati in ingresso a identif_proced
[M0,Post,Pre,fm,ex]=CS_infoPN2PN(q,A,b,ctype,lb,vartype,...
 P_inv,P_inc,P_dec,T_inv,T_inc,T_dec,...
 state_mach,mark_graph,ord_PN,Gmec,msglev,dual,lpsolver);
M0_PPL=M0
% M0_PPL è il vettore colonna marcatura iniziale di
% lunghezza q in cui ciascuna componente
% indica il numero di gettoni contenuti in un posto
Post_PPL=Post
% Post_PPL è la matrice detta funzione di
% post-incidenza in cui ciascun elemento Post(q,t) indica
% quanti archi vanno dalla transizione t al posto p;
% la dimensione è q x n
Pre_PPL=Pre
% Pre_PPL è la matrice detta funzione di pre-incidenza
% in cui ciascun elemento Pre(q,t) indica quanti archi
% vanno dal posto p alla transizione t; la dimensione è
% q x n. M0_PPL, Post_PPL e Pre_PPL
% rappresentano la soluzione del problema di programmazione
% lineare; i tre assegnamenti immediatamente sopra si sono
% resi necessari per mostrare sullo schermo tale soluzione
% visto che M0, Post e Pre
```

```
% saranno sovrascritti successivamente
fm
ex
[maxL,n]=PN_k2maxL_n(M0,Post,Pre,k);
size_maxL1=size(maxL)
% le dimensioni di maxL1 vengono mostrate sullo schermo
maxL1=maxL
% maxL1 è la matrice delle sequenze di scatto aventi la
% stessa lunghezza, la più grande possibile minore o uguale
% a k, dove ogni riga contiene gli indici i delle
% transizioni t_i che compongono la generica sequenza di
% scatto sigma, disposte nell'ordine in cui compaiono in
% tale sequenza. maxL1 è il linguaggio massimale generato
% dalla rete soluzione del problema di programmazione
% lineare e dev'essere uguale a maxL se la rete è
% identificata correttamente; l'assegnamento
% di cui sopra, anche se non necessario, è stato fatto
% solo per chiamare con nomi diversi il linguaggio
% osservato e quello generato dalla rete soluzione
% del problema di programmazione lineare
% sarà sovrascritto successivamente
[red_M0,red_Post,red_Pre]=...
 PN_D2post_red_PN(M0,Post,Pre,D,msglev);
% red_M0, red_Post e red_Pre rappresentano la soluzione
% finale della procedura completa di identificazione

M0=red_M0;
Post=red_Post;
Pre=red_Pre;
% i tre assegnamenti immediatamente sopra si sono resi
% necessari per calcolare il linguaggio massimale generato
% dalla rete soluzione dell'intera procedura
[maxL,n]=PN_k2maxL_n(M0,Post,Pre,k);
maxL2=maxL;
% maxL2 è il linguaggio massimale generato dalla rete
% soluzione dell'intera procedura e dev'essere uguale a
% maxL per la correttezza della procedura; l'assegnamento
% di cui sopra, anche se non necessario, è stato fatto
% solo per chiamare con nomi diversi il linguaggio
% osservato e quello generato dalla rete ridotta finale
```

Appendice B

GLPK e GLPKMEX

B.1 GLPK

GLPK sta per GNU *Linear Programming Kit* ed è un software progettato per risolvere problemi di programmazione lineare (LP o *Linear Programming*) su larga scala, programmazione lineare misto intera (MIP o *Mixed Integer Linear Programming*) e altri problemi correlati. GLPK è un insieme di *routines* scritte in ANSI C e organizzate in forma di libreria. Questa libreria è *open source* e include i seguenti componenti principali:

- Implementazione del metodo del simplesso revisionato (basato sulla tecnica a matrice sparsa, sul cosiddetto *steepest edge pricing* e sulla tecnica del pivot in due passi).
- Implementazione del metodo del punto interno primario-duale.
- Implementazione della procedura *branch-and-bound* (basata sul metodo del simplesso duale).
- Interfaccia di applicazione dei programmi (API).
- Linguaggio di modellazione GNU *MathProg*.
- GLPSOL, un risolutore indipendente LP/MIP.

GLPK è attualmente sviluppato e curato da Andrew Makhorin, un ricercatore russo del Dipartimento di Informatica Applicata dell'Istituto di Aviazione di Mosca e il suo utilizzo è autorizzato sotto la GNU General Public License (GPL). È un software gratuito. GLPK è un lavoro in stato di avanzamento ed è attualmente sotto continuo sviluppo. La corrente

versione 4.9 di GLPK è un risolutore basato sul simplesso capace di trattare bene problemi sino a 100.000 vincoli. Il risolutore del punto-interno non è molto robusto in quanto è incapace di trattare colonne dense e a volte termina a causa dell'instabilità numerica o della convergenza lenta. Il risolutore della Programmazione Misto Intera (MIP) attualmente è basato sul *branch-and-bound*, perciò è incapace di risolvere problemi difficili o molto grandi. Infatti il limite sul numero di variabili intere stimato è di circa 200. Tuttavia, qualche volta è capace di risolvere problemi più grandi sino a 1000 variabili intere, sebbene la dimensione dipenda dalle proprietà del particolare problema. Gli sviluppi pianificati per GLPK includono il miglioramento degli esistenti componenti GLPK chiave, come sviluppare una più robusta e più efficiente implementazione dei risolutori basati sul simplesso e sul punto interno. Futuri miglioramenti di GLPK riguardano l'implementazione di un risolutore *branch-and-cut*, di un pre-processore MIP, di un'analisi post-ottimale e della sensibilità e possibilmente di risolutori del simplesso di rete e di programmazione quadratica.

B.2 GLPKMEX

GLPKMEX è un'interfaccia MEX per la libreria GLPK ideata da Nicolò Giorgetti dell'Università di Siena. GLPKMEX è un software gratuito distribuito nella speranza che sia utile, ma senza alcuna garanzia. In pratica è un risolutore che è parte integrante di GLPK e il cui relativo codice può essere compilato e collegato con le librerie di Matlab e può essere usato in ambiente Matlab. Questa *routine* chiama la libreria GLPK per risolvere un problema LP/MIP. Un tipico problema LP ha la seguente struttura:

$$\begin{cases} [minimizza|massimizza] & \vec{C}' \cdot \vec{x} \\ tale\ che & A \cdot \vec{x}\ [\text{`}=\text{'}|\text{`}\leq\text{'}|\text{`}\geq\text{'}]\ b, \\ & \{\vec{x} \leq ub\}, \\ & \{\vec{x} \geq lb\}. \end{cases}$$

La sintassi della chiamata è:

```
[xmin,fmin,status,extra]=glpkmex
(sense,c,A,b,ctype,lb,ub,vartype,param,lpsolver,save);
```

sense: indica se il problema è di minimizzazione o di massimizzazione.

sense = 1: minimizzazione;

sense = -1: massimizzazione.

c: vettore colonna contenente i coefficienti della funzione obiettivo.

A: matrice contenente i coefficienti dei vincoli. A può essere una matrice sparsa (vedere '`help sparse`' in Matlab per i dettagli).

b: vettore colonna contenente il termine noto per ogni vincolo nella matrice dei vincoli.

ctype: vettore colonna contenente il senso di ogni vincolo nella matrice dei vincoli.

ctype (i) = 'F': variabile libera (illimitata);
ctype (i) = 'U': '$\leq$' variabile con limite superiore;
ctype (i) = 'S': '$=$' variabile fissa;
ctype (i) = 'L': '$\geq$' variabile con limite inferiore;
ctype (i) = 'D': variabile a doppio limite.

ctype è *case sensitive*

lb: vettore di dimensione pari al numero delle variabili contenente il limite inferiore per ciascuna delle variabili.

ub: vettore di dimensione pari al numero delle variabili contenente il limite superiore per ciascuna delle variabili.

vartype: vettore colonna contenente i tipi delle variabili.

vartype (i) = 'C': variabile continua;
vartype (i) = 'I': variabile intera.

param: struttura contenente alcuni parametri usati per definire il comportamento del risolutore. Per maggiori dettagli digitare '`help glpkparams`' in Matlab.

lpsolver: seleziona il risolutore da utilizzare per risolvere problemi LP.

lpsolver = 1: metodo del simplesso revisionato;
lpsolver = 2: metodo del punto interno.

Se il problema è un problema MIP questo argomento verrà ignorato.

save: salva una copia del problema se $save \neq 0$. Il nome del *file* non può essere specificato ed è "outpb.lp" per *default*. Il *file* di uscita ha il formato CPLEX LP.

xmin: vettore contenente il valore ottimo delle variabili.

fmin: valore ottimo della funzione obiettivo.

status: stato dell'ottimizzazione: specifica il tipo di soluzione trovata attraverso dei codici numerici. Tali codici sono elencati nel seguito rispettivamente per il metodo del simplesso, per il metodo del punto interno e per il metodo misto-intero.

- Metodo del simplesso -

valore	significato
180:	la soluzione è ottimale;
181:	la soluzione è ammissibile;
182:	la soluzione non è ammissibile;
183:	il problema non ha soluzione ammissibile;
184:	il problema non ha soluzione illimitata;
185:	lo stato della soluzione è indefinito.

- Metodo del punto interno -

valore	significato
150:	il metodo del punto interno è indefinito;
151:	il metodo del punto interno è ottimale.

- Metodo misto intero -

valore	significato
170:	lo stato è indefinito;
171:	la soluzione è intera ottimale;
172:	soluzione intera ammissibile ma la sua ottimalità non è stata dimostrata;
173:	nessuna soluzione intera ammissibile.

extra: una struttura contenente i seguenti campi:

lambda:	variabili duali;
redcosts:	costi ridotti;
time:	tempo (in secondi) usato per risolvere un problema LP/MIP;
mem:	memoria (in bytes) usata per risolvere un problema LP/MIP.

In caso di errore GLPKMEX restituisce uno dei seguenti codici (questi codici sono in **status**):

valore	significato

204: incapace di cominciare la ricerca;
205: limite inferiore della funzione obiettivo raggiunto;
206: limite superiore della funzione obiettivo raggiunto;
207: limite delle iterazioni raggiunto;
208: tempo limite raggiunto;
209: nessuna soluzione ammissibile;
210: instabilità numerica;
211: problemi con matrice di base;
212: nessuna convergenza (interna);
213: nessuna soluzione primaria ammissibile (pre-risolutore LP);
214: nessuna soluzione duale ammissibile (pre-risolutore LP).

Per maggiori informazioni sulle cause di questi codici far riferimento al manuale di GLPK.

Bibliografia

[1] E. Badouel e P. Darondeau. *Theory of regions*. In *Lectures Notes in Computer Science: Lectures on Petri Nets I: Basic Models, 1491: pagine 529-586, 1998*.

[2] T. Bourdeaud'huy e P. Yim. *Synthèse de réseaux di Petri à partir d'exigences*. In *Actes de la 5me conf. francophone de Modélisation et Simulation, pagine 413-420, Nantes, Francia, 2004*.

[3] M.P. Cabasino, A. Giua e C. Seatzu. *Identification of Petri nets from knowledge of their language*. In *Discrete Event Dynamic Systems, 17(4): pagine 447-474, 2007*.

[4] W.F. (IV) Doran e D.B. Wales. *The partition algebra revisited*. In *Journal of Algebra, 231(1): pagine 265-330, 2000*.

[5] M. Dotoli, M.P. Fanti e A.M. Mangini. *An optimization approach for identification of Petri nets*. In *Proc. IFAC WODES'06: 8th Work. on Discrete Event Systems, Ann-Arbor, MI, USA, 2006*.

[6] F. Garcia-Valles e J.M. Colom. *Implicit places in net systems*. In *Proc. of the 8th Int. Work. on Petri Nets and Performance Models, pagine 104-113, Saragozza, Spagna, 1999*.

[7] K. Hiraishi. *Construction of a class of safe Petri nets by presenting firing sequences*. In *Jensen, K., editore, Lectures Notes in Computer Science; 13th International Conference on Application and Theory of Petri Nets 1992, Sheffield, UK, volume 616, pagine 244-262. Springer-Verlag, 1992*.

[8] L. Lin e Y. Jiang. *The computation of hitting sets: Review and new algorithms*. In *Information Processing Letters, 86: pagine 177-184, 2003*.

[9] M.E. Meda-Campaña e E. López-Mellado. *Incremental synthesis of Petri nets models for identification of discrete event systems*. In *Proc. 41th IEEE Conf. on Decision and Control, pagine 805-810, Las Vegas, Nevada, USA, 2002*.

[10] M.E. Meda-Campaña e E. López-Mellado. *Required event sequences for identification of discrete event systems*. In *Proc. 42th IEEE Conf. on Decision and Control, pagine 3778-3783, Maui, Hawaii, USA, 2003*.

[11] T. Murata. *Petri nets: Properties, analysis and applications*. In *Proceedings of the IEEE, 77(4): pagine 541-580, 1989*.

[12] M.P. Cabasino, A. Giua e C. Seatzu. *Computational complexity analysis of a Petri net identification procedure*. In *2006 Int. Symposium on Nonlinear Theory and its Applications, Bologna, Italia, 2006*.

[13] M.P. Cabasino, A. Giua e C. Seatzu. *Linear Programming Techniques for the Identification of Place/Transition Nets*. In *Proc. 47th IEEE Conf. on Decision and Control, Cancun, Messico, USA, 2008*.

[14] J.H. van Schuppen. *System theory for system identification*. In *Journal of Econometrics, 118(1-2): pagine 313-339, 2004*.

[15] E. Mark Gold. *Complexity of automaton identification from given data*. In *Information and Control, 37(3): pagine 302-320, 1978*.

[16] D. Angluin. *Inference of reversible languages*. In *Journal of the ACM, 29(3): pagine 741-765, 1982*.

[17] R.S. Sreenivas. *On minimal representations of Petri net languages*. In *Proc. WODES'02: 6th Work. on Discrete Event Systems, pagine 237-242, Saragozza, Spagna, 2002*.

[18] R.K. Ahuja, T.L. Magnanti e J.B. Orlin. *Network Flows - Theory, Algorithms and Applications*.