Automi e Reti di Petri, A.A. 2002/2003

Progetto in ambito MATLAB di una funzione di visualizzazione dell'albero di raggiungibilità/copertura di una Rete di Petri.

Autore: **Diego Mura**, matr. 25643

Docente: A. Giua

Introduzione

Si è voluto implementare una rappresentazione grafica di un albero di raggiungibilità /copertura ottenuto mediante la funzione **tree** del pacchetto di toolbox di MATLAB sulle Reti di Petri realizzato da Riccardo Castrignano.
Tale funzione, dati in ingresso una matrice *Pre* , una *Post* e il vettore colonna $\boldsymbol{M_0}$ della marcatura iniziale, rende una matrice ***T*** nella quale, secondo un opportuno schema, sono riportate le caratteristiche dell'albero. Più precisamente, ad ogni nodo è associata una riga della matrice, nella quale sono riportati di seguito marcatura, nodo genitore, natura del nodo, transizioni abilitate e relativi nodi di arrivo. Vi è, inoltre, un'ultima riga, i cui primi due elementi indicano rispettivamente numero di posti e numero di transizioni della rete.

Utilizzando MATLAB 5.1, sono state create le seguenti funzioni:

- **plottree** [sintassi: PLOTTREE(T)]

  prende in ingresso la matrice T resa dalla funzione **tree** e dà una rappresentazione grafica completa dell'albero ad essa associata. Il codice di questa funzione è basato in parte su quello della funzione **treeplot** di MATLAB, anche se lo corregge, lo arricchisce e lo adatta al caso. E' la funzione principale, che richiama le altre come subroutines.

- **plottree1** [sintassi: PLOTTREE1(T)]

  analoga a **plottree**, fa però uso dei cappi, rendendo così una rappresentazione più compatta dell'albero di raggiungibilità/copertura.

- **depth** [sintassi: D=DEPTH(PARENT)]

  prende in ingresso il vettore PARENT dei genitori[1] e rende il vettore D delle profondità dei nodi. Qui, il nodo radice è considerato di profondità 0.

[1] Il vettore dei genitori presenta come i-esimo elemento l'indice del nodo genitore del nodo "i". I nodi sono intesi come indicizzati progressivamente da sinistra a destra e dall'alto in basso, riferendosi alla rappresentazione grafica

- **treelay** [sintassi: [X,Y,H]=TREELAY(PARENT,DEPTH)]

  dato il vettore PARENT dei genitori e quello DEPTH delle profondità dei nodi , rende i vettori X e Y delle coordinate alle quali centrare i rettangoli rappresentanti i nodi dell'albero. Rende inoltre l'altezza H dell'albero.
  A differenza dell'analogo **treelayout** ,che pone tutte le foglie alla base della figura indipendentemente dalla loro profondità, questa funzione pone a pari altezza tutti i nodi con la stessa profondità, equispaziandoli lungo l'asse orizzontale.

- **addbox** [sintassi: [Xb,Yb,COLOR]=ADDBOX(X,Y,W,H,NATURE)]

  Prende in ingresso i vettori X e Y delle coordinate dei punti in cui centrare i rettangoli, i vettori W delle loro larghezze e H delle loro altezze, il vettore NATURE del tipo di nodo; rende le matrici Xb e Yb , nelle quali ogni colonna contiene le coordinate relative ai vertici di un rettangolo (secondo il formato accettato dalla funzione **patch**), e il vettore COLOR dei colori con cui riempire i rettangoli.

- **dec2str** [sintassi: S=DEC2STR(X)]

  trasforma un vettore di interi X in una stringa S.
  Differisce dagli analoghi **num2str** e **int2str** in quanto qui gli elementi del vettore sono rappresentati in formato decimale e separati da un solo spazio.

Si noti che le ultime quattro funzioni, pur nascendo come subroutines di **plottree**, sono state rese volutamente il più generali possibile, in modo da poter essere utilizzate anche al di fuori dell'ambito del toolbox sulle Reti di Petri.

Sviluppo del progetto

In fase di ideazione del progetto, è stata subito chiara la necessità di avere una funzione che calcolasse i punti in cui rappresentare i nodi, in modo che il grafico risultasse leggibile. Pertanto, si è pensato di utilizzare la funzione **treelayout** di MATLAB; dopo una serie di prove volte a capirne il funzionamento, si è però visto che i risultati resi non sono attendibili. Da un'attenta analisi del suo codice, è emerso che è sbagliato il meccanismo di calcolo della profondità dei nodi. Inoltre, si è visto che le coordinate sono qui calcolate in modo che tutte le foglie stiano alla base della figura e i nodi interni vengano centrati orizzontalmente tra i propri figli, con il risultato di un grafico di forma pseudo-piramidale. Questo, oltre ad essere esteticamente sgradevole, comporta anche un problema: dato l'elevato numero di nodi duplicati con i quali si ha spesso a che fare quando si traccia l'albero di raggiungib./copertura di una Rete di Petri, e dato che questi sono sostanzialmente dei nodi-foglia, alla base del grafico si sarebbe trovata una moltitudine di nodi, con la conseguenza di doverli rappresentare mediante rettangoli troppo stretti.
E' stato quindi scritto da zero il codice della funzione **treelay**, che calcola le coordinate volute secondo uno schema di rappresentazione "a livelli": tutti i nodi con la stessa profondità sono e-

---

dell'albero. Si noti che il vettore dei genitori caratterizza completamente la struttura di un albero, in quanto dà sostanzialmente la mappatura padre-figlio.

quispaziati lungo l'asse X e posti a pari altezza nel grafico, indipendentemente dal fatto che siano foglie o meno.
Poiché questa funzione richiede il vettore delle profondità dei nodi, si è dovuta creare la funzione **depth**, che le calcola a partire dal vettore dei genitori, facilmente estrapolabile dalla rappresentazione tabellare dell'albero. Tale funzione individua innanzitutto le foglie (con un metodo preso di peso dal codice della funzione **treeplot**) e, a partire da ciascuna di queste, tramite due puntatori risale verso la radice, man mano aggiornando l'altezza dei nodi toccati nella scalata. Ognuno di questi processi finisce quando si incontra un nodo di cui è già stata calcolata la profondità oppure quando si giunge al nodo radice, che ha profondità 0.
A questo punto, è stato modificato il codice di **treeplot**[2], in modo che ricavi il vettore dei genitori dalla matrice –albero ***T*** presa in ingresso e chiami **depth** e **treelay** per ottenere le coordinate tra le quali tracciare gli archi. Dopo alcune prove, è risultato chiaro che la struttura dell'albero era correttamente visualizzata. Pertanto, la nuova funzione ottenuta è stata battezzata **plottree**.
Al fine di rappresentare i nodi, si è pensato di utilizzare dei rettangoli colorati all'interno dei quali scrivere la marcatura corrispondente e il cui colore fosse diverso a seconda del tipo di nodo; a questo proposito, si è scelto il celeste per i nodi duplicato, il giallo per i nodi di blocco e i verde per tutti gli altri.
Per realizzare ciò, si adopera la funzione **patch** di MATLAB**,** che traccia dei poligoni date le coordinate dei loro vertici e il vettore dei codici di colore ; di fronte alla necessità di calcolare tali valori (e contemporaneamente alleggerire il codice di **plottree)**, è stata creata la funzione **addbox**.
Questa, al fine di superare una serie di problemi legati alla definizione del vettore dei colori, imposta come mappa di colori della figura la mappa predefinita "colorcube"[3], la schiarisce e ne imposta la scala di valori; inoltre, gestisce a parte il caso speciale di soli tre rettangoli da rappresentare (si veda l'appendice A per maggiori dettagli).
Si noti che i vettori delle dimensioni dei rettangoli sono calcolati in **plottree** in base al massimo numero di nodi ad uno stesso livello di profondità.
A questo punto, si è scritta in **plottree** la parte di codice relativa all'inserimento delle marcature nei rettangoli. Queste vengono estratte dalla matrice-albero ***T*** e posizionate all'interno del nodo corrispondente tramite il comando **text**. Poiché questo lavora esclusivamente con le stringhe, si devono trasformare in stringa i vettori delle marcature.
Provati i comandi **num2str** e **int2str**, si è visto che calcolano il formato di rappresentazione dei numeri in base alla grandezza in modulo del maggior valore nel vettore da convertire; ciò faceva sì che la spaziatura tra gli elementi di una marcatura contenente il valore "Inf" fosse troppo larga, facendo debordare abbondantemente le cifre dai rettangoli dei nodi.
Modificando pertanto il codice di **int2str**, è stata creata la funzione **dec2str**, che trasforma un vettore in una stringa, rappresentandone gli elementi in formato decimale e separandoli con un solo spazio.
In **plottree** vengono poi sostituite con "w" tutte le ricorrenze di "Inf" nella stringa ottenuta, attraverso il comando **strrep**. L'effetto così ottenuto è quello di una rappresentazione delle marcature quanto più compatta possibile.
Inoltre, tramite alcune righe di codice di **plottree**, si è fatto in modo che il tipo e la dimensione del carattere con cui scrivere la marcatura dipendano dal massimo numero di nodi in uno stesso livello dell'albero: se questo è piccolo, i rettangoli dei nodi sono grandi e quindi è usato il carattere "arial" a 9 punti; se è grande, i rettangoli sono stretti, per cui viene usato un carattere a 7 punti di tipo "verdana" , facilmente leggibile anche se piccolo.

[2] Si noti che, così com'è, **treeplot** non funziona, in quanto si basa sulla funzione **treelayout**, che come si è visto dà valori sbagliati. Inoltre, nel suo codice è stato trovato un errore nella definizione della scala delle ordinate.
[3] Per conoscere i nomi delle mappe predefinite, vedere l'help di **graph3d**. Per la loro impostazione e visualizzazione, leggere l' appendice A.

Dopo di che, per ogni nodo vengono estratte da ***T*** una ad una le transizioni abilitate e il relativo nodo di arrivo; alla transizione, trasformata in stringa, viene aggiunta la "t" iniziale con il comando **strcat**. Infine, è posizionata vicino al punto medio dell'arco congiungente nodo in esame e nodo di arrivo. Lo scostamento dipende ancora una volta dal massimo numero di nodi presenti in un qualsiasi livello dell'albero: se tale valore è alto, gli archi degli eventi saranno molto vicini tra loro, per cui, onde evitare confusione, le transizioni relative sono scritte più vicino del solito ai punti medi. Ciò avviene anche nel caso di rette con inclinazione verticale o quasi.
Si era anche provato a definire lo scostamento in funzione dell'inclinazione dell'arco, ma si è visto che l'effetto porta confusione nel caso in cui da un nodo dell'albero sia abilitato un numero elevato di transizioni (gli archi molto inclinati risultano vicini, e le transizioni in tanti casi vengono posizionate vicino all'arco sbagliato!).
A questo punto, viene creata la legenda, utilizzando ancora dei rettangoli colorati tracciati con **addbox** e **patch** e delle stringhe posizionate con **text**.
Infine, vengono eliminati dalla figura gli assi cartesiani con le relative scale ed è attivata la funzione di **zoom**. Quest'ultima si rivela utile per alberi grandi, in cui le marcature spesso debordano dai rettangoli dei nodi e si sovrappongono: basterà infatti posizionare il puntatore del mouse sul nodo al quale si è interessati e premere il tasto sinistro per ingrandire l'area e leggere agevolmente la marcatura (vedi Fig. 4); il tasto destro del mouse riporta ad una visualizzazione normale.

Completata la scrittura dei listati, sono state stilate le guide (*help*), avendo cura di riportare innanzitutto la sintassi, poi la descrizione della funzione, dei suoi ingressi e delle sue uscite, infine eventuali note e accreditamenti.
Infine, per ciascuna funzione sono stati posti dei commenti vicino ai vari blocchi di codice e alle singole righe.

Successivamente, su consiglio del docente, sono state apportate alcune modifiche a **plottree**.
Innanzitutto, sono state centrate le marcature all'interno dei rettangoli: per far ciò, si è dovuto stimare, attraverso una serie di prove, l'ingombro orizzontale medio di un singolo carattere (che è risultato essere di 0.012 unità di misura, relativamente agli assi nella finestra grafica così come viene visualizzata al momento della sua apertura); quindi, si è calcolato il posizionamento della stringa in base alla sua lunghezza.
Sono state poi limitate le dimensioni massime dei rettangoli dei nodi, in modo da evitare che per alberi molto piccoli risultassero esagerate (in alcuni casi, si andavano addirittura a sovrapporre alla legenda). Anche in questo caso, i limiti sono stati calcolati sull'ingombro della marcatura.
E' stato quindi introdotto l'uso dei cappi, al fine di dare una rappresentazione più compatta dell'albero. La cosa si è dimostrata abbastanza difficile da implementare, in quanto è stato necessario affrontare un cambiamento di indicizzazione dei nodi. Più precisamente: con l'utilizzo dei cappi, vengono eliminati dall'albero completo alcuni nodi duplicato. Ciò ha imposto un nuovo posizionamento dei nodi rimanenti e di conseguenza una loro diversa indicizzazione. Questa, se da una parte non costituiva un problema per quanto concerne rettangoli e marcature, non andava però bene per le transizioni, che nella matrice T erano ancora descritte utilizzando gli indici di nodo relativi all'albero completo. Si è quindi dovuto mettere su un complesso sistema che tenesse traccia delle corrispondenze tra vecchia e nuova numerazione, e riscrivere il codice di **plottree** in tal senso (questa versione è poi stata chiamata **plottree1**).
Infine, si è valutata l'opportunità di adottare una struttura a grafo per due livelli contigui dell'albero, secondo il seguente schema: se da un nodo M(K) (nodo M al livello K) si va ad un nodo M'(K+1) che è duplicato di un nodo M'(K-1), viene eliminato M'(K+1) e viene messo un arco tra M(K) e M'(K-1).

Dopo un'analisi della realizzabilità della cosa, si è ritenuto che le difficoltà di implementazione probabilmente superano i vantaggi ottenuti. Questo perché una tale struttura comporta generalmente degli incroci tra archi, rendendo poco leggibile la rappresentazione.

Prove effettuate

Assumendo come buoni i risultati dati dalla funzione **tree**, si sono provate le funzioni **plottree** e **plottree1** (e conseguentemente tutte le altre create in sede di questo progetto) con i seguenti alberi di raggiungibilità/copertura (presenti nel *workspace* allegato al progetto):

1. T1.....fig. 4.16 del libro
2. T1bis..... " " " " , con marcatura iniziale M01bis=[2 2 1]
3. T_5_1.....esercitazione 5 del 2002, esercizio 1
4. Tbis_5_1..... " ", " ", marcatura iniziale M0bis_5_1=[1 0 0 1 0 0 2 1]
5. T_5_2.....esercitazione 5 del 2002, esercizio 2
6. Tbis_5_2..... " ", " ", marcatura iniziale M0bis_5_2=[3 2 1 0 0]
7. T_6_1_A.....esercitazione 6 del 2002, esercizio 1, prima rete
8. T_6_1_B.....esercitazione 6 del 2002, esercizio 1, seconda rete
9. T_6_1_C.....esercitazione 6 del 2002, esercizio 1, terza rete
10. T_mag.....fig 4.10(b) del libro (magazzino di capacità infinita)
11. Tbis_mag..... " " " , con marcatura iniziale M0bis_mag=[5]
12. T_pozzo.....rete con un posto, una transizione, un solo arco pre e nessun post, senza gettoni
13. Tbis_pozzo…" " ", con 12 gettoni
14. T_sorgente.....rete con un posto, una transizione, un solo arco post e nessun pre, senza gettoni

Per tutte le reti tranne la 2, la 6 e la 13, gli alberi raffigurati sono stati confrontati con la rappresentazione matriciale, non riscontrando errori in alcun caso. Le funzioni si sono dimostrate capaci di fronteggiare anche il caso speciale di un albero con tre soli nodi (si veda l'appendice A per maggiori dettagli) (reti 7 e 14) e quello di un albero con un solo nodo (rete 12).
Si è inoltre effettuato il confronto con i grafici tracciati a mano nelle esercitazioni (reti 3, 4, 5, 7, 8, 9) o sul momento (reti 10, 11, 12, 13 e 14), non notando alcuna differenza sostanziale.

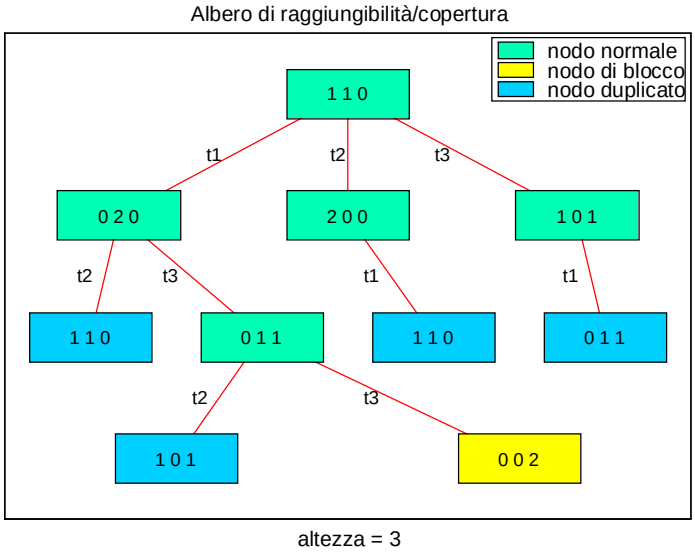

Fig. 1: albero di medie dimensioni (1)

Le reti marcate 2 e 6 sono invece state introdotte per valutare l'impatto dell'aumento delle dimensioni dell'albero sulla sua rappresentazione. Si è visto che per reti con pochi posti, le cui marcature sono corte, anche un albero con molti nodi viene ben visualizzato (si veda l'albero 2 in Fig. 2).

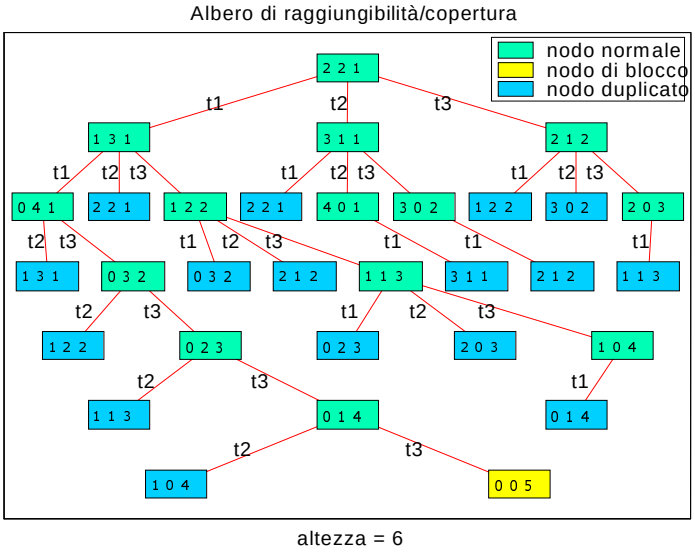

Fig. 2: albero di grandi dimensioni, ma con marcature brevi (2)

Se invece il numero di nodi aumenta ulteriormente e le marcature non sono più cortissime, queste eccedono i limiti del rettangolo cui corrispondono, spesso andando a sovrapporsi con quelle dei nodi adiacenti. Il problema è presto risolto ingrandendo il nodo d'interesse con il tasto sinistro del mouse (si veda a tal proposito l'albero 6 in Fig. 3 e in Fig. 4).

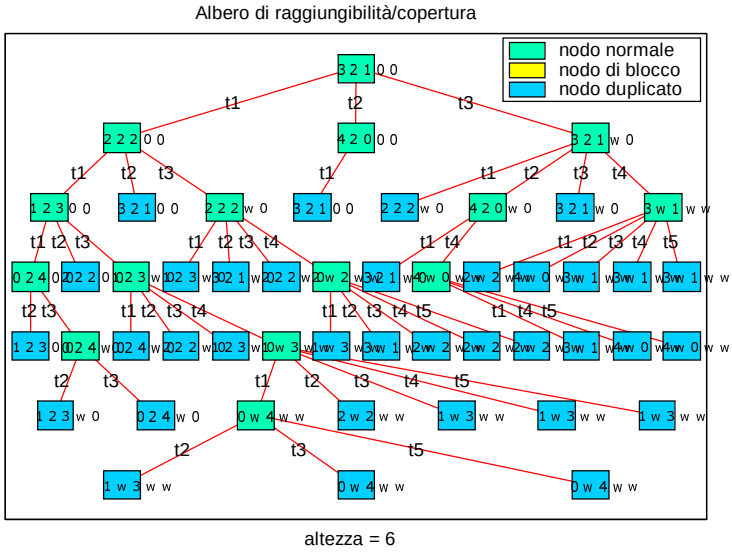

Fig. 3: albero di grandi dimensioni (6)

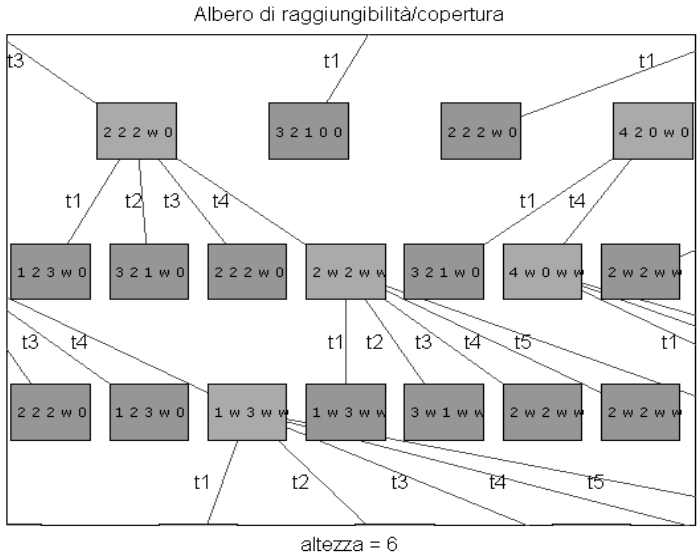

Fig. 4: effetto dell'ingrandimento dell'albero di grandi dimensioni (6)

L'albero 13 è stato studiato per valutare la massima altezza per la quale le marcature sono ancora ben leggibili all'interno dei nodi, che risulta pari a 12.

Infine, si è notato che quando si ingrandisce un'area si vedono le marcature anche nella cornice grigia della figura, ma questo è sicuramente un problema della funzione **zoom**.

Metodo di lavoro

Si è operato verificando ogni nuova modifica o aggiunta apportata ai codici, che venivano inizialmente salvati con un nome fittizio. Di fronte ad un errore, si apportavano le dovute correzioni, si salvava e si riprovava la funzione, fino a quando non si otteneva quanto voluto, senza problemi o messaggi di avviso nella *command window* di MATLAB. Infine, il listato era salvato con il nome definitivo della funzione.
Questa procedura garantiva di avere sempre sotto il nome definitivo il codice funzionante, punto di partenza per qualsiasi altra modifica. Ciò diventava però laborioso quando si provavano sul risultato complessivo gli effetti di variazioni o aggiunte in una subroutine, in quanto era necessario modificare la funzione superiore (**plottree**), correggendone le righe di codice nelle quali c'è la chiamata al sottoprogramma in questione con il suo nome fittizio e rinominando il tutto **plottree_2**.

E' stato spesso molto utile togliere i punto e virgola (;) in coda alle righe di codice in cui venivano definiti o modificati i dati interessati dall'errore che di volta in volta in volta si presentava: in questo modo tali dati venivano visualizzati nella *command window* e quindi, attraverso una analisi approfondita, si poteva risalire alla causa dell'errore.

Considerazioni sul progetto

Il programma qui creato è stato pensato per alberi di dimensioni medio-piccole e con marcature non troppo estese (reti con pochi posti). Perciò, per alberi molto grandi, pur funzionando, la sua resa a livello estetico e di leggibilità è fortemente compromessa.
Il fatto è che, per questi casi, è difficile pensare ad un buon metodo di visualizzazione: se le dimensioni dei caratteri fossero calcolate in base alle dimensioni dei rettangoli e alla lunghezza delle marcature, di modo che queste non escano mai dai bordi dei nodi, il risultato sarebbe sì un albero più ordinato, ma con scritte non leggibili. Di fatti, per come è concepito il programma, uno zoom ingrandirebbe la figura, ma non i caratteri di testo. Correlare le loro dimensioni al livello di ingrandimento della figura richiederebbe una conoscenza molto approfondita di MATLAB (sarebbe di fatto una cosa interattiva), e renderebbe comunque necessario uno zoom; in pratica, non si risolverebbe affatto il problema !
Per quanto riguarda il posizionamento delle etichette sugli archi, si è preferito accostarle bene, sempre alla sinistra del punto medio della retta, di modo che non ci possa essere confusione nei casi di rami molto fitti. Ciò, alcune volte, porta alla sovrapposizione della scritta con l'arco, ma, essendo questi di colore rispettivamente nero e rosso, la leggibilità non è compromessa, e con un ingrandimento si elimina il difetto.

Per quanto riguarda le subroutines di **plottree**, come già detto sono state rese volutamente il più generali possibile, in modo da poter essere adoperate anche al di fuori dell'ambito del toolbox sulle Reti di Petri.
Una funzione con buone possibilità di utilizzo in questo senso è **addbox**, anche se è piuttosto limitata in fatto di colori utilizzabili per i rettangoli: modificandone il codice, si può facilmente ovviare al problema (per la problematica legata alla definizione dei colori, vedere l'appendice A).

Appendice A

Per la definizione dei codici numerici corrispondenti ai colori scelti per rappresentare i vari tipi di nodo, sono stati riscontrati una serie di problemi di natura strettamente tecnica.
Dal momento che la loro descrizione e l'esposizione dei metodi usati per superarli avrebbero appesantito notevolmente la relazione, si è preferito riportarle in appendice, anche al fine di creare una sorta di guida per chi si accingesse ad analizzare il codice delle funzioni scritte per questo progetto.

Per realizzare una rappresentazione grafica dell'albero di aspetto gradevole, si volevano per i blocchi colori meno forti rispetto a quelli standard (indicati in MATLAB con 'g', 'y', 'c', ecc.), sui quali risaltassero comunque bene le marcature.
Dalla *command window*, attraverso il comando **colorbar**, si è visualizzata la mappa standard di colori, e su questa si sono scelti quelli da utilizzare, annotandone contemporaneamente i valori numerici corrispondenti. Nel listato di **addbox**, questi ultimi sono stati poi assegnati ai rettangoli, a seconda della natura del nodo corrispondente.
Da una prova della funzione così scritta si è però visto che il massimo e il minimo tra i valori scelti venivano assegnati come estremi della scala di colori utilizzata. Così, se inizialmente questa andava da 1 (blu notte) a 65 (rosso scuro) e si erano scelti i colori verde (30), celeste (21,5) e giallo (41), la scala numerica veniva reimpostata tra 21.5 e 41, facendo corrispondere così un verde sbiadito al 30, il blu notte al 21.5 e il rosso scuro al 41; i rettangoli visualizzati risultavano perciò di colori completamente inadatti.
Dalla *command window* si sono perciò visualizzate una ad una le mappe di colori predefinite, attraverso i comandi **colormap** e **colorbar**. La più idonea è risultata essere la mappa "colorcube", che oltre a contenere i colori scelti per i nodi, contiene anche il bianco, utilizzato per il rettangolo della legenda. Poiché però i colori risultavano troppo scuri, li si è schiariti mediante il comando **brighten**.
Nel codice di **addbox** si è quindi assegnata con **colormap** la mappa scelta, si è impostata su questa una scala arbitraria (da 1 a 10) attraverso il comando **caxis** e si sono assegnati ai rettangoli dei valori di colore a caso tra gli estremi della scala.
Dalla *command window* è stato fatto poi girare **addbox** con semplici dati in ingresso, è stato visualizzato tutto tramite il comando **patch** e, eseguendo **colorbar**, si è fatta apparire la mappa di colori relativa alla figura: la sua scala numerica era correttamente impostata tra 1 e 10.

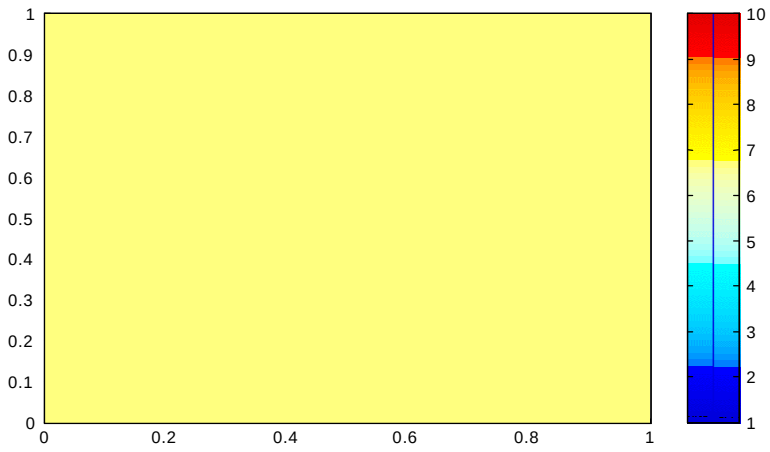

Fig. 5: visualizzazione della mappa di colori

I valori corrispondenti ai colori scelti per i nodi sono quindi stati annotati e assegnati definitivamente ai rettangoli all'interno del codice di **addbox**.

Inoltre, effettuando le prove sulla funzione **plottree**, ci si è resi conto che c'erano seri problemi con la rappresentazione di alberi costituiti da tre soli nodi. Ciò perché il vettore dei colori definito da **addbox** ha in questo caso lunghezza 3, per cui viene interpretato dalla funzione **patch** come una tripletta RGB, e non come vettore di colore di tre poligoni.
La soluzione adottata, pertanto, è stata la seguente: ogni qual volta debba essere rappresentata una serie di tre rettangoli, in **addbox** vengono automaticamente aggiunti ai loro dati quelli relativi ad un rettangolo fittizio posto nell'origine degli assi, di dimensioni nulle e colore bianco. In questo modo, il vettore dei colori ha lunghezza 4 e viene correttamente interpretato dalla funzione **patch**.

Appendice B

Successivamente allo sviluppo del progetto argomento di questa relazione, è stato affrontato il problema della rappresentazione di un grafo di raggiungibilità/copertura.
A partire dal codice di **plottree**, sono state perciò abbozzate due versioni di una funzione utile in tal senso:

- **plotgraph_beta** [sintassi: PLOTGRAPH_BETA(T)]

  è la versione più semplice: il grafo è ottenuto dall'albero semplicemente eliminando da questo i nodi duplicato e ritracciando gli archi che vi ci conducevano. Ciò porta ad una rappresentazione spesso squilibrata, che non ottimizza lo spazio della finestra grafica e che soffre delle stesse limitazioni che caratterizzano la funzione **plottree**, nonostante il numero di nodi da rappresentare sia sovente molto inferiore rispetto a quello dell'albero corrispondente.
  Si noti che prende ancora in ingresso la matrice T dell'albero, e non già la matrice G del grafo resa dalla funzione **graph** del pacchetto sulle Reti di Petri.

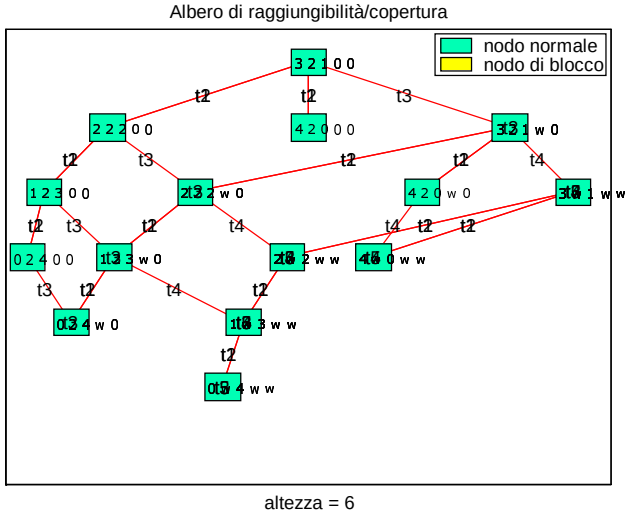

Fig. 6: grafo della rete (6) ottenuto con *plotgraph_beta*

- **plotgraph_alfa** [sintassi: PLOTGRAPH_ALFA(G)]

  è la versione più sofisticata: coordinate e dimensioni dei nodi sono calcolati in base al numero effettivo di nodi da rappresentare, producendo un grafo che ottimizza lo spazio della finestra e che, per i motivi sopra citati, è meno sensibile alla "esplosione" della rete.
  Prende in ingresso la matrice G resa dalla funzione **graph**.

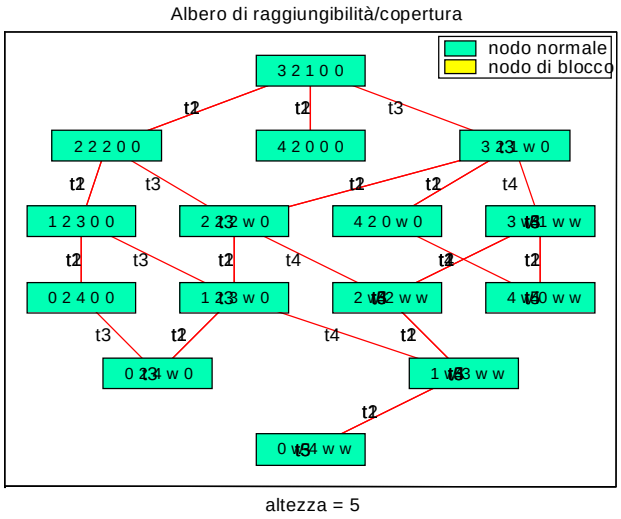

Fig. 7: grafo della rete (6) ottenuto con *plotgraph_alfa*

E' chiaro che sono solo degli abbozzi: nonostante in linea di principio funzionino, presentano una serie di problemi che rendono poco leggibile la rappresentazione del grafo.
Perché possano essere utilizzate, si dovrà pertanto:

- trovare il modo di indicare l'orientamento degli archi;
- impedire che gli archi intersechino i rettangoli dei nodi ;
- evitare la sovrapposizione degli archi quando due nodi si rimandano l'un l'altro;
- posizionare in maniera adeguata le etichette di transizione;
- rappresentare i cappi.

Per i cappi, si potrebbe adottare la soluzione sviluppata in **plottree1**, mentre per quanto riguarda il problema delle intersezioni degli archi, segue un'idea di cui deve però essere attentamente valutata la realizzabilità:

mantenendo la disposizione dei nodi fatta da **plotgraph_alfa**, bisognerebbe porre le coordinate di ogni punto disegnato nella figura in due vettori "contenitore".
Quindi, gli archi delle transizioni andrebbero tracciati non più come rette, ma come archi di parabola, secondo la seguente politica: si parte calcolando i vettori dei punti di un arco con convessità nulla (perciò degenere in una retta); se anche un solo punto è presente nel "contenitore", significa che c'è sovrapposizione con qualcosa che è già tracciato nella figura, per cui si ricalcolano le coordinate per una parabola a maggiore convessità. Si prosegue così finché non si trova un arco "buono", e rimane da gestire il caso in cui questo non si trovi.
Per quanto riguarda gli archi, si tratterebbe di sviluppare una funzione che, dati due punti di passaggio *P1* e *P2* e un valore *d*, calcoli le coordinate di una parabola che ha asse di simmetria perpendicolare alla congiungente *P1P2* e vertice distante *d* da questa.

Le difficoltà di tale progetto sono però molteplici: per dirne due, lo sviluppo della funzione che calcola i punti della parabola non è semplice, così come non lo è quello della gestione di casi con rami molto intricati.