Distributed Constraint Satisfaction Problem e arc-consistenza

Francesca Nocerino
Matricola 171107

Abstract. Molti problemi di cooperazione nell'ambito dei sistemi distribuiti possono essere modellati come problemi di soddisfacimento di vincoli in ambito distribuito. Con questo lavoro si vuole progettare e implementare un algoritmo distribuito che permetta a più agenti, che lavorano concorrentemente e in modo asincrono, di cooperare per raggiungere l'arc-consistenza in una rete di vincoli.

Keyword: vincoli, sistemi distribuiti, arc-consistenza, cooperazione tra agenti.

1.INTRODUZIONE

Sotto il nome di *cooperative distributed problem* (CDP) vengono riuniti numerosi problemi di varia natura, molti di questi problemi possono essere rappresentati in forma di *distributed constraint satisfaction problem* (DCSP). Ad esempio possono essere modellati come problemi di soddisfacimento di vincoli i problemi di pianificazione o i problemi di scheduling di risorse. È quindi evidente l'interesse nella risoluzione di questo tipo di problemi. Un CSP può essere risolto rendendo il grafo ad esso corrispondente arc-consistente ed effettuando una fase di ricerca. Di seguito viene descritto come più agenti, ciascuno a conoscenza solamente di una parte del problema globale, possono interagire al fine di rendere una rete di vincoli arc-consistente.
Nel paragrafo 2 sono richiamate alcune nozioni di base sui CSP, in particolare quelle di rete di vincoli e di arc-consistenza.
Nel paragrafo 3 è definito più dettagliatamente il problema in esame e viene illustrata la soluzione proposta affrontando sia l'aspetto legato a come ogni agente risolverà la sua porzione di DCSP, sia il problema di quali messaggi dovranno scambiarsi gli agenti e secondo quale protocollo.
Nel paragrafo 4 si parla dell'implementazione del sistema.
Nel paragrafo 5 vengono commentati i risultati della prove effettuate e viene introdotta un'euristica per partizionare un CSP su più agenti.

2. DEFINIZIONI

2.1 *Constraint Satisfaction Problem*

Un CSP (*Constraint Satisfaction Problem*) è rappresentato da:
- un insieme di *n* variabili;
- un dominio discreto per ciascuna delle n variabili;
- un insieme di vincoli.

La sua soluzione consiste nel trovare un assegnamento di valori per le variabili tale da garantire il soddisfacimento dei vincoli.
Senza perdita di generalità si può considerare il caso in cui tutti i vincoli siano binari, cioè coinvolgano solamente due variabili.
In questo caso un CSP può essere rappresentato come un grafo in cui a ogni nodo corrisponde una variabile, etichettata con il suo dominio, e, ad ogni arco tra un nodo *i* e un nodo *j*, corrisponde un vincolo binario C_{ij} tra le variabili x_i e x_j corrispondenti rispettivamente al primo e al secondo nodo (Figura 1).

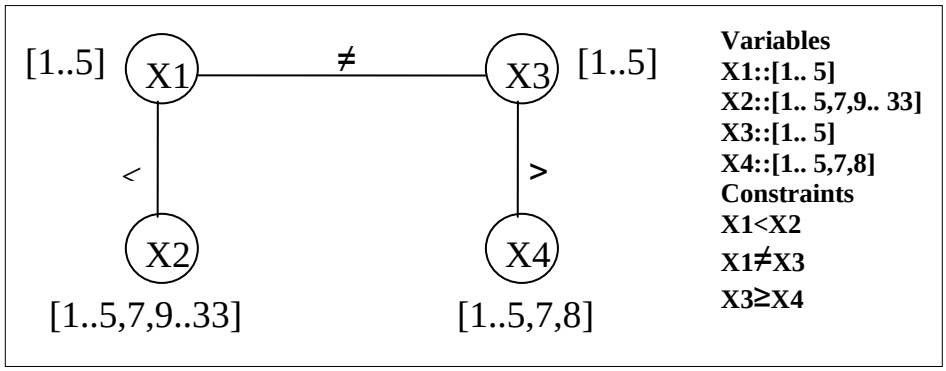

Fig. 1. Rete di vincoli.

2.1 Arc-consistenza

Una rete di vincoli è arc-consistente (Figura 2) quando per ciascuna coppia di variabili x_i e x_j legate da un vincolo C_{ij} si ha che per ogni valore appartenente al dominio di x_i esiste almeno un corrispondente valore appartenente al dominio di x_j tale che C_{ij} sia rispettato e viceversa.

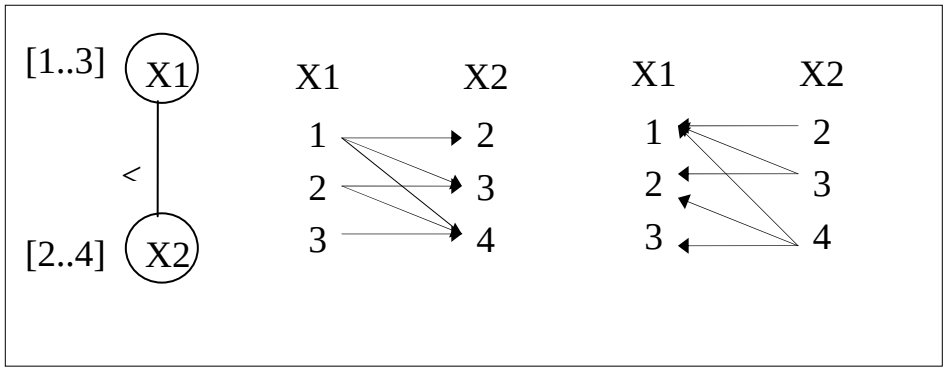

Fig. 2. Esempio di arc-consistenza.

3. ARC-CONSISTENZA E DCSP

3.1 *Distributed Constraint Satisfaction Problem*

Un CSP può essere partizionato su *k* agenti, con *k* compreso tra uno (caso locale) e *n* (caso in cui ad ogni agente è associata una sola variabile).

In generale ad ogni agente sono associati un sottoinsieme delle *n* variabili ed i relativi domini (v. par. 5).
Ogni variabile è associata a uno e un solo agente e tale agente è detto *responsabile* di quella variabile.
I vincoli riguardanti due variabili appartenenti allo stesso agente sono associati unicamente a tale agente e sono chiamati *vincoli interni* (o *internal constraint*).
Un vincolo che coinvolge due variabili legate a due agenti diversi è associato ad entrambi gli agenti e viene definito *vincolo esterno* (o *external constraint*).
Se due agenti A_1 e A_2 condividono un vincolo esterno si dice che A_1 è *vicino* (o *neighbour*) di A_2 e viceversa.
Il *neighbourhood* di un agente è quindi definito come l'insieme di tutti i suoi vicini.
Nella Figura 3 è riportato un esempio di CSP partizionato su due agenti.
Ad ogni agente è associato un CSP costituito dalle variabili di cui è responsabile e dai vincoli interni.
Per raggiungere l'arc-consistenza globale, però, non è sufficiente che ciascun agente renda la sua rete di vincoli arc-consistente.
È infatti necessario effettuare la propagazione anche sui vincoli esterni. Per fare ciò ciascun agente dovrà quindi cooperare con tutti i suoi vicini.

Una prima descrizione di come devono comportarsi gli agenti è la seguente:

```
while(la rete non è arc-consitente and c'è soluzione)
begin
      propaga localmente;
      coopera coi vicini;
end
```

Da quanto scritto sopra emerge che:

- ogni agente deve essere in grado di raggiungere localmente l'arc-consistenza (par. 3.2);
- ogni agente deve comunicare coi suoi vicini allo scopo di raggiungere l'arc-consistenza globale (par. 3.3);
- se uno o più agenti si accorgono che il proprio problema non ha soluzione devono avvisare tutti gli altri e l'algoritmo deve terminare (par. 3.4.1);
- gli agenti si devono coordinare per capire quando nessun valore può essere più eliminato dai domini delle variabili, cioè quando l'arc-consistenza totale è stata raggiunta in modo da far terminare l'algoritmo (par. 3.4.2).

Prima di spiegare come vengono rispettati i requisiti appena elencati è opportuno discutere della struttura della rete degli agenti.
Su tale struttura non vengono posti vincoli di nessun tipo. Non c'è quindi un ordinamento tra gli agenti.
Inoltre si possono presentare dei cicli e, perciò, se ne dovrà tenere conto per evitare che gli agenti vadano in deadlock.
Non imporre vincoli strutturali significa anche permettere che la rete di agenti non sia completamente connessa, cioè che ci siano due o più sottoinsiemi di agenti inidipendenti; questo caso corrisponde semplicemente ad avere due o più problemi distinti da risolvere, ciascuno associato a un sottoinsieme di agenti.

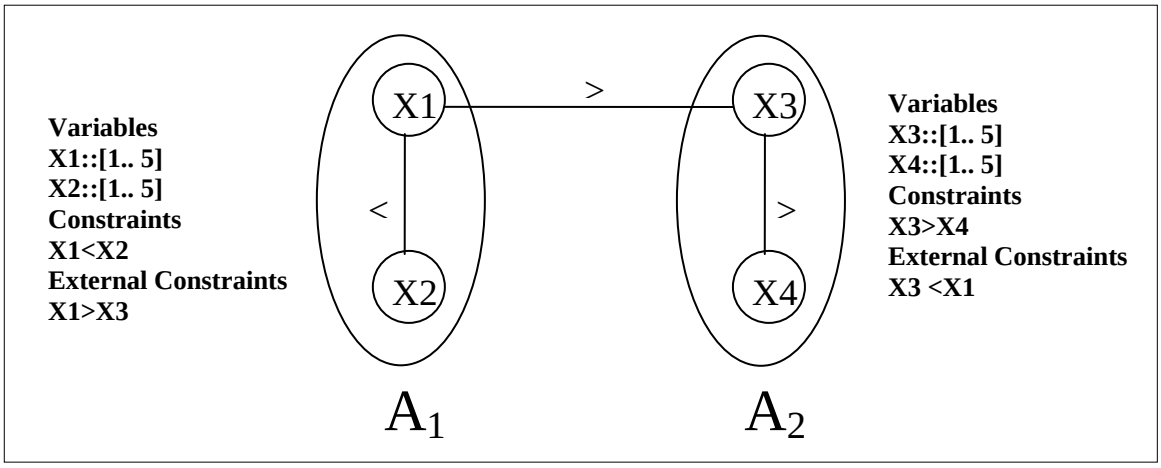

Fig. 3. Distributed CSP.

3.2 Arc-consistenza locale

Ogni agente, per raggiungere l'arc-consistenza locale, dovrà essere in grado di realizzare i seguenti passi:

```
inizializzazione: tutti i vincoli attivi
while (ci sono vincoli attivi)
begin
      scegli il primo vincolo attivo C
      for each variabile x su cui opera C
      begin
            for each valore del dominio di x
            begin
                  if(non esiste almeno un valore ammissibile per
                  l'altra variabile coinvolta nel vincolo)
                  then (togli tale valore dal dominio)
            end
            if (il dominio è stato modificato)
            then (risveglia tutti i vincoli addormentati che contengono
            la variabile corrente)
      end
      addormenta il vincolo C
end
```

3.3 Cooperazione tra gli agenti

Ogni agente deve cooperare con ciascun vicino per la propagazione dei vincoli esterni. È stato perciò definito un protocollo di comunicazione tra gli agenti.

Ogni agente deve conoscere tutti i suoi vicini e, per ognuno di essi, sapere quali sono le variabili di cui questo è responsabile e che sono coinvolte in un vincolo esterno con una delle sua variabili locali.

Non è necessario che un agente sappia di quali variabili sono responsabili tutti i suoi vicini, è sufficiente che sappia con quale agente deve comunicare per propagare ogni suo vincolo esterno.

Si è fatta l'ipotesi che i messaggi scambiati tra gli agenti arrivino, al massimo una volta, nell'ordine in cui sono stati inviati. Questa ipotesi è soddisfacibile usando TCP/IP.

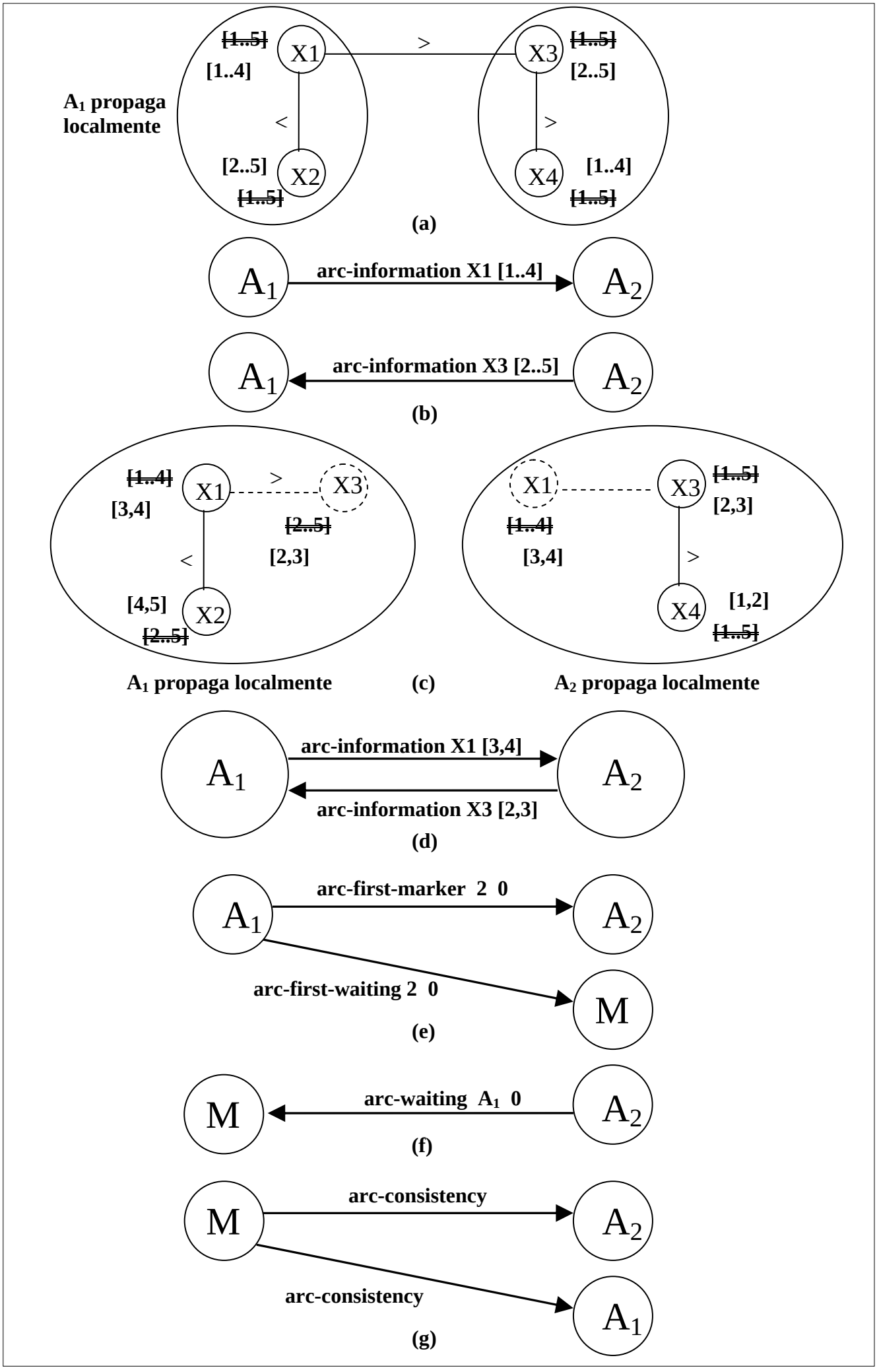

Fig. 4. Esempio di risoluzione di un DCSP.

Il protocollo pensato prevede che, inizialmente, ogni agente effettui la propagazione localmente (Figura 4-a). In seguito invia a ciascun vicino un messaggio

`arc-information X1 Dom1 X2 Dom2 ...`

contenente le informazioni su tutte le variabili locali coinvolte in un vincolo esterno con tale vicino (Figura 4-b). Effettuare sempre la propagazione locale prima di inviare messaggi **`arc-information`** ai vicini permette di diminuire il numero complessivo di messaggi che gli agenti si scambiano: nessun agente informerà mai un vicino che una sua variabile può assumere un certo valore se tale valore può essere eliminato dal dominio senza bisogno di interagire con un altro agente.

Dopo questa fase di inizializzazione, in cui ogni agente manda un messaggio ad ogni suo vicino, ciascun agente inizierà a ricevere dei messaggi **`arc-information`**. Alla ricezione di tali messaggi ogni agente propaga di nuovo localmente, ma tenendo conto dei vincoli esterni relativi alle variabili di cui ha ricevuto informazioni.
In seguito a queste propagazioni ciascun agente può aver modificato il dominio di alcune delle sue variabili locali (Figura 4-c).
Ad ogni vicino coinvolto in almeno un vincolo esterno la cui variabile locale è stata modificata verrà mandato un nuovo messaggio **`arc-information`** (Figura 4-d).
Quindi non vengono mai mandati messaggi **`arc-information`** se non sono necessari: se si mandasse un numero inferiore di messaggi alcuni agenti potrebbero non eliminare dai domini delle proprie variabili valori che dovrebbero essere eliminati per il raggiungimento dell'arc-consistenza.

3.4 Terminazione

Gli agenti possono terminare per due ragioni:
- il problema non ammette alcuna soluzione, cioè almeno un agente si accorge che il dominio di almeno una delle variabili di cui è responsabile è vuoto;
- l'arc-consistenza è stata raggiunta, cioè non è più possibile, per nessun agente, eliminare dei valori dal dominio di nessuna variabile.

Per discutere delle cause di terminazione è necessario introdurre il concetto di stato di un agente.
Ogni agente ha uno stato, i possibili stati in cui può trovarsi un agente sono:

- *waiting*: l'agente non sta effettuando localmente alcuna propagazione ed è in attesa che gli arrivino dei messaggi **`arc-information`** dai vicini;
- *not waiting*: l'agente sta propagando localmente o inviando ai vicini dei messaggi **`arc-information`**;
- *consistent*: in questo stato l'agente sa che è stata raggiunta la consistenza;
- *impossible*: in questo stato l'agente sa che il problema non ammette soluzione.

In Figura 5 è riportato il diagramma di transizione degli stati.
Dire che l'arc-consistenza è stata raggiunta significa dire che tutti gli agenti si trovano in stato di *waiting*.

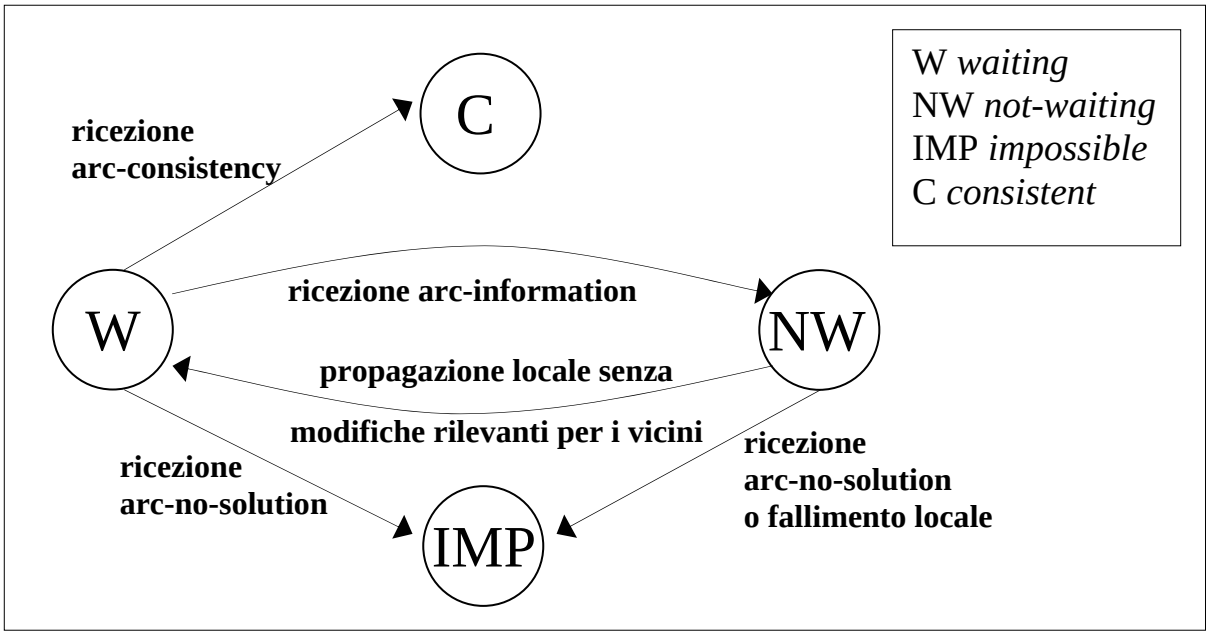

Fig. 5. Stati di un agente.

3.4.1 Problema impossibile

Se durante la propagazione locale un agente si accorge che non c'è soluzione invia a tutti i suoi vicini un messaggio

`arc-no-solution`

dopo aver aggiornato il suo stato, che diventa *impossible*, e prima di terminare.
Alla ricezione di un messaggio **`arc-no-solution`** ogni agente, se il suo stato non è già *impossible*, cambia il suo stato, avvisa i vicini e termina a sua volta.
Alle ricezione di un **`arc-no-solution`** è necessario che un agente verifichi il proprio stato, per evitare di mandare messaggi inutili, per due ragioni.
La prima è che la conclusione che il problema è impossibile può essere raggiunta da più agenti autonomamente, la seconda è che, non essendo stati imposti vincoli strutturali alla rete, un stesso agente può essere avvisato da più di un vicino.

3.4.2 Raggiungimento dell'arc-consistenza

Fino a questo momento non è stato affrontato il problema di come gli agenti rilevano che è stata raggiunta l'arc-consistenza.
L'arc-consistenza è raggiunta quando tutti gli agenti sono in attesa di un messaggio e nessun messaggio è più in transito. Si è detto che in questo caso tutti gli agenti si trovano in stato di *waiting*.
Per rilevare lo stato globale del sistema sarà necessario fare uso di un protocollo del tipo descritto in [1].
L'algoritmo di Chandy e Lamport, a grandi linee, è il seguente:

- uno o più processi prendono l'iniziativa di registrare il proprio stato;
- ogni processo *p*, dopo aver memorizzato il suo stato, invia ad ogni processo *q*, tale che esiste un canale di comunicazione *c* tra *p* e *q*, un *marker*, prima di

inviargli qualsiasi altro messaggio. Il *marker* deve contenere il numero *N* dei messaggi che *p* ha ricevuto da *q* fino a quel momento;

- quando *q* riceve un *marker* da *p* confronta *N* con *M*, numero dei messaggi che ha inviato a *p* fino al momento in cui ha ricevuto il *marker*. Se *N* è uguale a *M* il processo *q* registra il suo stato, registra lo stato di *c* come vuoto e invia a sua volta i *marker*. Se *N* è minore di *M* lo stato di *c* memorizzato da *q* corrisponde agli *M-N* messaggi inviati da *q* e non ancora ricevuti da *p*;

è inoltre necessaria la presenza di un ulteriore processo che agisca da *monitor*, come descritto ad esempio in [2], per mettere insieme gli stati di tutti i processi in modo da gestire lo stato globale del sistema.

Seguendo la traccia di [1], quando un agente A_1, in seguito alla propagazione locale, non deve mandare **`arc-information`** a nessuno prende l'iniziativa di registrare il suo stato, che è ovviamente *waiting*, e invia ai suoi vicini il messaggio

`arc-first-marker num-mess-ricevuti numero-iniziativa`

per informarli che per lui potrebbe essere stata raggiunta la consistenza (Figura 4-e).
Con questo messaggio A_1 fa sapere al destinatario che ha ricevuto da lui un certo numero di messaggi e gli dice anche quante volte si è già trovato in stato di *waiting*.
L'agente A_1, oltre a inviare i messaggi **`arc-first-marker`** ai vicini, invia al *Monitor* il messaggio

`arc-first-waiting my-id numero-iniziativa`

Alla ricezione di un **`arc-first-marker`** un agente A_2 verifica se i messaggi che ha inviato sono stati tutti ricevuti e, in tal caso, se è anche lui in stato di *waiting* invia al *Monitor* un messaggio

`arc-waiting init-id numero-iniziativa`

(Figura 4-f) altrimenti gli invia il messaggio

`arc-not-waiting init-id numero-iniziativa`

Se invece ci sono dei messaggi inviati da A_2 e non ancora ricevuti da A_1 viene inviato al *Monitor* il messaggio

`arc-message-travelling init-id numero-iniziativa`

A_2 inoltre, alla ricezione di un **`arc-first-marker`**, invia a ogni suo vicino un messaggio

`arc-marker num-mess-ricevuti init-id numero-iniziativa`

alla ricezione del quale ogni agente si comporta come alla ricezione di un **`arc-first-marker`**.
Per evitare una proliferazione di messaggi ogni agente, ovviamente, non manda il *marker* al vicino che lo ha mandato a lui.
Un agente potrebbe comunque ricevere due volte lo stesso *marker* da due vicini diversi, poiché non sono stati posti vincoli di nessun tipo sulla struttura della rete, e potrebbero perciò esserci dei cicli che portano al deadlock.
Per evitare un loop infinito in cui gli agenti continuano a far girare *marker* nella rete ogni agente memorizza i *marker* inviati e non invia mai due volte un marker con gli stessi **`init-id`** e **`numero-iniziativa`**.
Il *Monitor* memorizza lo stato di tutti gli agenti e, appena si accorge che essi si trovano tutti contemporaneamente nello stato di *waiting*, li avvisa che è stata raggiunta la

consistenza col messaggio **`arc-consistency`**, alla ricezione del quale ogni agente sa che può terminare (Figura 4-g).
Nel protocollo appena descritto si suppone che tutti gli agenti siano attivi dall'inizio; nell'implementare l'algoritmo tale condizione può essere verificata aggiungendo una fase di coordinazione iniziale con l'invio di messaggi di tipo **`arc-start`**.

4. IMPLEMENTAZIONE

Il sistema realizzato è costituito da 49 classi suddivise in cinque package:
- ***CSP***, per realizzare il risolutore di vincoli che deve risiedere su ogni nodo;
- ***CSPLexerParser***, ha la responsabilità dell'analisi sintattica e semantica del linguaggio usato per definire le variabili, i loro domini e i vincoli. È quindi possibile usare un linguaggio analogo a quello usato in Figura 1 e in Figura 3; la definizione della grammatica di tale linguaggio è riportata in Figura 6;
- ***DistributedCSP***, per la creazione, la cooperazione e la coordinazione degli agenti e del *monitor*;
- ***Exception***, per la gestione di tutti i casi di errore, ad esempio nell'uso del linguaggio per definire i problemi o nella comunicazione tra gli agenti;
- ***CSPApplication***, contiene le classi per l'interfaccia utente (per il funzionamento dell'applicazione si rimanda al manuale).

Per una descrizione dettagliata delle varie classi si rimanda alla javadoc.

alfabeto:={natural, string, =, !=, >, <, >=, <=, ::, [,], ,, .. , Variables, Constraint, External Constraint}

natural rappresenta un numero naturale.
string è una stringa alfanumerica che inizia con un carattere.
natural e string possono essere definiti mediante un'altra semplice grammatica.
Variables, Constraints, External Constraints sono parole chiave.

<distributedConstraintProblem>::= <constraintProblem>
External Constraints
< constraintList >
<constraintProblem>::= Variables
<variableList>
Constraints
<constraintList>
<variableList>::= <variable> | <variable> < variableList >
<variable>::= <varName> :: <domain>
<varName>::=string
<domain>::= [<domContent>]
<domContent>::= <domElem>| <domElem>,<domContent>
<domElem>::= natural|natural..natural
< constraintList > ::= <constraint> | <constraint> < constraintList >
<constraint>::=<varName><op>< varName >
<op>::= =|!=| >| <| >=| <=

Fig. 6. Grammatica del lingauggio.

5. ESPERIMENTI

Dai test realizzati si è visto che più i problemi associati ad ogni agente sono indipendenti e complessi più l'algoritmo distribuito si rivela migliore di quello sequenziale.
Per quanto riguarda la complessità del problema è intuitivo che, nel caso in cui ad ogni agente sia associato un problema di dimensioni ridotte, il tempo richiesto ogni volta dalla propagazione locale è poco, decisamente inferiore al tempo speso dagli agenti a inviare e ricevere messaggi per la coordinazione.
L'indipendenza tra i problemi è legata al modo in cui le variabili sono assegnate agli agenti.
Sebbene nella maggior parte dei problemi reali modellabili come DCSP le variabili siano assegnate a priori agli agenti è stato implementato un algoritmo che fa uso di una semplice euristica per partizionare un CSP in problemi il più possibile indipendenti.

5.1 Euristica per il partizionamento di un CSP

Partizionare un CSP significa partizionere l'insieme delle n variabili in k classi. Il numero delle possibili partizioni è dato dai numeri di Stirling di seconda specie, e non si può certo pensare di generare tutti i possibili partizionamenti e confrontarne la bontà.
L'algoritmo realizzato è un algoritmo Greedy.
Prevede n iterazioni, una per ogni variabile. Ad ogni passo si deve decidere a quale agente assegnare una certa variabile x_i, con $i=1…n$.
x_i viene assegnata all'agente A_{pref} per cui $pref(x_i, A_j)$, con $j=1…k$, è maggiore.
$pref(x_i, A_j)$ è la somma pesata di tre fattori:
- $(n-|A_j|)/n$, il cui scopo è quello di mantenere il carico di ogni agente il più possibile bilanciato, il caso ideale è quello in cui ad ogni agente sono assegnate n/k variabili;
- somma dei valori dei vincoli che sarebbero interni assegnando x_i a A_j diviso il numero di tali vincoli;
- somma dei valori, presi con segno negativo, dei vincoli che sarebbero esterni assegnando x_i a A_j diviso il numero di tali vincoli.

Il valore dei vincoli di tipo = è maggiore del valore dei vincoli di tipo > ≥ < ≤ che è a sua volta maggiore del valore dei vincoli di tipo ≠.
Il risultato di quest'algoritmo cambia a seconda dei pesi attribuiti ai tre fattori.
Inoltre, non essendo effettuato backtracking, il risultato dipende dall'ordine con cui vengono considerate le variabili e si è verificato che, se le variabili sono considerate per valori crescenti della somma dei valori dei vincoli ad esse associate, le partizioni ottenute sono migliori.

6. CONCLUSIONI

La soluzione proposta è corretta. Lo sviluppo del progetto è stato orientato al cercare di limitare il numero di messaggi scambiati tra gli agenti.
Ciò ha comportato un maggior onere a livello computazionale per ogni agente.
Infatti, dati due agenti A_1 e A_2 legati da un vincolo esterno, si ha che sia A_1 che A_2 almeno una volta devono propagare tale vincolo. Una soluzione alternativa avrebbe

potuto prevedere che A_1 e A_2 contrattassero per risolvere il vincolo lasciando soltanto a uno dei due il compito di effettuare la propagazione.
In questo modo, però, il numero di messaggi necessari per raggiungere l'arc-consisteza sarebbe, nel caso migliore, raddoppiato e, soprattutto, l'esecuzione risulterebbe sequenzializzata.
Per quanto riguarda invece i messaggi scambiati per la coordinazione, si ha che essi sono in numero superiore al numero minimo necessario per la corretta terminazione dell'algoritmo.
Ciò accade perché l'unico stato globale che è necessario rilevare è quello in cui tutti gli agenti sono *waiting*, mentre di fatto questo non è l'unico stato del sistema rilevato.
Infatti, poiché ogni messaggio contiene l'identificatore di chi ha preso l'iniziativa e il numero di iniziativa, se un agente invia al *monitor* **`arc-not-waiting`** o **`arc-message-travelling`** potrebbe anche non inviare i *marker* agli altri agenti, gli altri agenti non registrerebbero il proprio stato e il monitor saprebbe comunque che non è ancora stata raggiunta la consistenza.
Il motivo per cui si è preferito avere lo stato globale completo, anche quando non sembrerebbe necessario, è che, attualmente, viene resa la rete arc-consistente, ma non viene risolto il DCSP, mentre per calcolare una soluzione si deve effettuare anche una fase di ricerca, in cui gli agenti assegnano un possibile valore ad alcune variabili, rendono la rete nuovamente arc-consistente e eventualmente fanno backtracking. In questa fase, durante la propagazione dei vincoli, serve memorizzare anche altri stati globali in cui si può trovare il sistema.
Il protocollo di comunicazione tra gli agenti è quindi stato deciso tenendo anche conto di possibili estensioni dell'applicazione.

BIBLIOGRAFIA

[1] Chandy, Lamport. Distributed snapshots: determining global states of distributed systems. TOCS, 3(1): 63–75.

[2] Misra, Chandy. Termination detection of diffusing computations in communicating sequential processes. ACM Trans. Program. Lang. Syst. 4, 1 (Jan. 1982), 37-43.

[3] Dijkstra, Scholten. Termination detection for diffusing computations. Znf. Proc. Lett. 12, 1 (Aug. 1980), 1-4.

[4] Yokoo, Ishida, Durfee, Kuwabara. Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving. 12th IEEE International Conference on Distributed Computing System '92, pp.614-621.

[5] Bessière, C. (1994). Arc-consistency and Arc-consistency again. Artificial Intelligence, 65(1): 179-190.