POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Anno Accademico 2011/2012

Un simulatore multiagente basato su DCOP per lo studio della gestione di bacini idrici

Relatore:
Prof. Francesco Amigoni
Correlatore:
Ing. Matteo Giuliani

Studente:
Samuele Granata
Matr. 766507

Sommario

Il seguente lavoro di tesi presenta un tool in grado di simulare più scenari idrici, generati in base a rischieste dell'utente o mediante una scelta casuale.
I problemi sono stati formalizzati come DCOP (Distributed Constraint Optimization Problem), modellizzazione matematica che permette di descrivere problemi di ottimizzazione mediante l'utilizzo di variabili, il cui valore appartiene ad un dominio fissato, di agenti, che si occupano di assegnare un valore alle variabili, e di vincoli, che definiscono i valori ammissibili per tali variabili.
Il tool consente lo svolgimento di simulazioni singole o multiple e la risoluzione dei problemi generati mediante quattro algoritmi distribuiti per risoluzione di problemi DCOP, riportando poi i risultati mediante funzioni di reportistica e grafici.

Indice

1. Introduzione

Il seguente lavoro di tesi trae spunto dalla pubblicazione di Yang, Cai e Stipanovic (2009), intitolata "A decentralized optimization algorithm for multiagent system-based watershed managment " [1].
Gli autori si proponevano di modellizzare un semplice sistema idrico, composto di un fiume principale, affluenti, città e dighe, come un sistema multiagente, dove ogni agente descrivesse una città o una diga e avesse controllo su una sola variabile, che rappresentasse rispettivamente la quantità di acqua prelevata o la portata di acqua rilasciata.
Successivamente Giuliani, Castelletti, Amigoni e Cai, nell'articolo "Multi-Agent Systems optimization for distributed watershed managment" (2012) [2], hanno comparato le prestazioni in termini di qualità delle soluzioni al variare di numerosi parametri e al variare della modalità di risoluzione del problema, cioè in modo centralizzato o in modo distribuito.
Il seguente lavoro di tesi estende tali articoli presentando un tool in grado di simulare più scenari idrici, in base alle esigenze dell'utente, o generando scenari casuali in base a parametri immessi sempre dall'utente stesso.
Inoltre, rispetto al lavoro di Yanget al., i problemi sono stati formalizzati come DCOP (Distributed Constraint Optimization Problem), modellizzazione matematica utilizzata per problemi distribuiti che verrà presentata nel Capitolo 3.
Il programma consente di svolgere simulazioni singole o simulazioni multiple e di risolvere i problemi generati mediante algoritmi distribuiti per risoluzione di problemi DCOP.
Tale programma può risultare un valido aiuto a chi si occupa di ricerca in questo settore e necessita di un simulatore semplice, completo e affidabile.
La seguente trattazione si svolge in diversi capitoli.
Il Capitolo 2 presenta una panoramica sull'intelligenza artificiale, i sistemi multi-agente e il loro utilizzo nel modellare sistemi idrici.
Il Capitolo 3 espone in maggiore dettaglio la formalizzazione degli ambiti di studio di tali discipline, illustrando i fondamenti matematici per quanto riguarda i problemi CSP, DCSP e DCOP.
I concetti illustrati in questo capitolo saranno fondamentali per comprendere appieno il resto di questa tesi.
Il Capitolo 4 descrive in dettaglio il lavoro svolto in termini ancora matematici, formalizzando i vincoli del problema e fornendo una panoramica sul tipo di agenti in gioco, attraverso descrizioni formali ed esempi di casi studio.
Il Capitolo 5 si occupa di illustrare in dettaglio il programma sviluppato, descrivendo le tecnologie utilizzate e spiegando lo sviluppo del codice e le funzionalità implementate mediante schemi ed esempi.
Il Capitolo 6 è dedicato alle prove sperimentali effettuate e all'analisi dei dati raccolti

mediante l'utilizzo del tool sviluppato.
Infine il Capitolo 7 traccia le conclusioni e suggerisce nuovi possibili sviluppi del lavoro di tesi svolto.

2. Stato dell'arte: Intelligenza artificiale e sistemi multiagente

2.1 Intelligenza artificiale

L'espressione "Intelligenza Artificiale" (Artificial Intelligence) fu coniata nel 1956 dal matematico americano John McCarthy, durante uno storico seminario interdisciplinare svoltosi nel New Hampshire. Secondo le parole di Marvin Minsky, uno dei pionieri della IA (Intelligenza Artificiale), lo scopo di questa nuova disciplina sarebbe stato quello di "far fare alle macchine delle cose che richiederebbero l'intelligenza se fossero fatte dagli uomini".
Nel suo aspetto puramente informatico, l'IA comprende la teoria e le tecniche per lo sviluppo di algoritmi che consentano alle macchine (tipicamente elaboratori) di mostrare un'abilità e/o attività intelligente, almeno in domini specifici, occupandosi innanzi tutto dell'individuazione dei modelli (appropriata descrizione del problema da risolvere) e degli algoritmi (procedura effettiva per risolvere il modello), che cercano di soddisfare tutte le necessità tecnologiche attuali, in termini di efficienza e prestazioni [3].
Tuttavia l'IA ha una visione più larga e abbraccia diversi ambiti interdisciplinari quali la filosofia, l'informatica, la matematica, le neuroscienze, la psicologia e la teoria del controllo.
Storicamente è stato utilizzato un approccio sistematico per descrivere l'IA, attraverso 4 i punti di vista [3]. Questi ultimi corrispondono a degli obbiettivi che l'IA cerca di raggiungere nel corso di continue attività di ricerca:

- Il pensare umanamente: questo primo obbiettivo di ricerca si preoccupa come primo passo di determinare come il cervello umano elabora i pensieri e quali sono i suoi meccanismi per crearli.
  La via per ottenere questo sono l'introspezione, cioè il tentativo di osservare i pensieri durante le loro creazione, la sperimentazione psicologica, cioè l'osservazione della persona in azione, e l'imaging cerebrale, ossia l'osservazione del cervello in azione. Oggi questa area è un campo interdisciplinare che lavora a stretto contatto con le scienze cognitive.
- L'agire umanamente: questa obbiettivo venne formalizzato nel 1950 con il famoso test di Alan Turing, il quale propose un test per definire operativamente il concetto di intelligenza [4]. Per superare tale un computer avrebbe dovuto ingannare un esaminatore umano, facendogli credere, in seguito alle sue risposte, di essere un umano. Semplicisticamente, prendendo una citazione dello studioso Kurzweil, questo settore di ricerca è riassumibile in questa massima: “L'agire umanamente è l'arte di creare macchine che svolgano attività che richiedono intelligenza quando vengono svolte da persone” [5]. Appartengono quindi a questo ambito discipline come lo studio dell'interpretazione del linguaggio naturale, la rappresentazione della conoscenza per memorizzare dati con cui gli elaboratori entrano in contatto, il

ragionamento automatico per utilizzare i dati e le informazioni memorizzate in precedenza e l'apprendimento per adattarsi a nuove circostanze. In questa area hanno particolare importanza attualmente discipline come la robotica e la visione artificiale, che si occupano di sviluppare tecnologie per percepire gli oggetti nell'ambiente circostante e manipolarli fisicamente.

- Il pensare razionalmente: per perseguire questo obbiettivo i ricercatori lavorano a stretto contatto con la logica matematica. L'obbiettivo è quindi il riuscire a descrivere la conoscenza in termini formali, secondo cioè i linguaggi matematici propri di questa disciplina, in modo che in seguito alla elaborazioni delle regole e dei fatti presenti nella base di conoscenza della macchina, quest'ultima deduca qual è il modo razionale e quindi corretto di comportarsi, elaborando una strategia corretta. Particolare rilevanza in questo ambito hanno le tecniche di soft computing, legate all'apprendimento automatico, il cosiddetto “machine learning”.
- L'agire razionalmente: questo obbiettivo di ricerca è il naturale proseguimento del precedente, e consiste nell'attuare la strategia migliore per risolvere un problema specifico. Tale strategia è stata elaborata dalla macchina ed è la migliore tra le strategie attuabili dalla macchina. Quest'ultima viene considerata come agente, cioè come elemento del sistema che agisce, cioè compie delle azioni. Un'agente razionale agisce nel modo più idoneo per ottenere il miglior risultato oppure, in condizioni di incertezza, il miglior risultato atteso.
  Particolare enfasi è qui posta sul corretto funzionamento delle inferenze logiche e sugli attuatori che dovranno poi convertire tale pensiero in azioni.
  Questo tipo di approccio presenta almeno due vantaggi rispetto ai precedenti.
  Innanzi tutto la correttezza dell'inferenza è solo uno dei meccanismi possibili di raggiungere la razionalità. Quindi nel caso l'inferenza non andasse a buon fine ci sarebbero altre vie percorribili.
  In secondo luogo questo tipo di approccio meglio si lega ai recenti e continui sviluppi scientifici riguardo il comportamento e il pensiero umano. Infine bisogna però far notare che la razionalità perfetta è un requisito irraggiungibile in sistemi complessi a causa di rigidi vincoli computazionali.

Come si nota è possibile identificare una doppia classificazione nel catalogare gli obbiettivi di ricerca, la prima in base alle azioni svolte, la seconda in base alla somiglianza più o meno accentuata al comportamento umano.
Inoltre bisogna specificare il concetto di razionalità che può essere inteso nella tendenza della macchina a svolgere le migliori azioni rispetto alle proprie conoscenze pregresse e acquisite durante il funzionamento.

2.2 Agenti autonomi e sistemi multiagente

Un sistema multiagente o (sistema ad agenti multipli) è un insieme di agenti situati in un certo ambiente che interagiscono tra loro.
Un agente è un'entità caratterizzata dal fatto di essere, almeno parzialmente, autonoma, sia essa un programma informatico, un robot, un essere umano, e così via.
Il principio di funzionamento dell'agente mira a raggiungere uno scopo, un obbiettivo per il quale è stato progettato. Esso viene disegnato dal progettista in modo da non conoscere alcun algoritmo risolutivo. In altre parole l'agente conosce quindi il suo scopo ma non il modo completo per ottenerlo. Esso ha solo delle proprietà, delle azioni, delle conoscenze, anche apprese durante il funzionamento, che gli permettono di raggiungere il suo obbiettivo. Per questo esso è innanzi tutto definito autonomo (o anche pro-attivo), dato che dovrà cercare mediante le sue conoscenze e autonomamente il modo per raggiungere il suo obbiettivo [6] [7].
L'agente è però collocato in un ambiente complesso caratterizzato dalla presenza di altri agenti. Il sistema è quindi detto multiagente e tramite anche la interazione con altri agenti il suddetto agente dovrà trovare la via per risolvere i problemi per i quali è stato progettato (l'agente è definito quindi sociale) [6] [7].
Infine dovrà essere reattivo, cioè essere in grado di cambiare prontamente il proprio modo di agire elaborando nuove strategie in seguito a variazioni dell'ambiente circostante [6] [7].
Proprio per queste caratteristiche di generalità e similitudine a molti sistemi del mondo reale, i sistemi ad agenti multipli costituiscono un'interessante tipologia di modellazione di organizzazioni complesse, e hanno a questo riguardo vasti campi d'applicazione, che si estendono dall'IA fino alle scienze umane e sociali (economia, sociologia, etc.).

2.3 Applicazioni dei sistemi multiagente a problemi ambientali

2.3.1 Stato dell'arte

Durante gli ultimi anni i sistemi ad agenti hanno attratto un numero significativo di ricercatori che si occupano di informatica ambientale. L'approccio dei sistemi multiagente è stato adottato per sviluppare software ambientali per la gestione dei dati, il supporto alle decisioni o per svolgere simulazioni.
Lo sviluppo di tali programmi si è posto come evoluzione del paradigma ad oggetti e della computazione distribuita, utilizzando agenti come unità software base per sviluppare sistemi più complessi. I programmi che si occupano di informatica ambientale devono spesso fornire servizi differenti, come integrare informazioni provenienti da fonti eterogenee, maneggiare a dati variabili nel tempo e adattarsi a diverse condizioni mutevoli [8].
Inoltre, le applicazioni ambientali possiedono sia l'incertezza e la complessità insita nell'ambiente naturale.

I sistemi ad agenti sembrano essere particolarmente adatti per questo tipo di applicazioni, in quanto i sistemi naturali vedono infatti in larga parte l'azione di diversi attori, come città, dighe, sistemi idrici, fattorie, compagnie idroelettriche, che sono descrivibili mediante opportuni agenti. Inoltre l'interazione tra gli agenti è modellabile in modo realistico attraverso i protocolli di comunicazione sviluppati per gli algoritmi distribuiti usati in sistemi multiagente.
Nel prossimo paragrafo si descriveranno quindi diversi esempi di sistemi ad agenti sviluppati nell'ambito dell'informatica ambientale.

2.3.2 Tipologie di programmi di informatica ambientale

Esistono tre categorie di programmi che si occupano di problemi di informatica ambientale [8]:

- i programmi che si occupano di gestione delle informazioni,
- i programmi che supportano le decisioni in problemi ambientali,
- i programmi che simulano sistemi e processi ambientali o ecologici.

Nel resto di questo paragrafo verranno ora esposti alcuni esempi di programmi che sono stati sviluppati negli ultimi anni e che appartengono a queste categorie.

Per quanto riguarda i sistemi di gestione dell'informazione essi sono utilizzati per l'integrazione e la diffusione dei dati ambientali.
Un esempio di tali sistemi è EDEN-IW (Environment Data Exchange Network for Water Inland) [9] che mira a fornire ai cittadini, ricercatori e altri utenti i dati esistenti delle acque interne, come laghi o fiumi. Tale programma utilizza una tecnologia ad agenti che esegue la gestione dei dati, che estende quella implementata in un tool precedente, InfoSleuth [10], e interpreta le ricerche effettuate dagli utenti su una serie di distribuita di basi di dati eterogenei.
Un sistema molto simile che utilizza agenti software per accedere ai dati ambientali è NZDIS (New Zealand Distributed Information System). NZDIS è stato progettato per la gestione ambientale di meta-dati per ricerche su servizi eterogenei [11] [12].
Nella stessa categoria rientra il sistema BUSTER (Brema University Semantic Translator for Enhanced Retrieval), che utilizza ontologie per il recupero delle informazioni da fonti eterogenee e traduzione semantica nel formato desiderato [13].
Il sistema MAGIC (Sistema di diagnostica, acquisizione e gestione dei dati in sistemi complessi), che anche se non mirato solo per applicazioni ambientali, ha invece come obiettivo quello di sviluppare una flessibile architettura multiagente per la diagnosi di guasti nei sistemi complessi, adottando diversi metodi diagnostici in parallelo [14] [15].
Infine una applicazione simile, sviluppata dalla stesso team di sviluppo, è la architettura

distribuita DIAMOND utilizzato per controllo e diagnosi di sistemi naturali, che adotta agenti per il monitoraggio e la diagnosi distribuita [16].
Per quanto riguarda i sistemi di supporto alle decisioni essi sono utilizzati per studiare le decisioni ambientali e utilizzano la tecnologia ad agenti da una prospettiva di intelligenza artificiale distribuita.
Appartiene a questa categoria D-NEMO. Esso è un prototipo sperimentale e utilizza agenti per la gestione e il controllo dell'inquinamento atmosferico urbano. Gli agenti usati in D-NEMO incorporano azioni per attuare la classificazione e la categorizzazione dell'inquinamento atmosferico utilizzando reti neurali e la collaborazione tra agenti per la previsione dell'inquinamento dell'aria [17].
Un'altra applicazione è NED-2, sviluppata dall'Università della Georgia, che si occupa della simulazione di ecosistemi forestali e dei piani di gestione. In NED-2 gli agenti usano modelli di crescita e di rendimento per simulare i piani di gestione e generare report di risultati [18].
Ultimo, ma non meno importante, il sistema DAI-Depur, che è stato sviluppato presso l'Università di Catalogna, applica tecniche di intelligenza artificiale distribuita in un sistema di supporto decisionale per la supervisione di un impianto di trattamento delle acque di scarico. I processi dell'impianto sono rappresentati da agenti, che collaborano in un'architettura a strati [19] [20].
Infine l'ultima categoria di programmi sono i sistemi di simulazione ambientale.
Tali sistemi utilizzano agenti per la simulazione di sistemi ecologici o sociali.
Il sistema SHADOC appartiene a questa categoria e utilizza agenti per simulare il comportamento delle parti interessate e degli agricoltori coinvolti nell'irrigazione in una regione del Senegal [21].
Il sistema CATCHSCAPE è stato poi sviluppato come prosecuzione del precedente tool e si occupa dell'irrigazione del nord della Thailandia, utilizzando agenti per rappresentare tutti gli enti connessi al bacino idrologico [22] [23].
Altra applicazione che appartiene a questa categoria è Stau-Wien, che ha lo scopo di studiare la crescita urbana della città di Vienna e della sua periferia. L'obbiettivo di questo lavoro è quello di simulare i processi di trasformazione del paesaggio della regione suburbana nei dintorni della capitale austriaca. In questo tool gli agenti rappresentano le abitazioni e le industrie della città [24].
Infine il sistema multiagente GEMACE è utilizzato per studiare la gestione agricola e di caccia della Camargue e i suoi effetti. Esso simula le interazioni tra cacciatori, agricoltori e anatre che popolano questo habitat. Il sistema esamina le correlazioni tra le attività umane e l'ambiente e il loro impatto nell'utilizzo del territorio [25].

2.3.3 Conclusioni

Diventa evidente che la tecnologia ad agenti è tenuta in grande considerazione nel settore ecologico e ambientale per il supporto alle decisioni e per i sistemi di simulazione, e che la

programmazione ad agenti è ampiamente utilizzata per organizzare la tutela dei sistemi ambientali.
Il presente lavoro di tesi si propone quindi di progettare ed implementare un simulatore multiagente di sistemi idrici che possa essere completo, utilizzabile facilmente e affidabile.
Esso rientra quindi nella terza categoria di programmi descritta nel precedente paragrafo e verrà descritto nei prossimi capitoli.

3. Introduzione ai problemi multiagente: formalizzazione e risoluzione

3.1 Problemi CSP

Molti problemi nell'ambito dell'IA sono classificabili come problemi di soddisfacimento di vincoli (Constraint Satisfaction Problem o CSP); fra questi ad esempio vi sono problemi di allocazione di risorse, di pianificazione e di ragionamento temporale. Questi problemi possono essere risolti efficientemente attraverso tecniche ben note di risoluzione di CSP, che saranno descritte nei prossimi paragrafi.

Formalmente, un CSP può essere definito su un insieme finito di variabili X_1, X_2, ..., X_n i cui valori appartengono a domini finiti di definizione D_1, D_2, ..., D_n (esiste un dominio per ogni variabile) e su un insieme di vincoli.

Un vincolo su un insieme di variabili è una restrizione dei valori che le variabili possono assumere simultaneamente. Concettualmente, un vincolo può essere visto come un insieme che contiene tutti i valori che le variabili possono assumere contemporaneamente: un vincolo tra k variabili c(X_1, X_2, ..., X_k) è un sottoinsieme del prodotto cartesiano dei domini delle variabili coinvolte, cioè D_1, D_2, ...D_k, che specifica quali valori delle variabili sono compatibili con i valori delle altre. Questo insieme può essere rappresentato in molti modi, per esempio per mezzo di matrici, equazioni, disuguaglianze o relazioni.

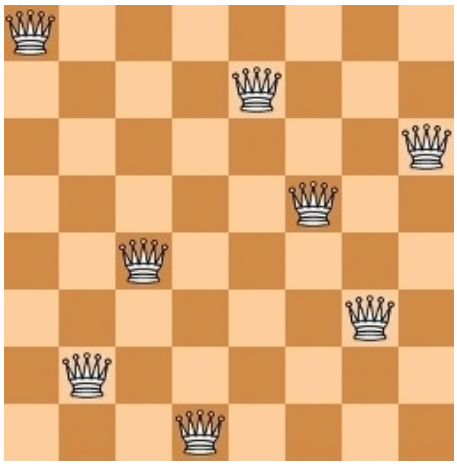

Fig. 3.1: Un esempio di disposizione delle 8 regine che rispetta i vincoli del problema

Un modo alternativo, ma equivalente, per definire un vincolo e rappresentarlo come una coppia <insieme variabili, relazione>, dove insieme variabili è una tupla di variabili che partecipano al vincolo e relazione è una relazione che definisce i valori che tali variabili possono assumere.

Una soluzione ad un CSP è un assegnamento di valori a tutte le variabili che soddisfi tutti i vincoli.

Il risultato finale di un modello CSP non consiste pertanto in una soluzione ottimale bensì in un sottoinsieme cartesiano dei valori che le variabili X_i possono assumere contemporaneamente.

Sono ora illustrati due esempi di problemi CSP in modo da chiarire come matematicamente vengono formalizzati questi tipi di problemi.

Un classico esempio di problema che può essere visto come CSP è il rompicapo delle otto

regine. Il problema consiste nel disporre su una scacchiera otto regine che non si attacchino a vicenda; naturalmente, non è possibile disporne più di otto, perché questo implicherebbe posizionarne due su una stessa riga. [26] [27]
Un esempio di corretto posizionamento è dato dalla Figura 3.1.
Per formalizzare il problema come CSP dobbiamo identificare un insieme di variabili, un insieme di domini ed un insieme di vincoli. Poiché su ogni riga e su ogni colonna della scacchiera dovrà essere posta esattamente una regina, una possibile formalizzazione del problema è quella di considerare come variabili ciascuna delle otto righe della scacchiera; l'insieme delle variabili sarà quindi $V = \{R_1, R_2, ...,R_8\}$.
Ciascuna di queste variabili può assumere un valore che rappresenta la colonna su cui si trova la regina corrispondente, per cui i domini delle variabili sono le otto possibili colonne: $D_1 = D_2 = \ldots = D_8 = \{1,2,3,4,5,6,7,8\}$.
Abbiamo quindi già inserito in questa formalizzazione il concetto che due regine non possono trovarsi sulla stessa riga; resta da imporre che due regine non possono trovarsi né sulla stessa colonna, né sulla stessa diagonale. L'insieme dei vincoli sarà dunque:

$\forall$ i, j $R_i \neq R_j$

ad indicare che due regine non possono trovarsi sulla stessa colonna e

$\forall$ i, j se $R_i = a$ e $R_j = b$ allora $i - j \neq a - b$ e $i - j \neq b - a$

per imporre che due regine non possono trovarsi sulla stessa diagonale.

Un secondo esempio è basato sul famoso problema del cosiddetto "map coloring".
Data una mappa con regioni confinanti il problema si propone di colorare ogni regione confinante con colori diversi, dato un dominio finito di tonalità.
Un esempio può essere il seguente [3]:

Data la mappa dell'Australia colorare ogni regione in rosso, verde o blu in modo che le regioni confinanti abbiano colori differenti.

La Fig. 3.2 fornisce una rappresentazione grafica del problema proposto.

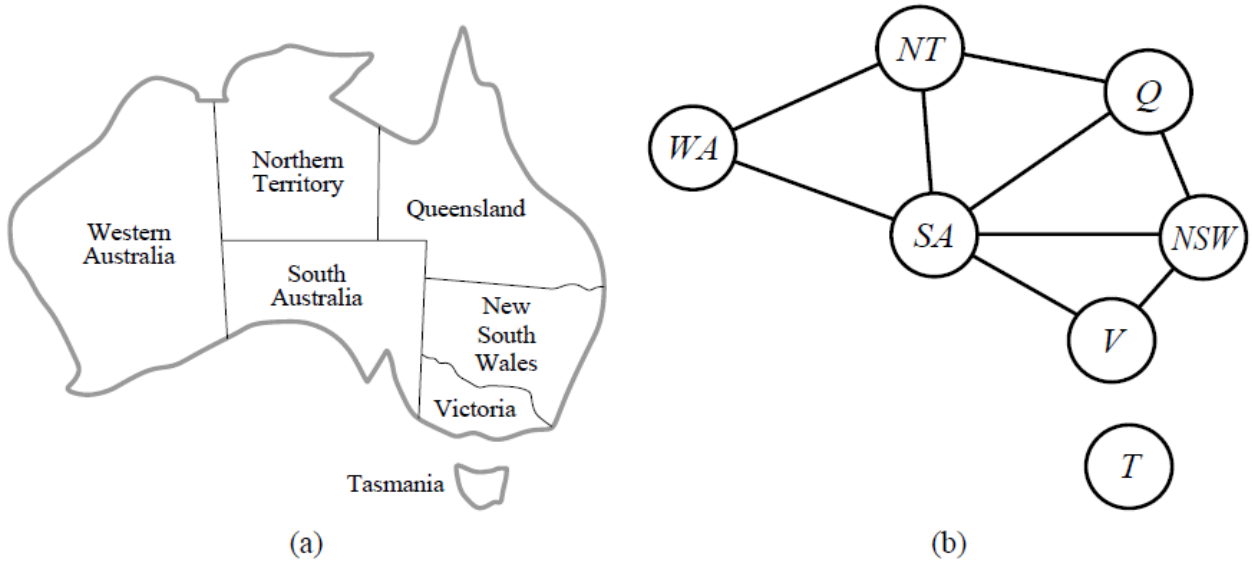

Fig 3.2: (a) I principali stati e territori dell'Australia. Il problema di colorazione dei vari stati può essere impostato come problema CSP. L'obbiettivo è di colorare ogni regione in modo che gli stati confinanti non abbiano lo stesso colore (b) Il problema rappresentato con un grafo dei vincoli

Il problema può cosi essere formalizzato:

Variabili: X = {WA,NT,Q,NSW, V, SA, T}
Dominio: D = {red, green, blue} attribuibile ad ogni variabile
Vincoli: C = {SA ≠ WA, SA ≠ NT, SA ≠ Q, SA ≠ V, SA ≠ SW, WA ≠ NT, . . .}

Questo problema rappresenta in realtà una classe più ristretta di problemi CSP, i problemi binari.
Un problema CSP binario è un problema dove tutti i vincoli sono binari o unari (cioè coinvolgono rispettivamente due variabile o una singola variabile), e ogni vincolo non binario può essere convertito in un vincolo binario aggiungendo variabili accessorie al problema. Un CSP binario può essere rappresentato da un grafo dei vincoli, che ha un nodo per ogni variabile e un arco tra due nodi se e solo se esiste un vincolo che coinvolge le due variabili. I vincoli unari sono auto-anelli intorno ai nodi interessati.
In Figura 3.2(b) è rappresentato il grafo dei vincoli relativo all'esempio precedentemente esposto.

3.2 Risoluzione dei problemi CSP

In questo paragrafo si introdurranno tecniche e algoritmi utilizzati per risolvere i problemi CSP. Si daranno solamente alcuni cenni in modo da introdurre il lettore alle tecniche base

per risolvere tali tipi di problemi.
Nei prossimi paragrafi e nel prossimo capitolo si descriveranno invece con maggiore dovizia di particolari le tecniche per risolvere i problemi DCSP e DCOP, che caratterizzeranno la parte fondamentale di questo lavoro di tesi.
Per risolvere problemi di soddisfacimento di vincoli sono utilizzati diversi algoritmi, di cui vengono in seguito citati solo alcuni esempi, principalmente per scopo bibliografico:

- Backtracking
- Backjumping
- Backmarking
- Forward Checking
- Arc-Consistenza
- Path-Consistenza

Nel seguito di questo paragrafo verrà approfondito solo l'algoritmo di backtracking, dato che esso sarà poi ripreso nei prossimi capitoli per quanto riguarda i problemi DCSP.
Il **backtracking** (in italiano, *ritorno all'indietro*) è una tecnica per trovare soluzioni a problemi in cui devono essere soddisfatti dei vincoli. Questa tecnica enumera tutte le possibili soluzioni e scarta quelle che non soddisfano i vincoli.
Una tecnica classica consiste nell'esplorazione di strutture ad albero e nel tenere traccia di tutti i nodi e i rami visitati in precedenza, in modo da poter tornare indietro al più vicino nodo che conteneva un cammino ancora inesplorato nel caso la ricerca nel ramo attuale non abbia successo. I nodi a profondità uguale rappresentano i possibili valori di una variabile.
Una applicazione comune del backtracking riguarda i programmi simulatori per il gioco degli scacchi, che generano tutte le mosse possibili per una profondità di N mosse a partire da quella attuale e poi esaminano con il backtracking le varie alternative, selezionando alla fine quella migliore. Il backtracking ha una complessità esponenziale, ma in generale l'algoritmo integra euristiche che permettono di diminuirne la complessità. Non sarà esposta in questa trattazione la implementazione del backtracking, dato che esso non sarà un algoritmo utilizzato direttamente nel lavoro di tesi svolto.

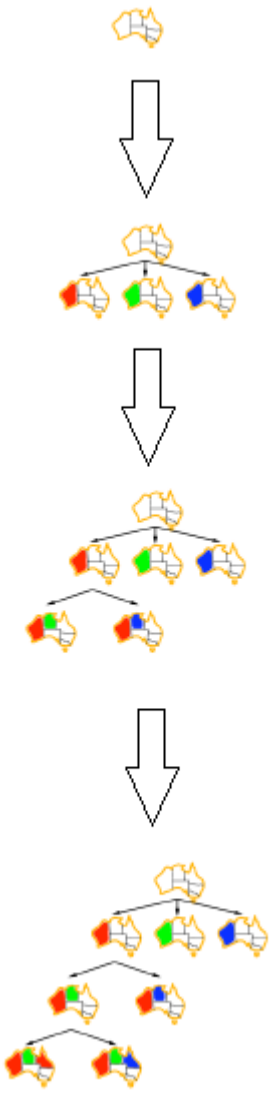

Fig. 3.3: Esempio di esecuzione dell'algoritmo di backtracking nel problema di map coloring. L'albero rappresentato è parziale

I brevi cenni che sono stati usati servono piuttosto a rendere l'idea di come operi, in modo da comprendere più avanti come invece funzioneranno gli algoritmi distribuiti che si rifanno al backtracking. Per maggiore chiarezza è stato riportato un esempio di esecuzione dell'algoritmo nella risoluzione del problema di map coloring precedentemente descritto (Figura 3.3).
Come si vede dalla sequenza, inizialmente si generano i tre rami derivanti dalla scelta dei diversi colori per la prima regione. Successivamente si esplora solo il primo dei tre rami, scegliendo per la regione confinante le due possibilità. L'albero prosegue in questo modo, sviluppando in profondità un ramo alla volta. Questa è quella che viene definita depth-first search.
Nel caso ci siano inconsistenze si percorre a ritroso l'albero di un passo, cercando di annullare gli assegnamenti di variabili che risultano incompatibili a causa dei vincoli del problema.

3.3 Problemi DCSP

3.3.1 Formalizzazione dei problemi DCSP

Ora affrontiamo una classe di problemi che estende la classe precedentemente presentata. Essi vengono definiti DCSP e presentano affinità con i problemi CSP, con la caratteristica di lavorare però in ambiente distribuito.
L'idea principale è quella di implementare un problema composto di vincoli e variabili, esattamente come nel caso CSP, ma facendo si che il problema non sia risolto in modo centralizzato, cioè da un risolutore esterno che controlla che tutti i vincoli siano soddisfatti, ma in modo distribuito, cioè mediante un lavoro autonomo degli agenti, che comunicano tra loro i vari risultati parziali.
Ogni variabile è controllata da un agente che assegna un valore ad essa. Ogni agente controlla una sola variabile.
Per attuare la risoluzione descritta gli agenti devono operare in coerenza, cioè la soluzione da loro trovata deve essere misurata e valutata in termini di efficienza riguardo le risorse usate, qualità, cioè devono rispettare tutti i vincoli, e coordinamento, cioè sincronizzando e allineando tutte le attività da loro compiute.

3.3.2 Risoluzione dei problemi DCSP: Asynchronous Backtracking

L'algoritmo risolutivo più diffuso per risolvere i problemi DCSP è l'Asynchronous Backtracking (ABT).
Esso riprende le fondamenta mostrate dal backtracking utilizzato per gli algoritmi CSP, estendendolo in modo che possa lavorare in ambiente distribuito.
In questo paragrafo vengono innanzi tutto descritta la logica dell'algoritmo ABT e in seguito viene illustrato un esempio per permetterne una migliore comprensione.

Nei problemi CSP esiste l'algoritmo di backtracking. Esso cerca di assegnare valori a una variabile e ricorsivamente cerca di assegnare nuovi valori a nuove variabili cercando vi non violare i vincoli.
Come già detto tale algoritmo lavora in modo centralizzato.
L'ABT è la sua evoluzione per problemi distribuiti, e consente la comunicazione tra i vari agenti utilizzando messaggi.
L'algoritmo ABT prevede poi alcuni miglioramenti in efficenza rispetto alla classica depth-first search implementata dal backtracking, ottenuti in gran parte mediante un opportuno ordinamento delle variabili. In generale vengono assegnati innanzi tutto i valori alle variabili coinvolte in più vincoli. Questa euristica funzione infatti in modo molto efficiente con risoluzione ABT.
Ogni agente è responsabile di una sola variabile ed è presente un ordinamento tra gli agenti stessi. Nel corso dell'esecuzione dell'algoritmo ogni agente può cambiare il valore della propria variabile.
Ogni agente ha poi una visione locale dell'assegnamento delle altre variabili, che si aggiorna nel corso dell'esecuzione. Inoltre inizialmente ogni agente ha dei vicini, cioè altri agenti di cui conosce l'esistenza.
Questo rappresenta un punto cruciale.
All'inizio dell'esecuzione infatti ogni agente conosce l'esistenza solo degli agenti che sono coinvolti in vincoli con lui. Questa lista può però crescere, quando un agente scopre l'esistenza di altri agenti, che sono coinvolti nel problema ma non direttamente coinvolti in vincoli con l'agente in questione. In questo modo ogni agente durante l'esecuzione possiede una vista locale degli altri agenti che cresce, si aggiorna e permette localmente di scegliere il valore più adatto per la propria variabile.
Ci sono tre tipi di messaggi che sono scambiati dai vari attori in questo algoritmo:

- ok?(j; x_j): tale messaggio è inviato dall'agente che controlla x. j è il nome di un altro agente e x_j il valore corrente della variabile. Questo messaggio chiede al ricevente se l'assegnamento viola alcuni dei suoi vincoli. Tale messaggio viene inviato dagli agenti a priorità più alta a quelli con priorità più bassa.
- nogood(j; nogood): tale messaggio è inviato dall'agente che controlla la variabile j per segnalare che non può assegnare un valore coerente alla variabile a causa degli assegnamenti già in vigore e di vincoli che non lo consentono. Al contrario del precedente tale messaggio viene inviato dagli agenti a priorità più bassa a quelli con priorità più alta.
- add-neighbor(j): viene inviato ad un agente chiedendogli di aggiungere j come vicino, cioè inserirlo nella sua vista locale e tenere conto dei vincoli che la riguardano.

Anche di questo algoritmo non viene fornito direttamente il codice, ma un chiaro esempio

che chiarisce come esso venga applicato per risolvere problemi DCSP. Ovviamente non essendo possibile su carta mostrare un esempio di esecuzione distribuita asincrona se ne mostrerà una versione semplificata, che però rende bene l'idea riguardo il suo funzionamento.

Si consideri il seguente problema di soddisfacimento di vincoli distribuiti:
variabili:

- X_1 di dominio D_1 = { a, b, c }
- X_2 di dominio D_2 = { e, f }

vincoli (combinazioni permesse):

- $p_1(X_1,X_2)$ = { <c,e>, <c,f> }

Inoltra l'agente A_1, che controlla X_1, ha priorità più alta dell'agente A_2, che controlla X_2.

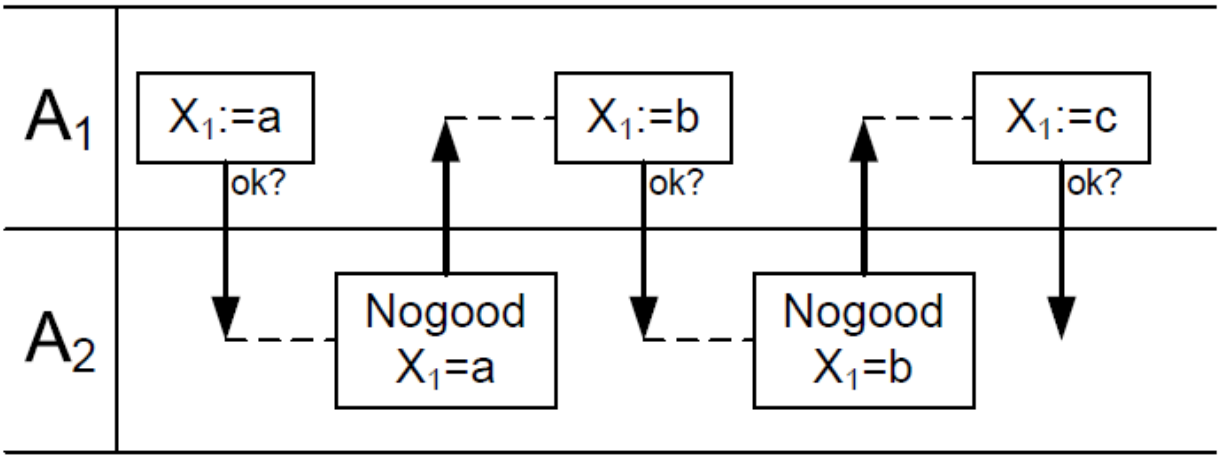

Fig. 3.4: Esempio di risoluzione del problema proposto mediante l'algoritmo ABT.

Una possibile esecuzione di ABT è rappresentata nella Figura 3.4.
Come si vede inizialmente A_1 seleziona per la propria variabile il valore 'a', ma A_2, mediante il messaggio di nogood segnala come questo valore non consenta di assegnare alcun valore a X_2. In questo modo A_1 assegna il valore 'b', che però risulta essere nuovamente non adeguato. Infine A_1 opta per il valore 'c', consentendo ad A_2 di selezionare 'e' o 'f' indifferentemente e trovare così una soluzione valida al problema.

3.4 Ottimizzazione dei vincoli e problemi DCOP

L'ultimo passo che va affrontato prima di esporre il lavoro effettuato è la descrizione dei problemi DCOP.
Essi rappresentano una evoluzione dei problemi DCSP e compongono l'intera parte del lavoro presentato in questa tesi.
Prima però di descrivere in dettaglio i problemi DCOP bisogna affrontare il problema generale di ottimizzazione dei vincoli.

3.4.1 Ottimizzazione dei vincoli

L'ottimizzazione dei vincoli si pone l'obbietivo di cercare non più solo una soluzione che rispetti i vincoli, ma la soluzione migliore, che cioè massimizzi una funzione di utilità (o in alternativa minimizzi una funzione di costo).
In questo modo un problema di ottmizzazione dei vincoli è esattamente come un normale problema CSP, dove i vincoli sono però pesati e l'obbiettivo è trovare la soluzione che massimizzi i pesi dei vincoli soddisfatti.
Alternativamente, un problema di ottimizzazione di vincoli può essere rappresentato in modo duale, cioè come un normale problema CSP, dove però i pesi dei vincoli sono visti come costi e l'obbiettivo diventa quindi minimizzare la somma dei pesi dei vincoli violati.
Come si sarà intuito la soluzione che si cerca di ottenere è quindi quella ottimale, cioè si sommano i pesi dei vincoli soddisfatti (violati) e si cerca di massimizzare (minimizzare) tale somma.
Esistono due diversi tipi di vincoli in questi tipi di problemi, detti vincoli hard e vincoli soft.
I primi devono essere soddisfatti per trovare una soluzione accettabile, i secondi invece esprimono solo preferenze verso certi tipi di soluzioni.

3.4.2 Problemi DCOP

Un Distributed Constraint Optimization Problem (DCOP or DisCOP) è la naturale evoluzione della categoria di problemi descritta nel precedente paragrafo. Un DCOP è la versione distribuita dei problemi di ottimizzazione dei vincoli.
In questo tipo di problemi ci sono agenti che, in modo indipendente e comunicando tra loro tramite messaggi, selezionano gli assegnamenti ottimali per le variabili a cui sono assegnati, in modo da sommare i pesi dei vincoli soddisfatti (violati) e cercare di massimizzare (minimizzare) tale somma.
I DCOP sono quindi la fusione di due categorie di problemi, i DCSP e i problemi di ottimizzazione.
Dai primi riprendono la formulazione, che quindi prevede diversi agenti che interagiscono tramite messaggi e che scelgono in modo autonomo il valore da assegnare alle variabili, variandolo eventualmente in base ai messaggi ricevuti. Dai secondi riprendono l'ottimizzazione, pesando dunque i vincoli è dando loro un significato di costo o utilità.

Formalmente questa è la definizione di un problema DCOP:
Un DCOP può essere definito come una tupla <A, V, D, a, F_i, n>, dove:

- *A* è un insieme di agenti;
- *V* è un insieme di variabili {v_1,v_2, ..., v_n};
- *D* è un insieme di domini {D_1,D_2, ..., D_n}. Ogni dominio dell'insieme è associato ad una sola variabile e sono tutti finiti;
- *a* è una funzione a: V → A, che quindi mappa le variabili al loro agente associato. Quindi a(v_i) = a_j implica che a_j è responsabile di assegnare la variabile v_i;
- F_i è una funzione che mappa ogni possibile assegnamento di variabili ad un costo. Essa è definita in questo modo:
  F_i: Asseg → N, dove *Asseg* è l'insieme di tutti i possibili assegnamenti, mentre *N* è l'insieme dei numeri naturali;
- *n* è la funzione che semplicemente aggrega tutti i costi parziali mediante una sommatoria.

3.4.3 Risoluzione dei problemi DCOP e concetti preliminari

Esistono diversi tipi di algoritmi con peculiarità differenti che possono risolvere un problema DCOP.
Per svolgere questo lavoro di tesi sono stati utilizzati 4 di essi, DPOP, ASO-DPOP, SynchBB e MGM, che verranno descritti nei prossimi paragrafi.
Prima di affrontare questa trattazione è però importante affrontare alcuni concetti preliminari fondamentali per capire il funzionamento degli algoritmi di seguito descritti.
Innanzi tutto una via possibile per risolvere un problema di ottimizzazione dei vincoli è la realizzazione del grafo dei vincoli. Esso è utilizzato da 3 algoritmi tra quelli selezionati, in particolare DPOP, ASO-DPOP e SynchBB.
Tale grafo è non diretto ed è una rappresentazione grafica formale del problema da risolvere.
Ogni agente (che come sappiamo controlla una sola variabile) è rappresentato da un nodo del grafo. Ogni nodo poi è collegato con un arco ad un altro nodo se e solo se esiste un vincolo che coinvolge le variabili controllate dagli agenti. I vincoli sono dunque rappresentati dagli archi.
Ottenuto questo grafico da esso si può ricavare l'albero DFS (Depth First Search tree).
Esso viene ottenuto utilizzando l'algoritmo DFS. In questa sede non è necessario avere una formale descrizione dell'algoritmo ma è però importante conoscere il principio su cui esso opera.
A livello descrittivo si può affermare che il meccanismo di funzionamento del DFS prevede dopo la visita di un nodo di scendere in profondità il più possibile, visitando i nodi figli fino a raggiungere le foglie. A questo punto l'algoritmo DFS procede a ritroso riproponendo

questa strategia.
Operativamente quindi la costruzione di tale albero procede in questo modo [28]:

- Dato un grafo G, tramite una visita DFS si determina un albero T, detto albero di copertura DFS.
- La costruzione dell'albero può essere effettuata durante la visita del grafo, aggiungendo all'albero T nodi del grafo visitati e archi che congiungono un nodo visitato con uno non ancora visitato. Il nodo da cui comincerà la visita sarà eletto nodo radice dell'albero di copertura.

Tale albero è diretto (i suoi archi hanno cioè una direzione) e può poi essere decorato con ulteriori archi, i back edge e i cross edge.
Nel dettaglio gli archi sono quindi divisibili in tre categorie:

- I forward edge, che sono cioè quelli che sono già stati rappresentati nell'albero DFS.
- I back edge, che sono aggiunti per indicare una dipendenza che era presente nel grafo dei vincoli ma che è stata persa nel all'albero DFS. Essi collegano solo nodi tra i quali esiste una relazione di tipo parentale.
- I cross edge, che hanno la stessa funzione dei back edge ma collegano nodi tra cui non vige alcun tipo di relazione parentale.

Definiti ora questi concetti preliminari possiamo passare all'analisi degli algoritmi utilizzati.
Di essi non verrà fornito direttamente lo pseudo-codice ma verrà invece descritto il principio di funzionamento, in modo da rendere comprensibile la loro logica e potere così valutare adeguatamente i risultati sperimentali.

3.4.4 DPOP

L'algoritmo DPOP [29] si articola in tre fasi.
La prima effettua le procedure per la costruzione dell'albero DFS e la decorazione dello stesso mediante i due tipi di archi aggiuntivi descritti nel precedente paragrafo.
La seconda fase procede all'invio dei messaggi di utilità, detti messaggi UTIL.
La sua elaborazione inizia dalle foglie dell'albero e procede fino alla radice.
Il messaggio UTIL è inviato da un nodo che può essere una foglia o un nodo qualsiasi dell'albero. Esso rappresenta una valutazione della utilità ottenuta dal nodo foglia con l'attuale assegnamento di variabili, o, nel caso in cui il nodo non sia una foglia, esso rappresenta l'utilità dell'intero sotto albero presieduto dal nodo di invio (semplicemente sommando le utilità dei nodi figli).
L'algoritmo in questa fase lavora sull'albero costruito nella prima fase e quindi nel caso più semplice un messaggio UTIL inviato da un nodo al suo genitore dipende solo dal sotto-

albero al rispettivo nodo, cioè l'albero di cui il nodo stesso è radice, e dal il vincolo tra il nodo e il suo genitore.
Questo potrebbe però non valere in generale. Nel caso infatti di esistenza di back edge un messaggio inviato da un nodo al suo genitore può dipendere anche da variabili che sono posizionati sopra il genitore stesso.
Lo scambio dei messaggi UTIL diventa quindi dipendente da più nodi e quindi da più variabili controllate da agenti differenti.
Proprio in questo secondo caso entra in gioco l'approccio dell'algoritmo DPOP, che può assemblare messaggi come un ipercubo con più dimensioni, dato che il messaggio dipenderà da più nodi (e quindi da più variabili).
DPOP prevede quindi messaggi di dimensione varibile (una per la variabile di destinazione, e una per ciascuna variabile di contesto) differenziandosi ad esempio da algoritmi come DTREE, dove i messaggi di utilità hanno una sola dimensione (sono lineari nella dimensione del dominio variabile di destinazione) [30].
I messaggi devono quindi essere combinati inizialmente, con la conseguenza di un aumento delle loro dimensioni. Come detto in precedenza questo è dovuto alla presenza di back edge.
Quando però tale messaggio multidimensionale raggiunge il nodo a cui un back edge punta, allora il messagio UTIL che esso emetterà sarà di dimesione uno.
Considerando che la radice dell'albero non può avere back edge (dato che questo sarebbe un non senso), si vede chiaramente come l'algoritmo riesca a limitare le dimensione dei messagi scambiati durante l'esecuzione.
DPOP è quindi strutturato per poter effettuare questa diminuzione nella dimensione dei messaggi e passare cosi alla terza e ultima fase dell'algotimo, detta propazione dei messaggi VALUE.
Questa fase non fa altro che inviare messaggi che memorizzano i valori assegnati alle variabili dalle foglie fino alla radice dell'albero, in modo da poter ricostruire la soluzione finale trovata.
La complessità spaziale di questo algoritmo è lineare nel numero di messaggi: ci sono infatti n-1 messaggi UTIL (uno per ogni arco dell'albero), dove n è il numero di nodi del problema, e O(m) messaggi VALUE, dove m è il numero di archi.
Considerando che sono i messaggi UTIL ad avere la caratteristica della multidimesionalità, possiamo concludere con certezza che la complessità risiede decisamente in essi, che però non possono avere dimensione superiore ad $O(2^p)$, dove p è la profondità dell'albero ottenuto nella prima fase dell'algoritmo [31].

3.4.5 ASO-DPOP

Anche l'algoritmo ASO-DPOP si articola in tre fasi [32] ed è basato sull'invio di messaggi, che possono essere di vari tipi.
La prima categoria di messaggi è l'ASK message, che se inviato ad un agente presuppone la

risposta da parte di quest'ultimo con l'assegnamento ottimale per la propria variabile, in base alla propria utilità.
Per rispondere alla richiesta viene utilizzato un GOOD message, che è una tupla *g* formata da tre valori, *s*, *u*, *b* dove il primo valore è l'assegnamento, il secondo è l'utilità dell'agente che invia il messaggio con tale assegnamento e *b* è una variabile booleana. Quando b = true, *g* è detto *true good*, cioè si basa su informazioni complete, mentre nel caso duale, in cui b = false allora *g* è un *false good*, cioè sarà usato per aggregare informazioni parziali.
Questo significa che nel caso un agente abbia informazioni solo parziali esso invia al richiedente un messaggio false good ed invia successivamente messaggi ASK ai suoi nodi figli per ottenere informazioni più complete.
L'obbiettivo dell'algoritmo è quindi quello di aggregare sempre più informazioni complete per ottenere il miglior assegnamento possibile per le varibile, in termini di numero di vincoli violati e utilità degli agenti.
Inoltre nel corso dell'esecuzione dell'algortimo viene sempre tenuta traccia di una soglia di utilità globale (che è calcolata aggregando le singole utilità parziali degli agenti) che è stata trovata fino a quel momento dell'esecuzione.
Questo consente di ignorare possibili assegnamenti che produrrebbero utilità inferiore alla soglia fino a quel momento trovata.
Ogni agente deve poter determinare quale sia il prossimo assegnamento che potrebbe generare una migliore utilità, e per fare ciò deve ricordare quale sia l'assegnamento che ha inviato al suo nodo padre. Per fare ciò esso mantiene una struttura dati detta $sentGood_i$, che memorizza tutti i true good che fino a quel momento ha inviato.
Data questa struttura e definendo Ass_i come l'insieme di tutti i possibili assegnamenti, ogni agente determina s_{max}, che è un elemento appartente alla differenza insiemistica tra Ass_i e $sentGood_i$ e come il valore che massimizza l'utilita del singolo agente.
A questo punto si può descrivere l'algorimo nel dettaglio.
Anche in questo caso esistono tre fasi.
La prima è di inizializzazione, in cui viene costruito l'albero DFS decorato.
La seconda fase parte dalle foglie, dato che inizialmente il nodo radice inizia a mandare messaggi ASK fino alle foglie. Quando quest'ultime ricevono tali messaggi, essi determinano il proprio s_{max} e, non avendo nodi figli, possono facilmente determinare la propria utilità. L'utilità è infatti calcolata utilizzando il valore s_{max}. In questo modo può rispondere con un messagio true good e memorizzare nella propria struttura $sentGood_i$ di aver inviato tale messaggio.
Se il messaggio ASK è ricevuto da un nodo che non è una foglia, dopo aver calcolato il proprio s_{max}, calcola ed invia un messagio true good se le informazioni sono complete. In caso contrario invierà un false good e un messaggio di ASK ai nodi figli per ottenere più informazioni.
La seconda fase dell'algoritmo viene svolta con diverse ripetizioni, fino al raggiungimento della convergenza, al termine del quale prende atto la terza fase, detta la propagazione dei

valori.
Similmente all'algoritmo DPOP questa fase permette agli agenti di scambiare messaggi che memorizzano i valori assegnati alle variabili dalle foglie fino alla radice dell'albero, in modo da poter ricostruire la soluzione finale trovata.
L'algoritmo ASO-DPOP termina quando i domini delle variabili in gioco sono finiti ed è inoltre in grado di trovare la soluzione ottima.

3.4.6 SynchBB

L'algoritmo SynchBB [33] simula il funzionamento del metodo “Branch and Bound” utilizzato per problemi CSP, applicandolo però in ambito distribuito e con formalizzazione di tipo DCOP.
Nell'algoritmo SynchBB l'ordinamento delle variabili e dei valori sono fissati in anticipo, e i *path*, cioè assegnamenti parziali per tutte le variabili, vengono scambiati tra gli agenti.
Un elemento del path consiste in una variabile, un valore per la variabile e il numero di vincoli violati dal valore assegnato alla variabile stessa.
La valutazione del path, che definiamo *v*, è un valore che rappresenta il numero massimo di violazioni di vincoli rapportato al numero di variabili nel percorso stesso.
Infine viene fissato un limite superiore che è valutato durante l'esecuzione dell'algoritmo ed è il valore minimo di *v* che è stato trovato fino a quel momento nel corso dell'esecuzione dell'algoritmo.
L'idea è quella di utilizzare i path in modo da generare poi il percorso completo, con l'assegnamento finale ed ottimo di tutte le variabili.
Per ottenere questo risultato l'algoritmo procede in sequenza:

- Si costruisce l'albero DFS.
- Il primo agente dell'ordinamento inizializza l'algoritmo inviando un path al secondo agente. Tale path contiene solo il valore assegnato alla sua variabile.
- Quando un agente riceve un percorso dall'agente che lo precede nell'ordinamento, valuta il path ricevuto e assegna come valore della sua variabile il primo valore presente nel suo dominio, che è stato ordinato in fase di inizializzazione dell'algoritmo. A questo punto l'agente invia all'agente successivo nell'ordinamento il nuovo percorso, formato da quello ricevuto con l'aggiunta del proprio valore scelto. Questo nel caso la sua valutazione sia minore del limite superiore *v* corrente, mentre in caso contrario continua a scegliere valori successivi nel suo domino fino a che non ne trova uno tale per cui la valutazione del valore sia minore di *v*. Se tale valore non esiste semplicemente invia il path ricevuto all'agente che lo precede nell'ordinamento.

SynchBB è un algoritmo completo (cioè è in grado di trovare una soluzione ammissibile se essa esiste), dato che esso riprende il metodo Branch and Bound applicandolo in ambito

distribuito ed ereditando quindi questa proprietà dall'algoritmo stesso [34]. Possiede però lo svantaggio di non essere in grado di permettere assegnamenti o cambiamenti di valori agli agenti in parallelo, perdendo così molto in prestazioni.

3.4.7 MGM

Prima di introdurre l'algoritmo MGM è importante fornire alcuni cenni preliminari [35].
Non entreremo nel dettaglio del funzionamento dell'algoritmo, ma verranno descritte le basi di teoria dei giochi che sono fondamentali per comprendere l'approccio di questo algoritmo.
MGM si basa su una struttura non gerarchica delle variabili evitando di costruire relazioni di parentela tra nodi (e quindi agenti e variabili).
Non è dunque un algoritmo che si basa su un albero.
Di conseguenza la sua concezione prevede di impostare un gioco dove le variabili a cui assegnare un valore sono i giocatori e le azioni corrispondono alla scelta dei valori da assegnare alle variabili.
Il problema viene quindi formalizzato come un insieme di vincoli che coinvolgono più giocatori che devono cercare un assegnamento delle variabili tali per cui le loro funzioni di utilità raggiungano il valore più alto possibile.
A questo punto si può introdurre il concetto di equilibrio di Nash, un insieme di strategie dalla quale nessun giocatore ha incentivo a deviare finché restano immutate le strategie di tutti gli altri giocatori. In altre parole, la strategia di ogni giocatore è la miglior risposta alle strategie giocate dagli altri [6].
Prima di trattare l'algoritmo è fondamentale impostare formalmente il gioco da risolvere. Interpretando ogni variabile come il nodo di un grafo, *E* è una matrice di adiacenza che assume valore 1 in posizione (i,j) se la variabile x_i è coinvolta in almeno un vincolo con la variabile x_j, 0 altrimenti.
Si definisce poi X_i come il dominio della variabile x_i, mentre *X* sarà il prodotto cartesiano di tutti gli X_i, quindi il prodotto cartesiano di tutti i domini.
Questo problema di ottimizzazione dei vincoli è completamente caratterizzato da (X, E, U), dove *U* è l'insieme di tutte le funzioni di utilità
E' stato provato [35] che l'assegnamento che ottimizza un problema DCOP caratterizzato da (X, E, U) è un equilibrio di Nash rispetto al gioco (X,E,u), dove in questo caso *u* è l'insieme di tutte le funzioni di utilità locali.
Ottimizzando inoltre su un insieme finito di valori (quindi con domini finiti) l'algoritmo garantisce di trovare un massimo, che corrisponde ad un equilibrio di Nash, garantendone così l'esistenza.
In un ambiente puramente egoistico gli agenti cercherebbero solo di ottimizzare la loro utilità locale, portando però il sistema ad essere instabile [36].
Imponenedo invece un struttura in cui essi devono tener conto dell'intero ambiente e di una utilità non solo locale allora ciò porta a stabilità e convergenza nella soluzione [37].

L'algoritrmo MGM modella quindi un ambiente distribuito tramite lo scambio di messaggi tra i vari agenti/giocatori che cercano di ottimizzare la propria utilità locale.
E' stato provato [35] che data una condizione iniziale con una utilità globale minima accettabile, MGM garantisce di trovarla con tempi inferiori agli algortimi descritti in precedenza.

4. Formalizzazione e risoluzione del problema DCOP

4.1 Tipologie degli agenti e elaborazione dei vincoli

4.1.1 Descrizione degli agenti

Innanzi tutto nel formalizzare il problema bisogna identificare i tipi di agenti in gioco e i vincoli che li caratterizzano.
I tipi di agenti in gioco sono essenzialmente due, le città e le dighe.
Le città rappresentano gli agenti che prelevano acqua dal corso del fiume. Noi li chiameremo per comodità città, ma in realtà questo tipo di agente può rappresentare entità come anche fabbriche o sistemi di irrigazione, che quindi hanno necessità di prelevare dei volumi di acqua dal corso principale per loro necessità.
Il secondo tipo di agente è rappresentato dalle dighe.
La diga è un tipo particolare di agente, che si occupa di produrre energia idroelettrica turbinando l'acqua rilasciata dal serbatoio (o lago regolato) che interrompe il naturale del fiume. La diga può quindi incanalare nel corso principale una certa quantità di acqua che può anche prelevare dei volumi dalla sua riserva personale (il serbatoio), che contiene un quantitativo di acqua fissato.
Di conseguenza dopo la diga il corso d'acqua prosegue portando solo la portata rilasciata dalla diga.
L'esempio di Figura 4.1 illustra chiaramente i precedenti concetti.
Vediamo di spiegare tale esempio in modo chiaro.
Innanzi tutto vediamo il parametro Q_1, che rappresenta la portata principale del corso d'acqua.
Il primo agente che entra in gioco è una città, che gestisce la variabile X_1.

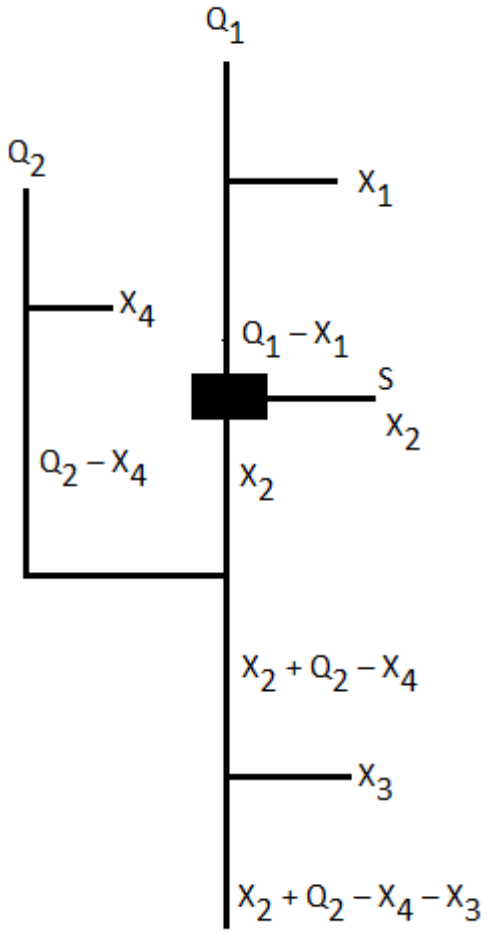

Fig. 4.1: Esempio di rappresentazione grafica di un possibile scenario idrico

L'agente preleva quindi una certa quantità d'acqua, secondo le proprie esigenze e di conseguenza l'acqua che potrà scorrere successivamente sarà $Q_1 – X_1$.
Il successivo agente è una diga. Esso, rappresentando una diga, ha già una riserva d'acqua pari ad S, che il parametro che rappresenta tale riserva. L'acqua che poi la diga rilascia è X_2.
L'ultimo agente che poi è illustrato sul corso d'acqua principale è X_3, un'altra città, ma prima di esso come si vede c'è un affluente che sfocia nel corso d'acqua principale.
Come si vede infatti c'è un altro corso d'acqua con portata Q_2 da cui l'agente X_4, una città, attinge una quantità d'acqua pari a X_4 e quindi l'acqua che si riversa nel corso d'acqua principale è $Q_2 – X_4$. Difatti come si vede successivamente nel corso d'acqua la portata totale è $X_2 + Q_2 – X_4$.

4.1.2 Formalizzazione dei vincoli

A questo punto è necessario formalizzare i vincoli da associare ad ogni agente.
Per fare ciò verrà utilizzato l'esempio precedentemente illustrato in Figura 4.1, che verrà quindi formalizzato e descritto. Inoltre i vincoli saranno suddivisi in due categorie trasversali.
La prima definisce i vincoli di due tipi, vincoli hard e vincoli soft, il cui significato matematico è stato descritto nel capitolo precedente.
La seconda categoria suddivide i vincoli in base al loro significato, dividendoli in vincoli fisici, normativi e di utilità.
Un vincolo quindi può essere contemporaneamente soft (o hard) e fisico, o soft (o hard) e normativo ad esempio.
E' importante sottolineare questa suddivisione dato che essa sarà fondamentale per comprendere la formalizzazione del problema.

<u>Vincoli fisici</u>
I vincoli fisici sono quelli che il sistema deve rispettare per poter lavorare in modo sensato, cioè quelli legati a proprietà fisiche del sistema. Ad esempio un agente non può prelevare dal corso d'acqua più di quella presente nel sistema.
Nell'esempio di Figura 4.1 questi sono i vincoli:

$X_1 \leq Q_1$
$X_4 \leq Q_2$
$X_2 \leq S + Q_1 – X_1$
$X_3 \leq X_2 + Q_2 – X_4$

Come si vede i primi due vincoli indicano che l'acqua prelevate dalle città deve essere ovviamente minore della portata d'acqua disponibile. Il terzo vincolo esprime che invece la diga può rilasciare una quantità di acqua che è minore della somma della portata di acqua disponibile e della riserva S che è propria della diga.

Infine l'ultimo vicolo è nuovamente legato all'ultimo agente città, X_3, che può prelevare una quantità d'acqua non superiore a quella disponibile.

Vincoli normativi

I vincoli normativi sono invece legati a restrizioni legate a norme legislative e quindi appunto normative.
Nell'esempio in questione i vicoli sono i seguenti:

$X_1 \geq a_1$
$X_4 \geq a_3$
$X_3 \geq a_5$

Analizzando tali vincoli si notano alcuni tratti comuni.
Infatti come si vede X_1, X_4 e X_3 devono essere maggiori di determinati parametri costanti, a_1, a_3 e a_5, che devono essere fissati per legge. Essi rappresentano quindi il fabbisogno minimo dei vari agenti che rappresentano le città.
Gli ultimi tre vincoli che definiscono i vincoli hard sono i seguenti:

$Q_1 - X_1 \geq a_2$
$Q_2 – X_4 \geq a_4$
$X_2 + Q_2 – X_4 - X_3 \geq a_6$

Anche questi tre vincoli hanno caratteristiche comuni, cioè sono legati agli agenti che rappresentano le città. In particolare indicano che la portata di acqua risultante dal flusso principale meno l'acqua prelevata dalle città, deve essere maggiore di determinate soglie, in questo caso a_2, a_4 e a_6. Esse difatti rappresentano il minimo livello di acqua che deve scorrere nel fiume.

Vincoli di utilità

Per parlare dei vincoli di utilità bisogna innanzi tutto descrivere le assunzioni che sono state fatte riguardo le funzioni di utilità descritte nell'articolo di Yang, Cai e Stipanovic da cui prende spunto questo lavoro di tesi [1] [2].
Secondo tali assunzioni ogni agente ha una funzione di utilità che è una parabola con concavità verso il basso.
In questo modo questa è la funzione di utilità che appare:

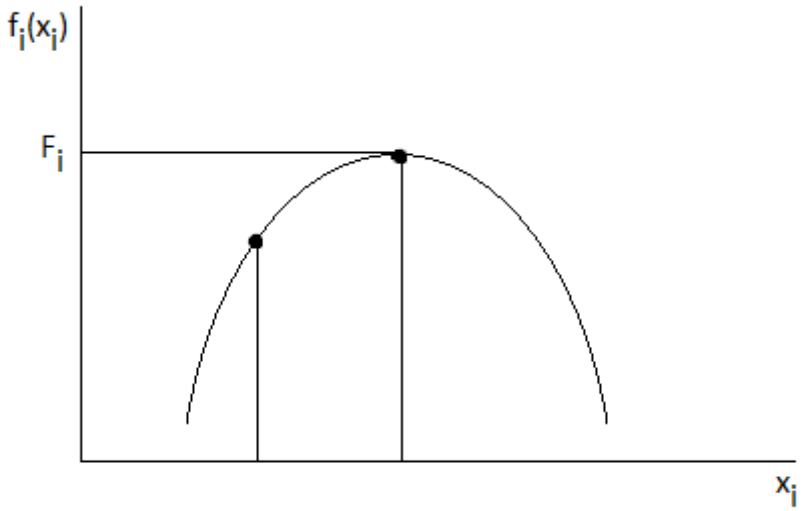

Fig. 4.2: Grafico della funzione di utilità di un generico agente

Come si vede quindi la funzione di utilità e massima nel vertice della parabola.
Tale valore è qui indicato con F_i.
Di conseguenza ogni agente tenderà ad aumentare la quantità di acqua prelevata, dato che inizialmente la quantità prelevata è nulla, aumentando quindi il valore assegnato alla propria variabile X_i.
Quindi esiste un vincolo soft per ogni agente, che cerca di garantire a ogni agente la massima utilità.
Il vincolo sarà quindi di questo tipo:

$$K_i = F_i - f_i(x_i)$$

dove K_i è il costo locale, cioè il costo relativo ad ogni agente. Può essere considerata come la mancata utilità realizzata dall'agente, essendo infatti calcolata come la semplice sottrazione tra F_i e $f_i(x_i)$.
Il problema avrà poi quindi come ultimo vincolo ulteriore la minimizzazione totale, cioè:

$$\min \Sigma K_i$$

Si sommano quindi tutti i K_i e questo rappresenta il costo totale del problema e tale costo deve quindi essere minimizzato.

4.2 Automatizzazione nella formalizzazione del problema

A questo punto abbiamo notato nei precedenti paragrafi che è possibile riscontrare dei tratti comuni nei vincoli associati ad ogni categoria di agente. Nel corso dell'analisi del problema è stato infatti possibile trovare dei vincoli caratteristici validi per ogni agente che

ora saranno illustrati.
Matematicamente sono vincoli molto semplici, ma la loro schematizzazione è fondamentale, come si vedrà nel prossimo capitolo.
La schematizzazione permette infatti di implementare un software che consente di generare in automatico scenari idrici casuali e coerenti, che rispettino tutti i vincoli necessari per formalizzare i problemi in questione.
Nei prossimi paragrafi vengono quindi illustrati i vincoli caratteristici per ogni categoria di agente.

4.2.1 Vincoli dell'agente città

L'agente preso in considerazione è una città e controlla una variabile X.
La situazione è illustrata in Figura 4.3.
L'agente X preleva una certa quantità d'acqua e il flusso rimanente che scorre verso il basso è $Q_1 – X$.
I vincoli da inserire sono:

$X \leq Q_1$
$X \geq a$ (a è la quantità minima di cui l'agente ha bisogno)
$Q_1 – X \geq b$ (b è la quantità minima che deve continuare a scorrere).

Q_1 ovviamente rappresenta la portata del flusso principale.
Graficamente l'aggiunta di tali vincoli può essere così rappresentata:

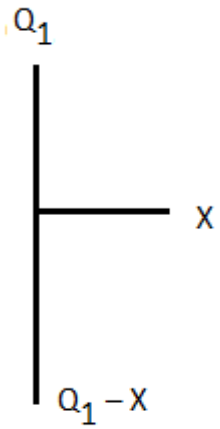

Fig. 4.3: Rappresentazione grafica dell'agente città

4.2.2 Vincoli dell'agente diga

Nel caso dell'agente diga lo schema grafico sarebbe quello della figura seguente:

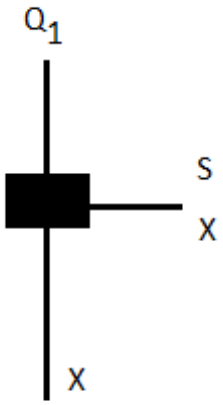

Fig. 4.4: Schema di rappresentazione grafica dell'agente diga

Il significato è quello fino ad ora concordato, cioè la diga rilascia una quantità X (che rappresenta l'agente) che non può essere maggiore della portata che scorreva fino a quel momento sommata alla riserva S che la diga stessa possiede.
Quindi l'unico vincolo da aggiungere è in questo caso $X \leq Q_1 + S$.

4.2.3 Memorizzazione dell'affluente nella portata d'acqua

Il seguente vincolo non è stato espresso esplicitamente nella formalizzazione matematica della sezione 4.2, ma è fondamentale.

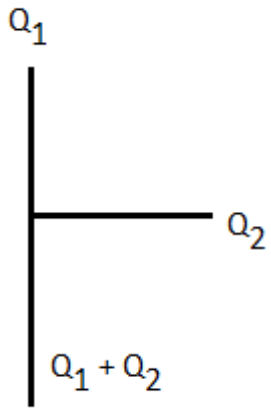

Fig. 4.5: Schema di rappresentazione grafica dell'affluente

Esso infatti permette di tener traccia dell'aggiunta di un affluente secondario che si immette nel flusso primario. In questo caso quindi bisogna tener traccia della nuova portata che scorrerà nel fiume.
L'afflusso non fa altro che rappresentare un altro fiume che si immette nella portata principale.
Viene rappresentato in Figura 4.4.

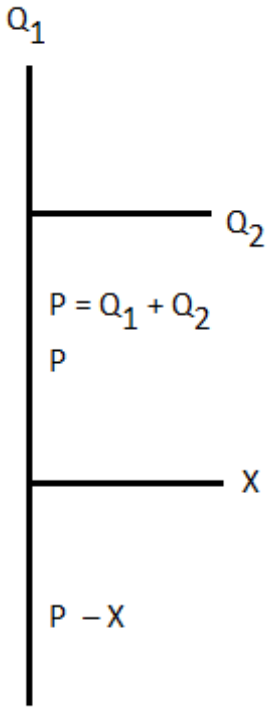

Fig. 4.6: Rappresentazione di schema idrico con variabili di supporto

Come si vede a questo punto la portata che scorre sempre verso il basso aumenta, dato che ad essa viene sommata quella dell'affluente.
In realtà tale vincolo era stato rappresentato anche in Figura 4.1, ma in modo implicito, cioè sommando o sottraendo di volta in volta l'acqua prelevata o aggiunta al flusso principale. Non veniva quindi utilizzata una variabile per tenere traccia della nuova portata o un vincolo di uguaglianza per memorizzarla.
Questo invece potrebbe essere fondamentale nel caso dell'implementazione di un programma, dato che questo permetterebbe di tenere traccia delle variazioni nel flusso d'acqua.
Questi aspetti verranno poi chiariti nel prossimo capitolo. Tuttavia per evitare fraintendimenti è giusto specificare che l'aggiunta di un affluente non necessariamente impone la presenza di un vincolo di uguaglianza e di una variabile in cui memorizzare una variazione della portata del fiume.
Possiamo dire che esistono due possibili implementazione di tale situazione.

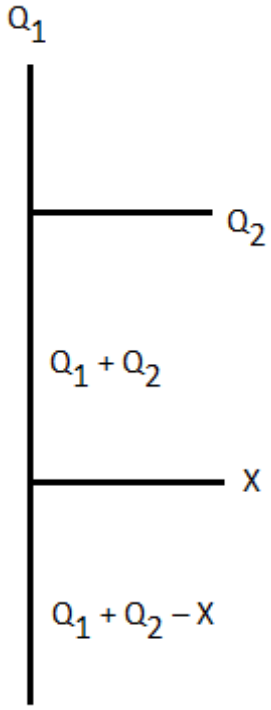

Fig. 4.7: Rappresentazione di schema idrico senza variabili di supporto

La prima aumenta il numero di variabili, aggiungendone una nuova ogni volta che viene aggiunto un affluente. In questo modo la nuova variabile memorizza la nuova portata, ottenuta come somma della portata precedente e dell'affluente. In questo modo la nuova variabile viene utilizzata come riferimento della nuova portata. Tale situazione è illustrata nella Figura 4.6.
Come si vede in questo caso la complessità dei vincoli rimane inalterata, dato che

aggiungendo poi un agente che controlla la variabile X, il vincolo non risulta molto complesso, dato che deve gestire una sottrazione tra la variabile P e la variabile X.
In compenso aumenta però il numero delle variabili.
La seconda implementazione invece prevede di non aggiungere alcun tipo di variabile accessoria, ma al contrario aumenta la complessità dei vincoli. Tale situazione è espressa chiaramente dalla Figura 4.7.
Difatti all'aumento di affluenti (o variabili) la portata si modifica, mediante sottrazioni o somme di portate. Come infatti si vede nella Figura 4.7 la portata si modifica è diventa più complessa, tenendo traccia di tutte le precedenti modifiche. Se immaginiamo uno scenario con molte variabili in serie ci possiamo rendere conto di come i vincoli diverranno sempre più lunghi e complessi.
Questa ultima è stata tuttavia la via scelta per implementare il programma illustrato nei prossimi capitoli, dato che meglio si adatta alle tecnologie utilizzate.
La scelta è stata fatta a causa delle prestazioni del simulatore FRODO, che verrà descritto nel prossimo capitolo e che non è in grado di gestire un sistema con un numero eccessivo di variabili.

4.2.4 Esempio di modellizzazione

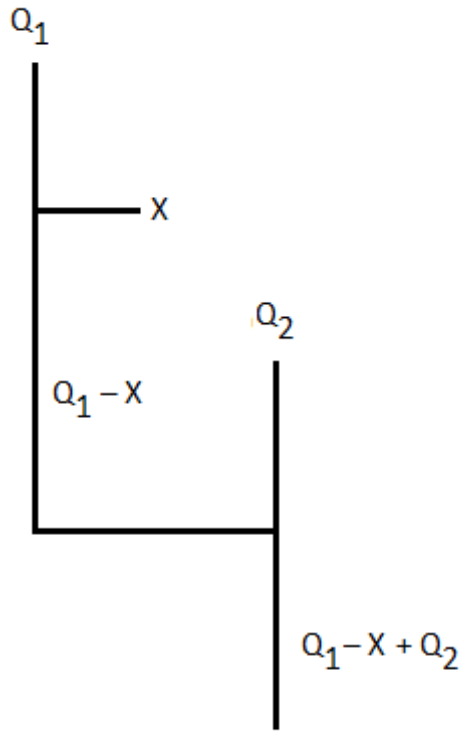

Fig. 4.8: Esempio di scenario da modellizzare

Questo paragrafo, che conclude il capitolo, illustra un esempio pratico in cui vengono applicate le indicazioni descritte nei precedenti paragrafi.
Innanzi tutto viene mostrato uno scenario in figura 4.8 e successivamente ne vengono specificati i vincoli.

Come vediamo l'esempio è molto semplice, entrando in gioco un solo agente.
L'unico agente è una città che preleva acqua dall'affluente.
I vincoli saranno dunque:

$X \leq Q_1$
$X \geq a$
$Q_1 - X \geq b$

con a e b dati.
Infine è possibili scegliere due strade per memorizzare la variazione della portata principale.

La prima è di utilizzare una variabile accessoria, P, per memorizzare la variazione della portata d'acqua.
In questo modo risulta $P = Q_1 – X + Q_2$.
La seconda alternativa è di non aggiungere nessun vincolo o variabile, ma in futuro se verranno aggiunti nuovi agenti a valle della situazione mostrata la portata su cui dovranno lavorare sarà $Q_1 – X + Q_2$, il che renderà i vincoli molto più complessi.
Se ad esempio venisse aggiunto un altro agente città, X_2, sulla portata principale i nuovi vincoli sarebbero:

$X_2 \leq Q_1 – X + Q_2$
$X_2 \geq a$
$Q_1 – X + Q_2 – X_2 \geq b$.

assumendo ovviamente che a e b siano uguali per tutte le città.

4.3 Implementazioni

Come si è visto i paragrafi precedenti hanno sottolineato per l'agente città e per l'agente diga quali siano i vincoli fisici e quali siano i normativi.
Nel corso di questo lavoro di tesi sono state presentate due diverse formalizzazioni del problema.
La prima considera come vincoli hard sia i vincoli fisici che quelli normativi.
Questo implica una formalizzazione molto più restrittiva, dato che in questo modo il numero di vincoli da rispettare è molto più alto, portando ad simulazioni più lunghe e complesse.
In questo caso il costo da minimizzare è solo quello dato dalle funzioni di utilità.
Questo tipo di formalizzazione sarà richiamata nel seguito di questo elaborato col nome di “Formalizzazione 1”.
Il secondo tipo di formalizzazione prevede invece di considerare come hard solo i vincoli fisici, considerando invece come soft i vincoli normativi e di utilità.
Questo porta quindi ad avere un insieme delle soluzioni accettabili più ampio del caso precedente. In questo caso il costo da minimizzare è dato dalle funzioni di utilità a cui viene sommato quello dato dai vincoli normativi non rispettati.
Per valutare il costo di un vincolo normativo violato si procede in modo molto semplice.
Ad esempio se assumiamo di avere il seguente vincolo normativo:

$a + X_1 - K \leq X_2$

assumendo i seguenti valori:
a=3, X_1 = 4, K = 1 e X_2 = 1 si otterrebbe la disuguaglianza 6 <= 1, che risulta palesemente

falsa.
Il costo è quindi dato da 5, cioè quel valore che dovrebbe essere aggiunto al secondo membro per verificare il vincolo.
Sommando tutti i costi dei vincoli normativi violati e aggiungendo tale quantità al costo ottenuto dai vincoli di utilità si ottiene il costo totale da minimizzare.
Questa seconda formalizzazione sarà richiamata nel seguito di questo elaborato col nome di “Formalizzazione 2”.

5. Architettura del sistema

5.1 Strumenti e tecnologie utilizzate

5.1.1 FRODO

FRODO è un framework Java open-source utilizzato per problemi di ottimizzazione di vincoli distribuiti [38]. Per questo lavoro di tesi è stata utilizzata la versione di FRODO 2.10.4, che è stata completamente riprogettata e reimplementata a partire dalla versione iniziale sviluppata da Adrian Petcu [39]. FRODO supporta molti algoritmi per risolvere problemi DCOP, come SynchBB, MGM, MGM-2, ADOPT, DSA, DPOP, S-DPOP, O-DPOP, MB-DPOP, Max-Sum, ASO-DPOP, P-DPOP, P2-DPOP e E[DPOP]. Come già citato, solo quattro di questi algoritmi sono stati utilizzati per questo lavoro.
FRODO ha un'architettura modulare organizzata in tre strati, che formano tre livelli, il livello di comunicazione, il livello dello spazio delle soluzioni e il livello degli algoritmi.
Il livello di comunicazione è responsabile del passaggio di messaggi tra gli agenti.
I messaggi sono scambiati tramite memoria condivisa, se gli agenti risiedono sullo stesso elaboratore e quindi sulla stessa Java Virtual Machine, o tramite TCP in un reale ambiente distribuito, cioè se gli agenti sono collocati su elaboratori diversi.
In FRODO ogni agente ha la propria coda che utilizza per ricevere e mandare i propri messaggi. Il livello dello spazio delle soluzioni è una piattaforma disegnata per risolvere problemi di ottimizzazione.
Lo spazio delle soluzioni fornisce classi che possono essere utilizzate per modellare questi tipi di problemi. Dato uno spazio di possibili assegnamenti alle variabili, uno spazio delle soluzioni è una rappresentazione degli assegnamenti in modo tale che essi siano una soluzione del problema. E' possibile poi parlare di spazi di soluzioni anche in merito all'utilità, privilegiando determinate soluzioni che forniscono maggiore utilità agli agenti.
L'ultimo livello è quello degli algoritmi, che, costruito sopra il livello precedente, è utilizzato per implementare gli algoritmi che verranno scelti per risolvere i vari problemi.
In FRODO gli algoritmi sono implementati in modo modulare in modo da semplificare la loro modifica o il loro aggiornamento.
Con questo paragrafo finisce la breve trattazione di FRODO, che risulterà il simulatore da noi scelto per risolvere tutti i nostri problemi.
Per maggiori dettagli si faccia riferimento alla bibliografia [38] [39] [40].

5.1.2 MATLAB

MATLAB (abbreviazione di Matrix Laboratory) è un ambiente per il calcolo numerico e l'analisi statistica che comprende anche l'omonimo linguaggio di programmazione creato dalla MathWorks. MATLAB consente di manipolare matrici, visualizzare funzioni e dati,

implementare algoritmi, creare interfacce utente, e collaborare con altri programmi. MATLAB è usato da milioni di persone nell'industria e nelle università per via dei suoi numerosi tool a supporto dei più disparati campi di studio applicati e funziona su diversi sistemi operativi, tra cui Windows, Mac OS, GNU/Linuz e Unix [41].
Per realizzare questo lavoro è stata poi utilizzata una libreria open-source, che ha facilitato la creazione di file XML mediante il linguaggio MATLAB.
Il nome di questa libreria è 'xml_toolbox_pcode_20110801' ed è liberamente scaricabile. Anche in questo caso si rimanda alla bibliografia per ulteriori dettagli [42].

5.1.3 XML

XML (sigla di eXtensible Markup Language) è un linguaggio di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo. Utilizzando XML è possibile creare strutture innestate che ben descrivono una struttura dati. Esse possono poi essere navigate senza difficoltà da programmi di gestione XML e cosi i dati possono essere recuperati facilmente [43].

5.2 Progettazione e sviluppo del sistema

5.2.1 Architettura e funzionamento

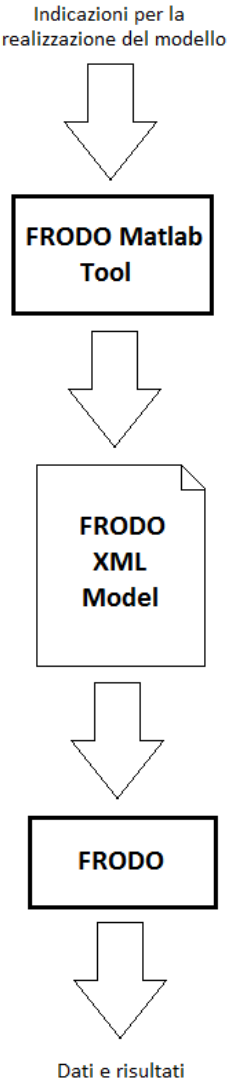

Fig. 5.1: Architettura del sistema

La Figura 5.1 mostra il funzionamento dell'intero programma sviluppato in questo lavoro di tesi. Esso è un tool utilizzato insieme a FRODO per generare scenari idrici con determinati parametri.
Come si vede innanzi tutto l'utente fornisce le indicazioni per realizzare il modello.
Il tool implementato crea il modello generando un file XML che descrive opportunamente il modello richiesto. Successivamente tale file XML viene trasmesso a FRODO, che si occupa di interpretarlo e risolvere il problema in esso descritto.
Infine vengono mostrati all'utente i risultati.
Come si vede quindi l'architettura è modulare e organizzata in strati.
Vediamo adesso nel dettaglio come l'intero sistema funziona.
Innanzi tutto l'utente può decidere di generare il modello in base a diverse funzionalità offerte dal sistema, che però non saranno

completamente prese in esame in questo paragrafo.
Le principali funzionalità saranno infatti introdotte nella prossima sezione.
L'utente potrà generare un modello creando un'opportuna lista in linguaggio MATLAB.
La lista infatti consente di utilizzare diversi livelli di annidamento e in questo modo consente di descrivere un corso d'acqua con i suoi affluenti.
Ad esempio consideriamo la seguente lista:

[Q_1, ('city agent', X_1, a_1, b_1, est_inf_D_1, est_sup_D_1, a_par$_1$, b_par$_1$, c_par$_1$), ('dam agent', X_2, S, est_inf_D_2, est_sup_D_2, a_par$_2$, b_par$_2$, c_par$_2$), (Q_2, ('city agent', X_3, a_3, b_3, est_inf_D_3, est_sup_D_3, a_par$_3$, b_par$_3$, c_par$_3$))]

Il significato di questa lista è insito nella struttura dati.
Quello che verrà quindi generato è un corso di acqua di portata Q_1 dove in successione ci saranno prima un agente città che controlla la variabile X_1 e subito dopo un agente diga che controlla la variabile X_2. Ovviamente all'interno della lista si può creare una seconda lista innestata che sta a significare la creazione di un affluente che si immette nel corso principale.
Questo è quello che infatti avviene, dato che il terzo elemento della lista è una sotto-lista, che sta ha indicare che nel corso d'acqua principale si immette un affluente di portata Q_2 su cui è presente un agente città che controlla la variabile X_3.
Come poi si nota ogni agente possiede parametri che lo descrivono.
Essi sono i parametri a_i e b_i, che formalizzano le soglie presentate nei vincoli descritti nel Capitolo 4, gli estremi superiore e inferiore del dominio della variabile x_i ed infine i tre parametri a_i, b_i e c_i che descrivono la parabola di utilità della variabile x_i.
Si può quindi vedere come sia possibile descrivere un modello idrico mediante la struttura a lista.
L'utente quindi può decidere di creare autonomamente una lista o utilizzare gli strumenti forniti dal tool per creare uno scenario casuale, con un numero fissato di agenti città e agenti diga, i cui parametri vengono scelti casulamente all'interno di un certo intervallo di valori.
Generata la lista, il tool offre strumenti per tradurre tale lista in un file XML.
Prendiamo un esempio molto semplice, un modello con solo due agenti, una città e una diga.
Il file XML risultante avrà la seguente struttura:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<instance>
<presentation format="XCSP 2.1_FRODO" maxConstraintArity="30" maximize="false"
name="TEST1_1_city_1_dam_MGM_simulation_2"/>
        <agents nbAgents="2">
            <agent description=" " id="X2" name="X2"/>
```

```
        <agent description=" " id="X4" name="X4"/>
    </agents>
    <domains nbDomains="1">
        <domain name="D" nbValues="50">1..50</domain>
    </domains>
    <variables nbVariables="2">
        <variable agent="X2" description="Variable_var_X2" domain="D" id="var_X2" name="var_X2"/>
        <variable agent="X4" description="Variable_var_X4" domain="D" id="var_X4" name="var_X4"/>
    </variables>
    <predicates nbPredicates="4">
        <predicate name="constraint_X21">
            <parameters>int x int q int s</parameters>
            <expression>
                <functional>le(x,add(q,s))</functional>
            </expression>
        </predicate>
      ....
    </predicates>
    <functions nbFunctions="2">
        <function name="utility_constraint_X2" return="int">
            <parameters>int f_max int a int b int c int x</parameters>
            <expression>
                <functional>sub(f_max,add(mul(a,mul(x,x)),add(mul(b,x),c)))</functional>
            </expression>
        </function>

      ....
  </functions>
  <constraints nbConstraints="6">
    <constraint arity="1" name="utility_constraint_X2" reference="utility_constraint_X2" scope="var_X2">
        <parameters>2605 -6 250 0 var_X2</parameters>
    </constraint>
  ....
  </constraints>
</instance>
```

Commentiamo ora il codice XML.
Innanzi tutto si notano alcune parti in grassetto che sono state evidenziate dato che rappresentano le porzioni di codice fondamentali all'interno del file.
Inoltre non è stato riportato interamente il codice per evitare ripetizioni non utili per questa esposizione. Al loro posto sono stati messi dei puntini di sospensione, ad indicare quindi del codice che non viene qui riportato ma che è presente nel file originale.
Come si vede la radice dell'intero file è il tag 'istance', che funge quindi da nodo principale.
Subito dopo il primo nodo figlio è il tag 'presentation', all'interno del quale sono presenti diversi attributi. Il principale è 'maximize' che è assegnato con la stringa 'false', ad indicare che questo è un problema DCOP di minimizzazione, si devono cioè minimizzare i costi (cioè i pesi associati ai vincoli soft sono visti come costi e non come utilità).
Successivamente si nota il tag 'agents' che contiene i due sotto-tag 'agent', che servono a

dichiarare l'esistenza di due agenti, nello specifico X2 (una diga) e X4 (una città).
Il tag successivo, 'domains', è utilizzato per dichiarare l'esistenza di uno o più domini. In questo caso è presente un solo domino, D, con 50 valori (dato che è possibile creare solo domini finiti) a partire da 1 fino a 50. Tale dominio è dichiarato nel tag figlio 'domain'.
Il tag seguente è 'variables'. Al suo interno i tag 'variable' servono a dichiarare le variabili del problema, segnalando nei loro attributi il domino, il nome e l'agente di riferimento che si occuperà del loro assegnamento.
A questo punto devono essere implementati i vincoli.
Per fare ciò si percorrono 3 passi.
Il primo consiste nel dichiarare i vincoli hard. Essi sono dichiarati come predicati nel tag 'predicates' e nei sotto-tag 'predicate'.
Ogni predicato ha un nome (nell'attributo 'name'), dei parametri che vengono dichiarati nel tag 'parameters' e una implementazione nel tag 'functional'. L'implementazione del vincolo è scritta in formato XCSP [44] [45].
Come quindi si nota la dichiarazione del predicato avviene come fosse una funzione.
Nel nostro esempio i puntini di sospensione indicano non sono riportati tutti i vincoli per motivi di semplicità.
In modo equivalente vengono implementati i vincoli soft, uno per ogni agente, utilizzando non più però i tag 'predicates' e 'predicate', ma i tag 'functions' e 'function'.
Come già illustrato precedentemente, tali vincoli sono utilizzati per l'ottimizzazione e quindi per privilegiare l'utilità dei vari agenti nel caso delle funzioni di utilità e per rappresentare i vincoli normativi nel caso si decida di formalizzarli come vincoli soft.
Compiendo questi due passi si sono implementati tutti i vincoli, ma a questo punto è necessario istanziarli.
Per fare ciò è necessario compiere l'ultimo passo, cioè utilizzare il tag 'constraints' e al suo interno tanti tag 'constraint' quanti sono i vincoli hard e soft.
In questo modo si passano a tali vincoli i parametri numerici adeguati per poter risolvere il problema.
Anche qui per semplicità non sono riportati tutti i vincoli e al loro posto sono utilizzati i soliti puntini di sospensione.

A questo punto il file XML è completo e descrive il modello in modo esauriente.
Il file può ora essere utilizzato da FRODO che lo userà per risolvere il problema DCOP in esso descritto.
La risoluzione avverrà mediante l'algoritmo selezionato dall'utente.

5.2.2 Funzionalità

L'obbiettivo di questo paragrafo è di mostrare direttamente le potenzialità del tool, descrivendo brevemente ad alto livello tutte le funzionalità proposte dal programma implementato.

Il tool è composto di diverse funzioni MATLAB, che sono file con estensione *.m* che contengono codice MATLAB.
Le prime funzioni che sono state implementate sono quelle che consentono di creare e manipolare file XML, creando quindi modelli di problemi DCOP, aggiungendo variabili e vincoli.
L'utente può quindi creare un modello anche partendo da queste funzioni, che consentono un immediata creazione del file XML.
Un metodo meno dispendioso è però quello di creare una lista, come esposto nel precedente paragrafo, di tipo *cell array*. Cell array è una struttura dati in MATLAB che può essere utilizzata per creare strutture dati a lista, con anche sotto-liste innestate.
Il tool a questo punto fornisce funzioni per tradurre la lista in un file XML che ne descrive opportunamente agenti e vincoli.
Sono disponibili due differenti possibili traduzioni, come esposto nel precedente capitolo, quella che crea variabili accessorie e mantiene i vincoli semplici, e quella senza variabili accessorie che però complica notevolmente i vincoli aumentandone l'arità.
Il tool fornisce poi la funzionalità per eseguire FRODO direttamente all'interno dell'ambiente MATLAB, risolvendo quindi il problema DCOP descritto nel file XML generato.
Sono poi presenti funzioni di reportistica per creare file di log in formato HTML che raccolgono i dati delle esecuzioni.
Infine sono state implementate le funzionalità che permettono di generare i grafici, sia dei dati raccolti sia delle utilità ottenute dagli agenti a seguito dell'esecuzione.

6. Validazione sperimentale

6.1 Formalizzazione delle prove sperimentali

I test effettuati riprendono i due tipi di formalizzazioni descritti nel Capitolo 4, gestendo quindi due tipi di situazioni.
La prima è quella che è stata definita col nome di “Formalizzazione 1”.
Essa si propone di modellare i vincoli fisici e normativi come hard constraints, lasciando alla categoria dei soft constraints solo i vincoli di utilità.
Per essa sono stati utilizzati quattro algoritmi, DPOP, ASO-DPOP, MGM e SynchBB, raccogliendo dati riguardanti il tempo impiegato per eseguire un'istanza dell'algoritmo selezionato, il costo totale, cioè la somma dei pesi dei vincoli violati, il numero di messaggi scambiati e la loro dimensione totale in byte.
La seconda formalizzazione è quella che è stata definita col nome di “Formalizzazione 2”.
Essa riprende la Formalizzazione 1, trasformando però i vincoli normativi in soft constraints.
Per essa sono stati utilizzati tre algoritmi, DPOP, ASO-DPOP e MGM, raccogliendo gli stessi tipi di dati descritti nel caso della Formalizzazione 1.
SynchBB non è stato utilizzato a causa dei tempi di esecuzione eccessivi.
Per queste simulazioni sono stati utilizzati i valori numerici ripresi dall'articolo di Yang, Cai e Stipanovic [1].
In particolare i parametri per eseguire i test hanno un valore fisso in ogni simulazione o sono stati scelti casualmente in un intervallo fissato.
I parametri e i valori numerici a loro assegnati sono stati i seguenti:

- numero di affluenti o corsi d'acqua secondari: compresi da 0 a 5
- dominio delle variabili: compresi da 1 a 50
- parametro *a*: compreso tra 10 e 15
- parametro *b*: compreso tra 6 e 10
- parametri funzione di utilità agente città: $a_{città} = -20$, $b_{città} = 600$, $c_{città} = -500$
- parametri funzione di utilità agente diga: $a_{diga} = -6$, $b_{diga} = 250$, $c_{diga} = 0$

Le simulazioni della Formalizzazione 1 sono così strutturate.
Per ognuna delle quattro grandezze misurate sono state realizzate simulazioni con un numero variabile di agenti, fissando il numero di agenti di una categoria (ad esempio delle città) e aumentando progressivamente il numero di agenti dell'altra categoria (ad esempio dighe), variando il numero da 1 a 5.
Per ognuno di questi scenari sono state realizzate 10 simulazioni, eseguendo poi la media algebrica delle simulazioni significative, cioè quelle terminate prima del timeout

dell'algoritmo scelto, che è stato fissato a 600 secondi (dopo cioè 600 secondi l'algoritmo termina forzatamente e restituisce l'assegnamento corrente delle variabili).
Per fare ciò sono state considerate significative le simulazioni terminate prima di 570 secondi, in modo da avere risultati di sicura significatività.
Inoltre la quantità di acqua presente nel sistema (cioè la somma di tutte le portate Q_i e dei laghi artificiali S_i) è stata valutata in modo da essere pari al numero degli agenti del problema da risolvere (cioè la somma del numero delle dighe e del numero delle città), moltiplicato per un fattore 20.
Infine l'acqua inizialmente disponibile nel fiume è pari ad una percentuale dell'acqua totale disponibile nel sistema, scelta casualmente tra il 30% e il 50%.
In questo modo si è cercato di osservare e analizzare il comportamento del simulatore in scenari di complessità crescente.
Le simulazioni della "Formalizzazione 2" sono strutturate in modo identico, ma facendo variare il numero di agenti da 1 a 4 per categoria.
Nei prossimi paragrafi verranno mostrati grafici che illustrano i risultati ottenuti e infine si validerà il simulatore implementato alla luce dei risultati ottenuti.

6.2 Risultati delle prove sperimentali

Nel seguito vengono mostrati i grafici che raccolgono i risultati delle simulazioni effettuate.
Viene presentata prima la Formalizzazione 1 ed in seguito la Formalizzazione 2.
Le indicazioni sugli assi di ordinata e ascissa per ogni figura indicheranno a che tipo di simulazione si riferiscono.
Non sono riportati tutti i grafici, ma solo i più significativi. Questa scelta è dovuta al fatto di non voler appesantire eccessivamente l'esposizione.
Inoltre i grafici non illustrati sono stati scartati per non aver riportato un andamento significativo o comunque già descritto qualitativamente dagli scenari invece selezionati ed illustrati in seguito.

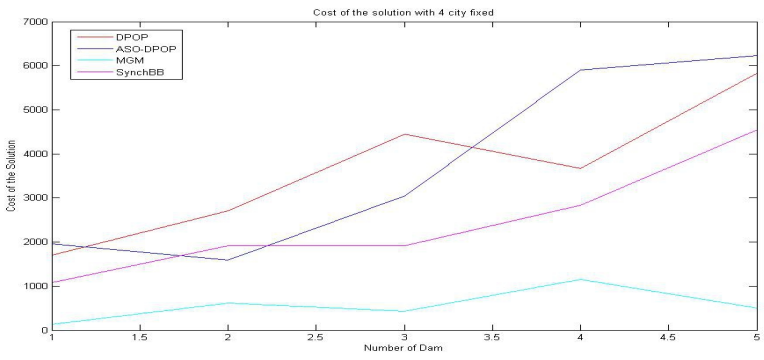

Fig. 6.1: Costo della soluzione nelle simulazioni della Formulazione 1 nel caso con 4 agenti città fissati

6.2.1 Formalizzazione 1

L'analisi viene presentata partendo dal **costo della soluzione** ottenuto nelle simulazioni della Formalizzazione 1.
Innanzi tutto si analizza il caso con il

numero di agenti città fissato.

Viene mostrata in Figura 6.1 la situazione con 4 città fissate, in modo da presentare il comportamento degli algoritmi con molti agenti in gioco.

Come si vede MGM

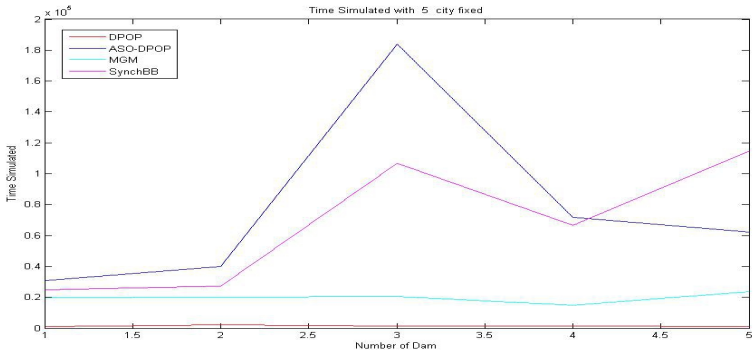

Fig. 6.2: Tempo di esecuzione nelle simulazioni della Formulazione 1 nel caso con 5 agenti città fissati

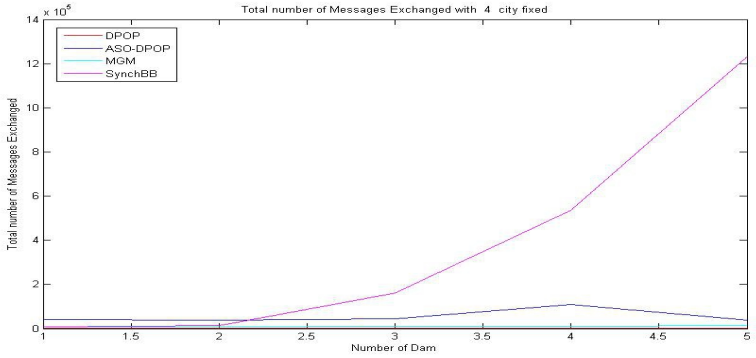

Fig. 6.3: Numero di messaggi scambiati nelle simulazioni della Formulazione 1 nel caso con 4 agenti città fissati

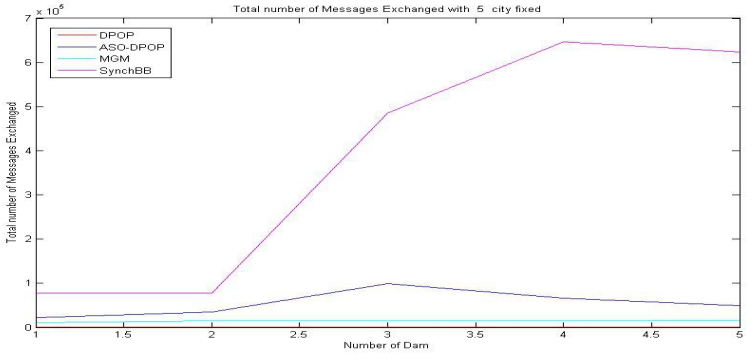

Fig. 6.4: Numero di messaggi scambiati nelle simulazioni della Formulazione 1 nel caso con 5 agenti città fissati

mantiene un costo molto basso nella soluzione, anche se non necessariamente ottimo, dato che questo algoritmo non garantisce di trovare la migliore soluzione esistente.

ASO-DPOP e DPOP invece dovrebbero trovare la soluzione ottima, cosa che però non avviene. Questo tipo di comportamento dovrebbe manifestarsi nel caso in cui l'algoritmo terminasse forzatamente.

Come illustrato precedentemente tuttavia le

simulazioni che sono state selezionate per ottenere i risultati presentati avrebbero dovuto terminare almeno circa 30 secondi prima del timeout, per cercare di ottenere risultati significativi.
Questa contraddizione può quindi essere spiegata come una inconsistenza nel simulatore FRODO, che quindi implementa versioni degli algoritmi DPOP e ASO-DPOP che non ricercano la soluzione ottima, ma solo sufficiente.
SynchBB propone soluzioni con un costo medio, dimostrando quindi di non essere l'algoritmo più efficente.
Analizzando la Figura 6.2 si possono tracciare le conclusioni per quanto riguarda il **tempo di esecuzione**. In questo caso è stato scelto uno scenario con 5 agenti città fissati, per osservare il comportamento degli agenti con un numero considerevole di attori in gioco.

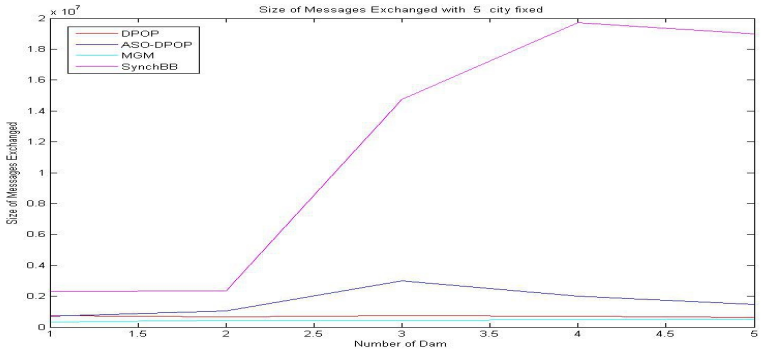

Fig. 6.5: Dimensione dei messaggi scambiati nelle simulazioni della Formulazione 1 nel caso con 5 agenti città fissati

Si nota come DPOP risulti essere molto rapido, insieme ad MGM. Questo evidenzia sia le ottime prestazioni dell'algoritmo MGM, sia come DPOP impieghi ovviamente un tempo non eccessivo per trovare una soluzione non ottima.
Questa ultima osservazione può essere considerata come un prima prova sperimentale a supporto della tesi di correttezza del tool implementato.
ASO-DPOP invece, nonostante rappresenti un'evoluzione di DPOP si dimostra

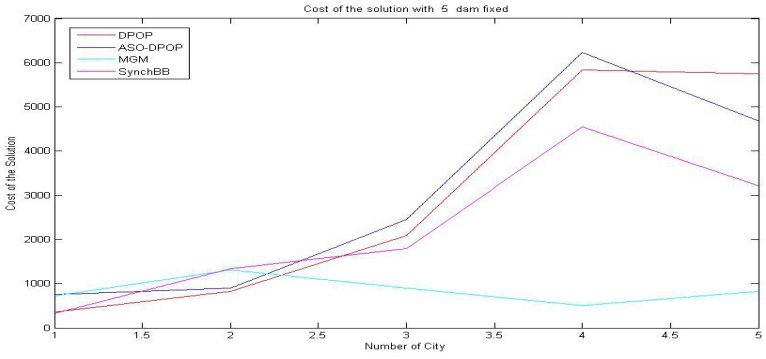

Fig. 6.6: Costo della soluzione nelle simulazioni della Formulazione 1 nel caso con 5 agenti diga fissati

molto lento. Analizziamo ora il **numero di messaggi scambiati** e la loro **dimensione**.
Anche in questo caso sono riportati gli scenari con 4 e 5 agenti città, essendo significativo soprattutto analizzare il comportamento asintotico degli algoritmi.

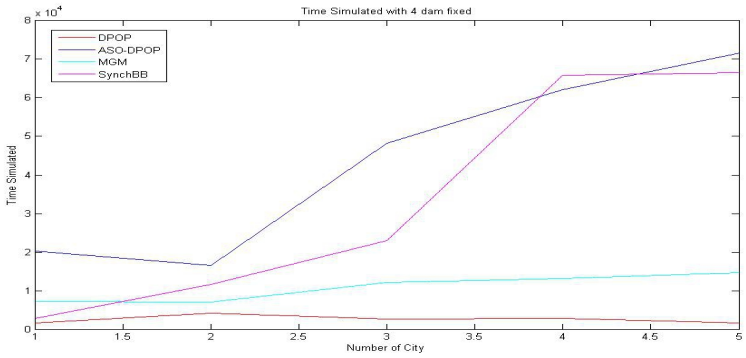

Fig. 6.7: Tempo di esecuzione nelle simulazioni della Formulazione 1 nel caso con 4 agenti diga fissati

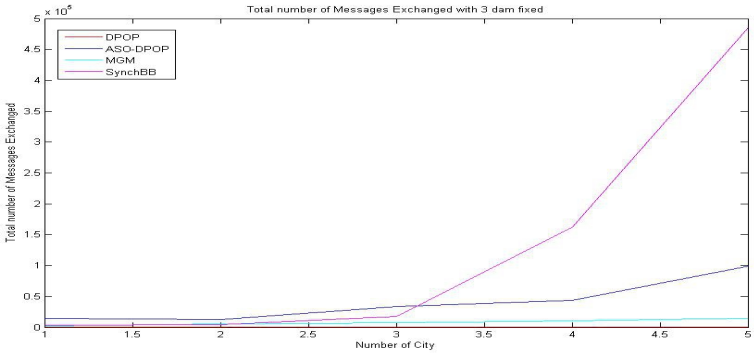

Fig. 6.8: Numero di messaggi scambiati nelle simulazioni della Formulazione 1 nel caso con 3 agenti diga fissati

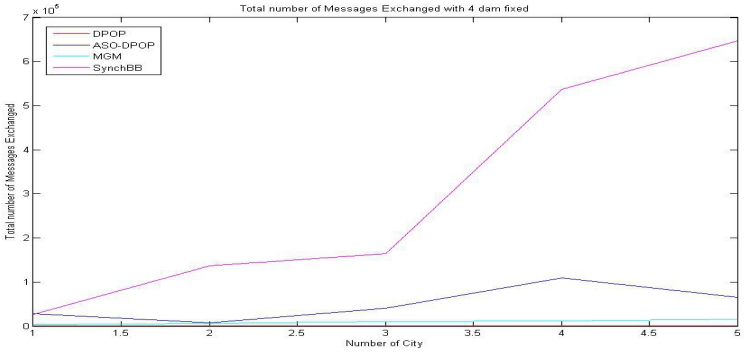

Fig. 6.9: Numero di messaggi scambiati nelle simulazioni della Formulazione 1 nel caso con 4 agenti diga fissati

Analizzando le Figure 6.3 e 6.4 si nota come DPOP riesca a reagire molto efficentemente all'aumento degli agenti per quanto riguarda il numero di messaggi inviati.
Un discorso analogo vale per l'altro algoritmo che utilizza l'albero dei vincoli, ASO-DPOP.
MGM, pur non utilizzando il grafo dei vincoli si dimostra molto efficente nel gestire il numero di messaggi scambiati, nonostante il costante aumento degli agenti diga.
SynchBB al contrario vede un incremento molto accentuato del numero dei messaggi, che dipende dall'aumentare degli agenti in gioco.
Come si vede, il numero di messaggi aumenta fino alla saturazione, mostrata in Figura 6.4.
L'andamento della

dimensione dei messaggi rispecchia le considerazioni appena descritte. Come si vede in Figura 6.5 le dimensioni totali dei messaggi scambiati risultano molto elevate per SynchBB (fino a saturare con 5 città e 5 dighe), mentre gli altri algortimi mantengono dimensioni totali contenute.
Come si nota fino a questo punto il tool si comporta coerentemente, mostrando per le quattro grandezze misurate dei dati verosimili e non in contrasto tra loro.
Da questa prima parte dell'analisi sperimentale si inizia quindi a delineare la correttezza del simulatore progettato.

Analizziamo ora la seconda parte dell'analisi della Formulazione 1, fissando il numero degli agenti diga. Anche in questo caso l'analisi parte dal **costo della soluzione**.
Analizziamo lo scenario con 5 agenti diga, essendo sicuramente il più significativo avendo un alto numero di agenti in gioco. In Figura 6.6 si nota come ancora MGM si dimostri un algortimo molto efficente, riuscendo a trovare soluzioni con costi molto bassi. Questo è spiegabile nuovamente con il fatto che questo algoritmo utilizza un approccio che si basa sulla teoria dei giochi, che si dimostra vincente con un numero di agenti contenuto (massimo 10 agenti). Come nel caso con gli agenti città fissati, DPOP e ASO-DPOP si dimostrano invece poco efficenti dal punto di vista del costo della soluzione. Ancora una

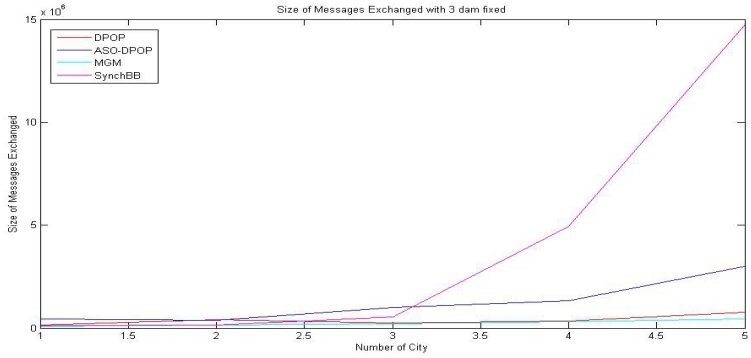

Fig. 6.10: Dimensione dei messaggi scambiati nelle simulazioni della Formulazione 1 nel caso con 3 agenti diga fissati

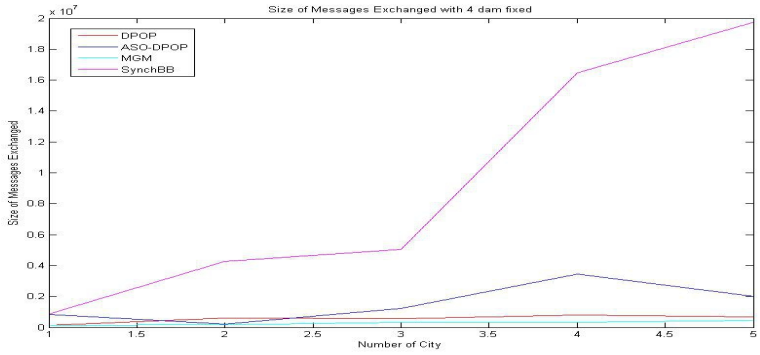

Fig. 6.11: Dimensione dei messaggi scambiati nelle simulazioni della Formulazione 1 nel caso con 4 agenti diga fissati

volta questo è spiegabile con il fatto che vengono violati dei vincoli hard.
Per quanto riguarda il **tempo di esecuzione** le considerazioni sono ancora una volta le stesse riportate precedentemente nel caso del numero di agenti città fissato e ciò è visibile dalla Figura 6.7.
In questo caso lo scenario rappresentato è quello con 4 agenti diga.
MGM e DPOP si dimostrano i più rapidi, mentre SynchBB risulta essere molto lento.
Prima di terminare l'analisi della Formalizzazione 1 bisogna presentare i risultati riguardanti il **numero di messaggi scambiati** e la **dimensione totale di informazione** scambiata nel corso dell'esecuzione degli algoritmi.
Come si nota dai grafici in Figura 6.8. e 6.9, esattamente come nel caso con numero di città fissato, il numero di messaggi scambiati rimane contenuto per tutti gli algotimi, a meno che per SynchBB.
Questo è spiegabile con il fatto che i problemi generati hanno un numero contenuto di agenti e ciò agevola DPOP, ASO-DPOP e MGM. SynchBB anche con un numero basso di agenti (al massimo 10 agenti in queste simulazioni) produce una grande quantità di messaggi, fino alla saturazione, visibile in Figura 6.9.
Per quanto riguarda la dimensione dei messaggi scambiati si possono tracciare considerazioni simili a quelle già descritte, oltre che a quelle riportate nella situazione analoga nel caso con gli agenti città fissati.
Per completezza in Figura 6.10 e 6.11 vengono mostrati i grafici che supportano queste conclusioni.

6.2.2 Formalizzazione 2

Alla luce dei risultati mostrati nell'analisi delle simulazioni della Formalizzazione 1, vengono ora mostrati i dati inerenti la Formulazione 2.
In questo secondo caso sono stati utilizzati solo 3 algoritmi, DPOP, ASO-DPOP e MGM.
Anche in questo caso l'analisi partirà dallo scenario con il numero di agenti città fissato.
Innanzi tutto consideriamo il **costo della soluzione**.
Prendendo come esempio il caso con 4 agenti città si possono notare in Figura 6.12 delle

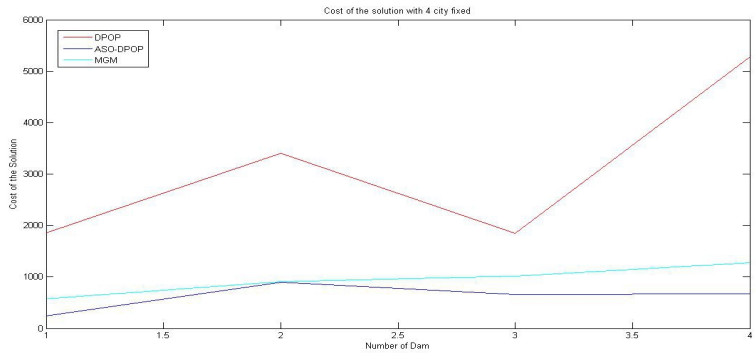

Fig. 6.12: Costo della soluzione nelle simulazioni della Formulazione 2 nel caso con 4 agenti città fissati

analogie con il caso della Formalizzazione 1, come ad esempio l'efficenza dell'algoritmo MGM.
DPOP al contrario produce le soluzioni più costose, mostrando poca efficenza anche in questa seconda formalizzazione.

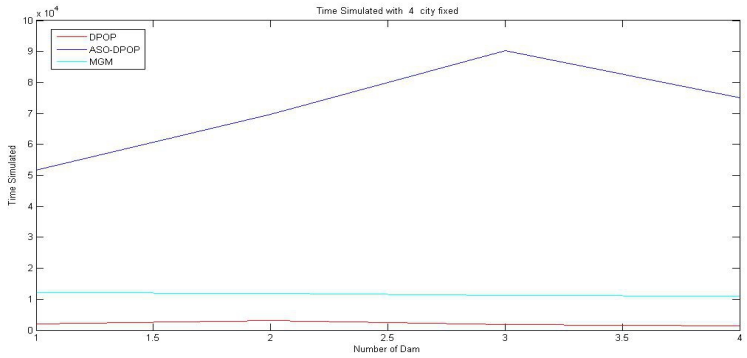

Fig. 6.13: Tempo di esecuzione nelle simulazioni della Formulazione 2 nel caso con 4 agenti città fissati

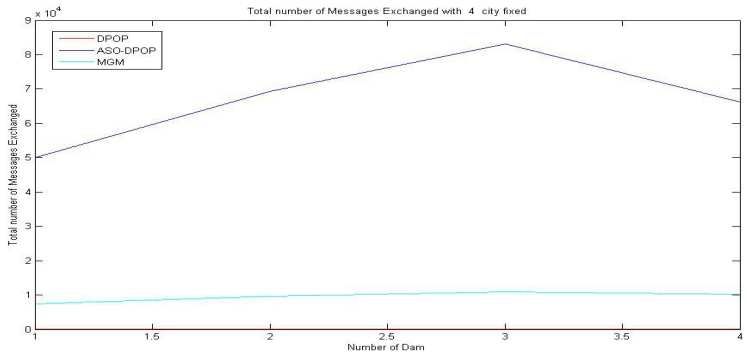

Fig. 6.14: Numero di messaggi scambiati nelle simulazioni della Formulazione 2 nel caso con 4 agenti città fissati

Nonostate la presenza di meno vincoli hard non riesce infatti a produrre soluzioni di costo accettabile.
DPOP si dimostra quindi poco flessibile, rispetto invece ASO-DPOP, che ne rappresenta infatti un'evoluzione.
ASO-DPOP riesce infatti a produrre soluzioni con costi molto vicini ad MGM, mostrando come una formulazione meno rigida consenta di sfruttare maggiormente le sue potenzialità.
Analizziamo adesso il **tempo di esecuzione**.
Prendendo come esempio la Figura 6.13, nel caso con 4 agenti diga fissati (con un numero totale di agenti variabile quindi tra 5 e 8) si notano risultati in linea con quanto appena osservato.
Producendo soluzioni molto efficenti ASO-DPOP impiega comprensibilmente un tempo molto elevato, a differenza di DPOP che si dimostra molto rapido, producendo soluzioni poco efficenti.
MGM si dimostra invece sempre molto efficente sia in termini di qualità che di efficenza.

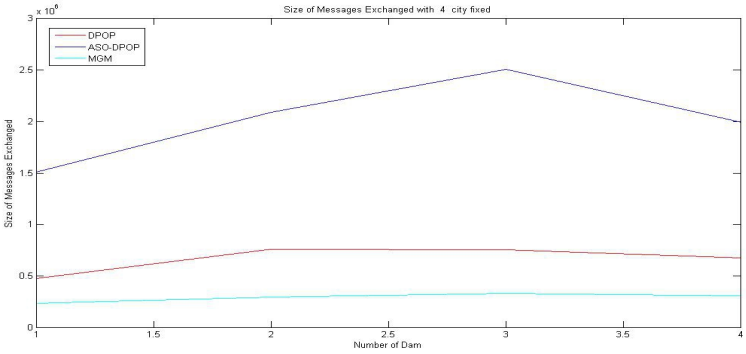

Fig. 6.15: Dimensione dei messaggi scambiati nelle simulazioni della Formulazione 2 nel caso con 4 agenti città fissati

Queste prove sperimentali non possono che validare ulteriormente la correttezza del tool implementato.
Analizziamo ora il **numero messaggi scambiati** e la loro **dimensione totale**.
Dalla Figura 6.14 si evince come nel caso di una formalizzazione con molti vincoli soft DPOP riesca ancora a gentire bene il numero di messaggi, a differenza di ASO-DPOP, che mantiene un numero molto elevato di messaggi scambiati, come nel caso della prima formalizzazione.
MGM si dimostra ancora un ottimo compromesso.
Per quanto riguarda le dimensioni valgono considerazioni simili, come testimonia il grafico in Figura 6.15.
L'unica differenza la si nota nel fatto che MGM scambia una dimensione di informazione inferiore rispetto a DPOP. Questo è spiegabile col fatto che DPOP scambia molti messaggi multidimensionali, e che quindi, anche se in numero limitato, risultano comunque di dimensione significativa.

Analizziamo ora l'ultima situazione, quella in cui viene <u>fissato il numero di dighe</u>.
Iniziamo l'analisi partendo dal **costo della soluzione**.
La situazione è descritta efficacemente dalla Figura 6.16, che analizza il caso con 4 agenti diga e un numero

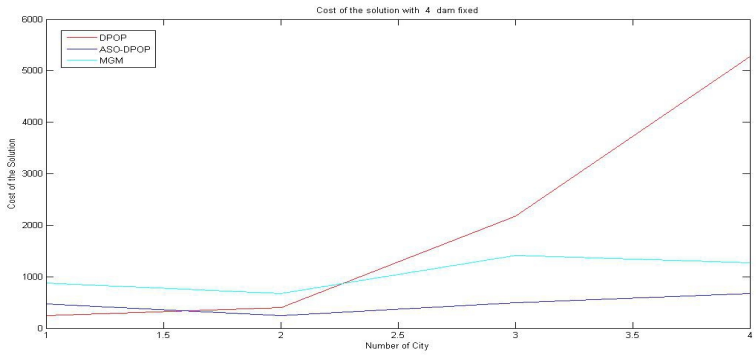

Fig. 6.16: Costo della soluzione nelle simulazioni della Formulazione 2 nel caso con 4 agenti diga fissati

crescente di agenti città.
Esattamente come nel caso di città fissate, ASO-DPOP si dimostra ancora l'algoritmo in grado di trovare soluzioni migliori.
MGM ancora una volta si pone in posizione intermedia. SI può quindi affermare che in problemi con un numero di agenti limitato e un elevato numero di vincoli soft ASO-DPOP sfrutta bene le proprie caratteristiche, mentre l'approccio DPOP fallisce.

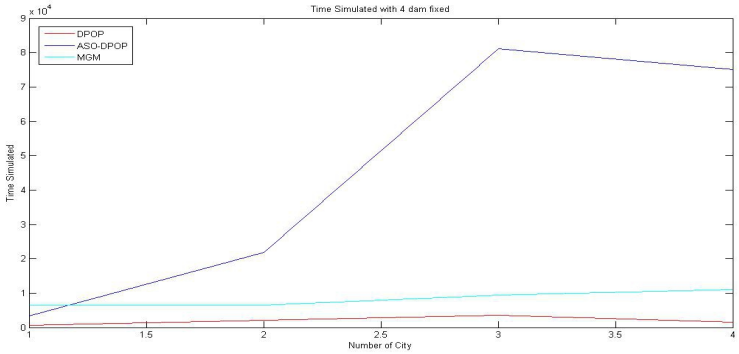

Fig. 6.17: Tempo di esecuzione nelle simulazioni della Formulazione 2 nel caso con 4 agenti diga fissati

Analizziamo ora il **tempo delle simulazioni**.
Osservando la Figura 6.17 (che rappresenta il caso con 4 agenti diga e un numero crescente di agenti città) si possono trarre delle considerazioni che sono profondamente legate a quelle legate al costo della soluzione.
Come si vede infatti, proprio come nel caso di città fissate, con un elevato numero di vincoli soft DPOP impiega un tempo molto basso, non riuscendo a trovare quindi la soluzione migliore, mentre invece ASO-DPOP risulta essere molto lento, anche se in grado di ottenere una soluzione molto inferiore in termini di costo.

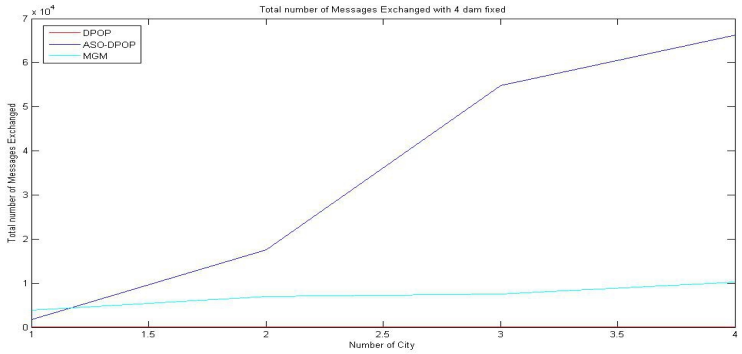

Fig. 6.18: Numero di messaggi scambiati nelle simulazioni della Formulazione 2 nel caso con 4 agenti diga fissati

MGM si dimostra nuovamente il miglior compromesso.
Queste conclusioni sono un'ulteriore conferma della correttezza del simulatore.
Concludiamo la sezione sperimentale analizzando il **numero di messaggi scambiati** e la loro

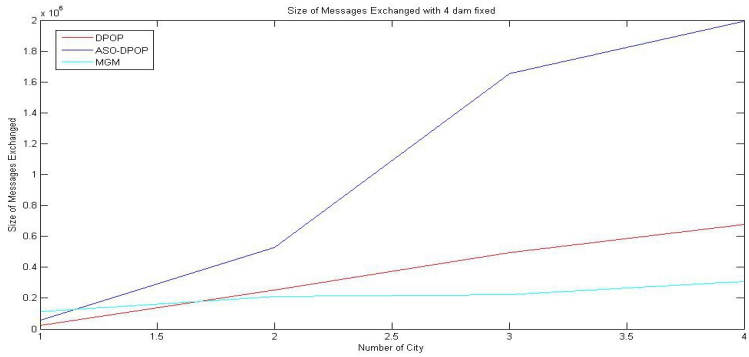

Fig. 6.19: Dimensione dei messaggi scambiati nelle simulazioni della Formulazione 2 nel caso con 4 agenti diga fissati

dimensione totale.
Riguardo queste grandezze si nota come per raggiungere una soluzione ottima ASO-DPOP necessiti di una grande elaborazione e quindi di un numero molto elevato di messaggi scambiati. Anche a questo si deve l'elevato tempo di esecuzione.
Al contrario invece MGM e DPOP vantano soluzioni meno efficenti, ma tempi di elaborazione inferiori, che si riflettono in una quantità inferiore di informazione scambiata, in termini di numero di messaggi e dimensione totale.
Queste conclusioni sono osservabili nelle Figure 6.18 e 6.19, dove è nuovamente presa in considerazione la situazione con 4 agenti diga.

6.3 Sintesi dei risultati

A questo punto è possibile tracciare una sintesi dei risulatti appena esposti.
Analizzando innanzi tutto la Formalizzazione 1 (dove cioè il numero di agenti varia da 1 a 5 per ogni categoria e gli algoritmi utilizzati sono DPOP, ASO-DPOP, MGM e SynchBB) si nota come sia ininfluente che il numero di agenti fissati siano città o dighe, dato che le prestazioni degli algoritmi sono identiche in entrambi i casi.
E' importante analizzare i risultati a livello asintotico, quindi per un numero alto di agenti.
Fatte queste premesse, i risultati sono sintetizzati dalla tabella seguente, dove il simbolo '<' viene utilizzato per ordinare in modo crescente le prestazioni degli algoritmi, mentre il simbolo '/' sta ad indicare che non è possibile fissare un preciso ordinamento tra gli algoritmi alla luce dei risultati sperimentali ottenuti.

Costo della soluzione	MGM < SynchBB < ASO-DPOP/DPOP
Tempo di esecuzione	DPOP < MGM < SynchBB/ASO-DPOP
Numero messaggi scambiati	DPOP < MGM < ASO-DPOP < SynchBB
Dimensione messaggi scambiati	DPOP/MGM < ASO-DPOP < SynchBB

In modo analogo si possono sintetizzare i risultati relativi alla Formalizzazione 2 (dove cioè il numero di agenti varia da 1 a 4 per ogni categoria e gli algoritmi utilizzati sono DPOP, ASO-DPOP e MGM):

Costo della soluzione	ASO-DPOP < MGM < DPOP
Tempo di esecuzione	DPOP < MGM < ASO-DPOP
Numero messaggi scambiati	DPOP < MGM < ASO-DPOP
Dimensione messaggi scambiati	MGM < DPOP < ASO-DPOP

Come si vede, fatta eccezione per quanto riguarda il costo della soluzione, gli andamenti per quanto riguarda le due formalizzazioni si dimostrano molto simili, indicando come per le restanti tre grandezze l'unica variabile che governa le prestazioni sia il numero totale di agenti nel problema.
Il numero di messaggi scambiati da ASO-DPOP risulta essere molto alto e ciò è spiegabile con il fatto che l'algoritmo impiega diverse fasi per raggiungere la convergenza. Esso inoltre dimostra di essere un algortimo piuttosto pesante anche a livello di tempo di esecuzione.
Tuttavia nella seconda formalizzazione si può apprezzare come le sue soluzioni siano quelle che mediamente hanno costo minore.
Questo è spiegabile con il fatto che ASO-DPOP riesce ad esprimere maggiormente le proprie potenzialità in un problema con molti vincoli soft rispetto ad una formulazione con un grande quantitativo di vincoli hard.
MGM risulta evidentemente l'algoritmo più efficente, riuscendo a scambiare un numero di messaggi contenuto, di dimensioni non elevate, avendo un tempo di esecuzione non eccessivo e ottenendo in entrambe le formalizzazioni dei costi non elevati.
Queste prestazioni sono spiegabili con il fatto che MGM non basa la risoluzione dei problemi su una struttura ad albero degli agenti, ma utilizzando la teoria dei giochi e una rappresentazione a grafo solo per tenere conto dei vincoli del problema.
Questo approccio si dimostra vincente in scenari con al massimo 10 agenti.
SynchBB al contrario mostra prestazioni molto inferiori, se rapportate agli altri algoritmi.
Il lungo tempo di esecuzione ad esempio può essere spiegabile con il fatto che esso non permette agli agenti di cambiare i loro assegnamenti in parallelo (cioè non permette quindi delle variazioni multiple del valore delle variabili) e ciò sicuramente degrada questo indice prestazionale.
Infine DPOP mostra prestazioni intermedie, con numero e dimensione dei messaggi contenuto, tempo di esecuzione non eccessivamente alto, ma presentando un alto costo della soluzione.

7. Conclusioni e lavori futuri

Questo capitolo conclude il lavoro di tesi esposto nelle precedenti pagine.
L'obbiettivo che è stato raggiunto è stato quello di realizzare un simulatore complesso basato su DCOP, utile in fase di ricerca e sperimentazione per quanto riguarda lo studio di dei bacini idrici.
La correttezza del simulatore è stata verificata sperimentalmente come illustrato nel Capitolo 6. Come si è visto il cuore del lavoro risiede nel generatore di problemi e nelle funzioni di parsificazione e traduzione, che trasformano la formalizzazione del problema, traducendolo dalle strutture dati di MATLAB al modello XML XCSP.

Ovviamente sono possibili molte estensioni.
Un possibile miglioramento potrebbe essere il rendere possibile la risoluzione di problemi formalizzati con l'aggiunta di variabili accessorie, come descritto nella sezione 4.2.3 del Capitolo 4.
Allo stato attuale è infatti disponibile nel tool la funzionalità che genera il file XML XCSP per FRODO che rispetta questo tipo di formalizzazione, ma la risoluzione è di fatto impossibile per problemi con più di 8 - 10 agenti.
Questo tipo di modifica andrebbe però eseguita all'interno del simulatore FRODO stesso, dato che la traduzione effettuata a monte della risoluzione mediante FRODO risulta infatti veloce e corretta.
Inoltre si potrebbe svolgere uno studio sperimentale più approfondito cercando di validare il simulatore utilizzando anche altri tipi di algoritmi e creando strumenti di reportistica adeguati a raccogliere i dati derivanti dalle simulazioni eseguite con questi nuovi algoritmi.
Si potrebbero svolgere anche simulazioni con un numero di agenti maggiore di 10, anche se in questo caso sarebbe consigliabile utilizzare elaboratori con prestazioni elevate, in modo da poter scalare il sistema efficacemente all'aumentare del numero degli agenti.
Infine una ulteriore estensione potrebbe essere quella di rendere l'intero simulatore distribuito anche fisicamente, permettendo quindi di lavorare con più elaboratori fisici tali per cui ogni agente risieda su un elaboratore e lo scambio di messaggi avvenga sulla infrastruttura di rete.

Bibliografia

[1] Yang, Cai, Stipanovic, *A decentralized optimization algorithm for multiagent system-based watershed managment*, Water Resources Research, Vol. 45, 2009

[2] Giuliani, Castelletti, Amigoni, Cai, *Multi-Agent Systems optimization for distributed watershed managment*, 2012 International Congress on Environmental Modelling and Software, Managing Resources of a Limited Planet, Sixth Biennial Meeting, Leipzig, Germany, 2012

[3] Russel, Norvig, *Artificial Intelligence: A Modern Approach,* Pearson Education*,* 2003

[4] Turing, *Computing Machinery and Intelligence*, Mind, 59, pp. 433-460, 1950

[5] Kurzweil, *The Age of Intelligent Machines*, MIT Press, 1990

[6] Wooldridge, *An Introduction to MultiAgent Systems*, John Wiley & Sons , 2009

[7] Wooldridge, *Reasoning about Rational Agents*, MIT Press, 2000

[8] Athanasiadis, *A review of agent-based systems applied in environmental informatics*, MODSIM 2005 Int'l Congress on Modelling and Simulation, pp. 1574-1580, 2005

[9] Haastrup, Würtz, *Environmental Data Exchange Network for Inland Water,* Elsevier Science, 2007

[10] Bayardo Jr., Bohrer, Brice, Cichocki, Fowler, Helal, Kashyap, Ksiezyk, Martin, Nodine, Rashid, Rusinkiewicz, Shea, Unnikrishnan, Unruh, Woelk, *InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments*, in Proceedings of the 1997 ACM International Conference on the Management of Data (SIGMOD), Tucson, Arizona, 1997

[11] *New Zealand Distributed Information Systems*, http://www.nzdis.org

[12] Purvis, Cranefield, Bush, Carter, McKinlay, Nowostawski, Ward, *The NZDIS project: an agent-based distributed information systems architecture,* System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on, 2000

[13] Neumann, Schuster, Stuckenschmidt, Visser, Vögele, *Intelligent brokering of*

environmental information with the BUSTER system, Sustainability in the Information Society - 15th International Symposium Informatics for Environmental Protection, pp. 505–512, 2001

[14] Kőppen-Seliger, Marcu, Capobianco, Gentil, Albert, Latzel, *MAGIC: An integrated approach for diagnostic data management and operator support*, in Proceedings of the 5th IFAC Symposium Fault Detection, Supervision and Safety of Technical Processes - SAFEPROCESS05, Washington D.C., pp. 2626-2628, 2003

[15] Albert, Längle, Wörn, Capobianco, Brighenti, *MULTI-AGENT SYSTEMS FOR INDUSTRIAL DIAGNOSTICS, In Proceedings of 5th IFAC Symposiumon Fault Detection, Supervision and Safety of Technical Processes, Washington DC.*, pp. 483-488, 2003

[16] Wörn, Längle, Albert, Kazi, Brighenti, Revuelta, Senior, Sanz-Bobi, Villar, *DIAMOND - Distributed Multi-Agent Architecture for Monitoring and Diagnosis,* EU Esprit Project No. 28735, Vo.15, Numer 2, pp. 189-200, 2004

[17] Kalapanidas, Avouris, *Air quality management using a multi-agent system*, Int. J. Comput. Aided Civil Infrastruct. Eng., Vol.17, pp. 119-130, 2002

[18] Nute, Potter, Maier, Wang, Twery, Rauscher, Knopp, Thomasma, Dass, Uchiyama, Glende, *NED-2: An agentbased decision support system for forest ecosystem management*, Environmental Modelling & Software, Vo.19, Issue 9, pp. 831–843, 2004

[19] Cortès , Rodrìguez-Roda , Sanchez-Marrè, Comas, Cortès, Poch, *DAI-DEPUR: An Environmental Decision Support System for control and supervision of Municipal Waste Water Treatment Plants*, ECAI 2002: 15th European Conference on Artificial Intelligence, July 21-26 2002, Lyon, France, pp. 603-607, 2002

[20] Sànchez-Marrè , Snchez , Cortés , Lafuente , R-Roda , Poch, *DAI-DEPUR: an integrated and distributed architecture for wastewater treatment,* Art. Intell. Engng, Vol.1, pp. 275–285, 1996

[21] Barreteau, Bousquet, *SHADOC: A multiagent model to tackle viability of irrigated systems,* Annals of Operations Research 94, pp. 139-162, 2000

[22] Becu, Walker, Barreteau, Page, *Agent based simulation of a small catchment water management in Northern Thailand: Description of the CATCHSCAPE model,* Ecological Modelling, Vol. 170, pp. 319-331*,* 2003

[23] Becua, Perezb, Barreteauc, Walkerd, *How Bad Isn't the Agent-Based Model CATCHSCAPE?,* Proceedings of the First Biennial Meeting of the International Environmental Modelling and Software Society on Integrated Assessment and Decision Support, Vol. 2, SEA, Como, 2002

[24] Loibl, Toetzer, *Modeling growth and densification processes in suburban regions-simulation of landscape transition with spatial agents,* Environmental Modelling and Software, Vol.18, pp. 553-563 *,* 2003

[25] Mathevet, Bousquet, Page, Antona, *Agent-based simulations of interactions between duck population, farming decisions and leasing of hunting rights in the Camargue (Southern France)*, Ecological modelling, Vol. 165, pp.107-126, 2003

[26] Hoffman, *Construction for the Solutions of the m Queens Problem. Mathematics Magazine*, Vol. 20, Issue 2, pp. 66-72, 1969

[27] Rossi, van Beek, Walsh, *Handbook of Constraint Programming,* Elsevier Sciencem, 2006

[28] *Istituto Nazionale di Fisica nucleare, Sezione di Ferrara*, http://www.fe.infn.it,

[29] Petcu, Faltings, *A Scalable Method for Multiagent Constraint Optimization*, In International Joint Conference on Artificial Intelligence, 2005

[30] Petcu, Faltings, *A Value Ordering Heuristic for Local Search in Distributed Resource Allocation*, Recent Advances in Constraints: Joint ERCIM/CologNet International Workshop on Constraint Solving and Constraint Logic Programming, Springer, pp. 86-97, 2004

[31] Dechter, *Constraint Processing*, Elsevier Science, 2003

[32] Ottens, *Coordinating Agent Plans Through Distributed Constraint Optimization,* In Proceedings of the Multi Agent Planning Workshop - ICAPS 2008, 2008

[33] Hirayama, Yokoo, *Distributed partial constraint satisfaction problem*, G. Smolka Ed., Principles and Practice of Constraint Programming, 1997

[34] *Branch and Bound Methods*, http://www.stanford.edu

[35] Maheswaran, Pearce, Tambe, *Distributed Algorithms for DCOP: A Graphical-Game-Based Approach*, In Proceedings of ISCA PDCSISCA, pp.432-439, 2004

[36] Maheswaran, Bas, *Decentralized network resource allocation as a repeated noncooperative market game,* In Proceedings CDC, Orlando, FL, Vol.5, pp. 4565-4570, 2001.

[37] Maheswaran, Bas, *Multi-user flow control as a nash game: Performance of various algorithms,* In Proeedings CDC, Tampa, FL, Vol.1, pp. 1090-1095, 1998

[38] Léauté, Ottens, Szymanek, *FRODO 2.0: An Open-Source Framework for Distributed Constraint Optimization,* In Proceedings of the IJCAI'09 Distributed Constraint Reasoning Workshop (DCR'09), pp. 160–164, Pasadena, California, USA, 2009

[39] Petcu, *FRODO: A FRamework for Open/Distributed constraint Optimization*, Technical Report 2006/001, EPFL, Lausanne (Switzerland), 2006

[40] *FRODO: An Open-Source Framework for Distributed Constraint Optimization*, http://frodo2.sourceforge.net/

[41] *Matlab*, http://www.mathworks.it/products/matlab/

[42] *Modelit - adviesbureau voor wiskundige methoden*, http://www.modelit.nl

[43] *Extensible Markup Language (XML)*, http://www.w3.org/XML/

[44] Roussel Lecoutre, *XML Representation of Constraint Networks Format XCSP 2.1,* Computing Research Repository, 2009

[45] *XCSP 2.1: a format to represent CSP/QCSP/WCSP instances*, http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html