Approcci Multiagente per algoritmi di scheduling

Luca Merli

19 gennaio 2010

Sommario

In questo elaborato verranno presi in analisi diversi approcci alla gestione di algoritmi di scheduling basati su sistemi multiagente. Verrà prima presentato il problema in generale e successivamente sarà eseguita una carrellata sulle soluzioni proposte negli anni inerenti l'argomento

1 Problema di scheduling - Formulazione e Rappresentazione

I problemi di scheduling emergono in svariati settori, tra loro eterogenei: dalla corretta gestione di sistemi di produzione, alla elaborazione di processi, dalla gestione di risorse in ambito industriale alle strategie di gestione militare. Nonostante questa diversità, la formulazione di questi problemi è comune: si tratta infatti di trovare la giusta sequenza di azioni per ottimizzare, a seconda dell'ambito, le risorse da elaborare mappandole sulle risorse computazionali disponibili, nel rispetto di vincoli temporali e relazionali. La rappresentazione di questi problemi è possibile mediante la rappresentazione ad albero: sia $G =< T, E >$ il grafo composto da T nodi e E archi, è possibile associare

a T le attività da schedulare e a E i vincoli relazionali, che possono essere pesati o meno. Un arco (T_i, T_j) indica che T_j non può iniziare prima che T_i sia completato e che T_i invia i propri risultati a T_j. Associando un costo ad un arco, si può determinare il cammino su un grafo di costo minimo che visiti tutti i nodi, detto cammino Hamiltoniano, che ottimizza lo scheduling delle attività rappresentate mediante il grafo. A differenza dei sistemi tradizionali, i sistemi ad agenti hanno elevati ritardi nel trasferimento dei dati dovuto alla molteplicità di flussi di controllo, collegamenti di rete tra loro eterogenei e una gamma molto ampia di velocità di elaborazione: per tutti questi motivi molte assunzioni dei sistemi tradizionali non sono valide.

Gli algoritmi di scheduling in sistemi ad agenti devono lavorare in ambienti eterogenei in cui:

1. il numero di macchine è limitato
2. le strutture dei task sono generali
3. il ritardo nel trasferimento dei dati è generale
4. non è permessa la duplicazione dei task

per questi motivi i problemi di scheduling multiagente sono problemi NP completi.

2 Approcci

Nella letteratura scientifica disponibile sono presenti numerosi approcci al problema. Tra le opzioni possibili c'è quella di sviluppare un sistema concentrato funzionante e di replicarlo spalmandolo sugli agenti, oppure a partire dalla nozione di agente per raggiungere sistemi autoorganizzanti in risposta alla variazione dei valori di ingresso. Il termine di paragone di questi algoritmi è la versione concentrata dello stesso, preso come upper bound per le valutazioni.

2.1 Dal centralizzato al distribuito

Un primo approccio ai problemi di scheduling multiagente viene fornito da Rong Xie, Daniela Rus e Cliff Stein [1]. Si parte dal componente chiave di ogni

[1] Scheduling Multi-Task Multi Agent Systems, Dartmouth college Hanover - 2001

sistema ad agenti pratico, ossia l'*agent scheduling* e dal controllo di come gli agenti accendano alle risorse. Per sistemi multitask multiagente occorre valutare correttamente il tradeoff tra la quantità di parallelismi utilizzati nelle applicazioni e la quantità di overhead di dati da trasferire incontrata. A partire dalle definizioni del problema generale, si aggiungono $Pred(T_i)$ come lista di task che precedono il task T_i, $M = \sum_{k=1}^{M} M_k$ l'insieme delle macchine in una rete completamente connessa, $r(M_i, M_k)$ il data transfer rate tra la macchina i e la macchina k (∞ se i=k) e $p(T_i, M_k)$ il tempo di esecuzione del processo T_i sulla macchina M_k. L'obiettivo è di trovare una funzione di assegnamento $M : T \rightarrow M$ e un set di tempi di inizializzazione $st(T_i), i = 1 \cdots n$ in cui ogni task T_i sia schedulato per essere processato su una macchina $M(T_i)$ e inizi al tempo $st(T_i)$ in modo che tutti i vincoli di precedenza siano rispettati e che sia minimizzato il tempo totale di scheduling $C_{max} = maxft(T_i) = max(st(T_i) + p(T_i, M(T_i)), 1 < i < n$. A questo punto viene elaborato un algoritmo centralizzato in cui un agente composto da n task viene elaborato in n step (one-to-one). Ad ogni step l, il task T_i è detto *ready* se non è stato ancora schedulato, ma lo sono stati tutti i suoi predecessori $Pred(T_i)$. Sia $DA(T_i, M_k)$ il data available time del task T_i sulla macchina M_k e sia $MA(M_k)$ il tempo che ogni macchina impiega per processare tutti i suoi task e quindi tornare disponibile. La versione base dell'algoritmo prevede l'inizializzazione del set di task ready R e del set di task in ingresso T. Ad ogni step l si cercano il task ready T_i e la macchina M_k che minimizzano:

$$max \quad DA(T_i, M_k)) + \frac{c * p(T_i, M_k)}{maxp(T_j, M_j)}$$

.

In caso di una formulazione non corretta dell'albero da esplorare questo algoritmo ha performance di scarso livello. Per risolvere questo problema si lavora anche sull'albero inverso, ottenuto invertendo la direzione degli archi: in questa maniera l'albero e l'albero inverso hanno lo stesso scheduling minimo. L'algoritmo prende il nome di *Forward Backward Schedulinge* funziona nel seguente modo: si esegue l'algoritmo versione base e sull'albero diretto e si ottiene lo scheduling S, successivamente si esegue l'algoritmo base anche sull'albero inverso, ottenendo lo scheduling S'. Se $C_{max}(S) < C_{max}(S')$, allora viene generato S, altrimenti S'.

Se l'albero contiene sia alberi entrati ed uscenti mal formati, viene proposta una versione dell'algoritmo detta *PFB: Partial Forward Backward.* Si esegue l'algoritmo versione base sul problema originale $G =< T, E >$, ottenendo S; sia S=S'; $\forall T_j \in T$ si inverte la parte di S' per i task che iniziano dopo $max \quad ft(T_i)$ in S, ottenendo S_1; a partire da S_1 si esegue l'algoritmo

1 sull'albero inverso per ottenere S_2; ottenuto S_2 si inverte ottenendo S": se $C_{max}(S'') < C_{max}(S')$, allora S"=S' e viene generato S'.

Definito questo approccio centralizzato, si passa all'ambito distribuito. In un sistema mobile ad agenti, gli agenti multi-task arrivano nel corso del tempo. Viene proposto un framework in cui si assegna per ogni agente di questo tipo il proprio scheduler, ossia un manager di risorse che utilizza la versione distribuita dell'algoritmo (sotto riportata). In questo approccio, un task è detto *ready* se tutti i suoi predecessori hanno iniziato l'esecuzione.

2.1.1 Algoritmo 4: Algoritmo Distribuito

Si esegue l'algoritmo PFB sull'agente multi-task per ottenere l'output S_0. Si inizializza R come il set degli entry task in T. Fino a quando tutti i task degli agenti non sono stati schedulati: si aggiorna R e finchè R $\neq \emptyset$

1. Trova la coppia task *ready* T_i* e macchina M_k* che minimizzano

$$max \quad DA(T_i, M_k)) + \frac{c * p(T_i, M_k)}{maxp(T_j, M_j)}$$

   tra tutte le coppie di task ready $T_i \in R$ e macchine $M_k \in M$;

2. Se l'ACCR[2] dell'albero diretto DAG è maggiore di λ allora: $\forall$ task ready già schedulato A che abbia un successore comune con T_i*, che sia assegnato alla stessa macchina di T_i* in S_0 e che il minimo dei ritardi dovuti al trasferimento dati tra A e T_i* al loro successore comune sia α volte più grande del massimo tra i tempi di esecuzione standard di A e T_i*, imposta M_k* come macchina che ha assegnato A;

3. Schedula il task T_i* sulla macchina M_k*

2.1.2 Valutazione algoritmi

Utilizzando l'algoritmo distribuito basato su scheduling PFB, con valori pari a λ=4 e α=2, si nota che le performance sono significanti quando l'ACCR è ampio, quindi in risposta a comunicazioni intensive dei sistemi multi agenti.

L'algoritmo distribuito assume che lo scheduler abbia una conoscenza totale dell'agente multi-task che deve essere schedulato e della informazione totale della rete, aspetto non realistico in modelli reali. In molti casi è possibile infatti fornire solo delle stime dei parametri: per questo è importante

[2]Avarage Communication-to-Computation Ratio

valutare la tolleranza dell'algoritmo di scheduling distribuito nei confronti della stima degli errori dei parametri. Vengono scelti 3 parametri: il tempo di esecuzione standard, la dimensione di trasferimento dati tra i task e il transfer rate dei collegamenti di comunicazione. Per ognuno di essi si forniscono 3 range di tolleranza e si definisce il *degradation ratio*, ossia il rapporto tra la somma dei tempi di esecuzione con parametri corretti e quella con parametri stimati. Se il rapporto è molto al di sotto di 1.0 allora significa che l'algoritmo è molto sensibile alla variazione dei parametri. Sperimentalmente si osserva che l'algoritmo è più sensibile alla dimensione dei dati da trasferire e alle comunicazioni piuttosto che per i tempi di esecuzione dei task. Il degradation rate peggiore è del 20%.

3 Scheduling MultiAgente con Programmazione a Vincoli

Un altro aspetto rintracciabile nella letteratura scientifica è quello della programmazione a vincoli; in particolare Willem-Jan van Hoeve, Carla P.Gomes, Bart Selman e Michele Lombardi[3] ne forniscono un modello.

3.1 Introduzione

L'algoritmo che viene presentato ha l'intenzione di risolvere problemi di scheduling multiagente centralizzati e deterministici. Dal momento che esistono interdipendenze tra i task, gli agenti devono comunicare tra loro qualunque proposta di cambiamento. L'obiettivo principale è quello di garantire al sistema ottimalità e efficenza computazionale. I punti di forza dell'algoritmo sono la possibilità di fornire specifiche al problema con un ricco linguaggio di modellazione, una rappresentazione molto prossima alla realtà del problema, la possibilità di fornire un'euristica di ricerca molto dettagliata, in base al problema da affrontare e l'implementazione di una ottimizzazione dei vincoli, basandosi su relazioni di programmazione lineare

[3] Optimal Multi Agent Scheduling with Constraint Programming, 2007

3.2 Descrizione del problema

Il problema consiste in un set di agenti che devono eseguire certi metodi. Subentra il concetto di *qualità*: ogni metodo eseguito aumenta la qualità della funzione obiettivo gerarchica, tuttavia ci sono da rispettare le relazioni di interdipendenza e i vincoli temporali. Viene utilizzato il linguaggio TAEMS[4] nella versione cTAEMS, per problemi di coordinazione. Ogni agente A ha un metodo i, ed ogni agente ha l'obbligo di eseguire almeno un metodo per ogni intervallo di tempo: ogni metodo eseguito genera una certa qualità $Q[i]$ e la sua esecuzione ha durata $D[i]$. Un task invece non è di proprietà dell'agente, ma sere per accumulare qualità mediante i suoi subtask, valutabile mediante una funzione di accumulazione di qualità (QAF). Sia i task che i metodi hanno una finestra temporale in cui è possibile la loro esecuzione; inoltre possono esistere relazioni di precedenza tra i task e i metodi.

Esistono diversi tipi di approcci: il Markov Decision Process per problemi cTAEMS non deterministici, la formulazione di Szekely che prevede la combinazione di metodi euristici per la soluzione e quella di Smith come semplici reti temporali: tuttavia tutti questi metodi non forniscono una soluzione ottima ai problemi cTAEMS deterministici e non. Il modello proposto consiste in 2 set di variabili decisionali; il set delle variabili di assegnamento x_i e il set delle variabili di inizio esecuzione $start_i$, per ogni metodo i. Si applica una ricerca depth-first in due fasi, la prima è la fase di selezione, la seconda è quella di scheduling vero e proprio. Nella fase di selezione vengono assegnate tutte le variabili in modo greedy, partendo dalla variabile x_i con qualità $Q[i]$ minore. Applicando la strategia depth-first si inizia da un scheduling vuoto fino a raggiungere la qualità maggiore possibile. Nella fase di scheduling invece si assegnano le variabili di inizio esecuzione, iniziando da quella con il dominio minore.

Se esistono parti di problema totalmente indipendenti tra loro, queste si possono risolvere in maniera indipendente. Tuttavia tutte le variabili decisionali sono tra loro connesse mediante la funzione obiettivo e i vincoli di qualità: per ovviare al problema in questa presentazione si deve decomporre la funzione obiettivo, processo che richiede estrema attenzione.

3.3 Risultati

Gli aspetti da valutare sono svariati, ma in particolare ci si focalizza sulla ampiezza della finestra di esecuzione, sul numero sul tipo delle relazioni di

[4] Task Analisys, Environment Modeling and Simulation, Horling 1999

precedenza e sulle diverse funzioni di accumulazione della qualità. Nel test dell'algoritmo viene utilizzato un set di 2550 istanze, ognuna delle quali contenente da 8 a 64 metodi e dai 2 ai 9 agenti: applicando la decomposizione del problema, e applicando l'Unary Resource[5] , ottenendo i migliori risultati per la soluzione dei problemi all'ottimalità, migliorando notevolmente le performance degli algoritmi esistenti in materia di problemi cTAEMS.

4 Apprendimento basato su agenti per goal multipli di planning e scheduling

L'apprendimento rinforzato è stato proposto come un metodo efficiente per l'acquisizione di conoscenza dei sistemi multiagente. Tuttavia, la gran parte dei ricercatori applicano l'apprendimento rinforzato, mette il proprio focus sul metodo per ridurre la complessità dovuta alla esistenza di agenti multipli e goal multipli: l'aspetto che dovrebbe essere preso in considerazione è quello dell'emergere di un comportamento cooperativo in domini multiagente, come suggerito da Sachiyo Arai, Katia Sycara e Terry R.Payne[6] . Nel loro elaborato viene utliizzato il Pursuit Game esteso in cui esistono prede multiple e cacciatori multipli. Ogni cacciatore è un learning agent, mentre la preda non ha capacità di apprendimento e si muove a caso nell'ambiente, che è composto da celle triangolari, in modo da ridurre la dimensione dello stato dello spazio per una cattura (con un triangolo occorrono solamente 3 cacciatori per bloccare una preda). Un cacciatore può conoscere la posizione di una preda solo quando la preda è a portata della sua vista , che è suddivisa in 15 aree e in cui ogni area rappresenta il suo stato in termini di {posto vacante, esistenza di un cacciatore, esistenza di una preda}. L'obiettivo finale di un agente è di catturare tutte le prede nell'ambiente. Sotto queste condizioni, ogni cacciatore non solo deve trovare il cammino per raggiungere la preda, ma deve anche decidere quale preda debba essere in comune con gli altri cacciatori. Per trovare il path per raggiungere la preda, i cacciatori devono

[5] l'insieme dei metodi che non si sovrappongono, composto da tempi di inizio, tempi di fine e variabile richiesta da ogni metodo

[6] Multi-agent Reinforcement learning for Planning and Scheduling Multiple Goals, Robotic Institute Carnegie Mellon University

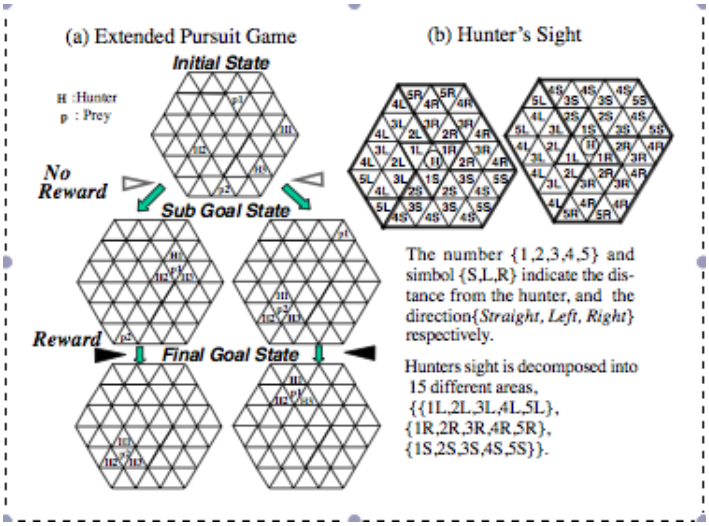

Figura 1: Pursuit Game

avvicinarsi alla preda, mentre per quanto riguarda la decisione dell'obiettivo, è necessaria una cooperazione per creare una sequenza corretta per tutti i cacciatori: per questo motivo è necessario prendere coscienza del *perceptual aliasing problem* e dell'apprendimento concorrente degli agenti.

4.1 Approccio Profit-sharing

Viene utilizzato questo tipo di approccio in quanto in grado di gestire correttamente i problemi di inconsistenza legati all'esistenza di agenti multipli, come l'apprendimento concorrente e l'aliasing percettivo. Inoltre può salvare uno spazio di memoria richiesto in quanto non deve mantenere le azioni ammissibili degli agenti e l'intero spazio degli stati che l'agente ha visitato.

Quando un agente osserva lo stato corrente σ_t, controlla nella sua lookup table di ricerca il matching con il proprio stato σ_t e prende la decisione in base al suo set di azioni $A_t = \{$ left, right, straight, stay$\}$, disponibili all'istante t. L'azione viene scelta mediante una roulette di selezione, con metodi soft-greedy, in cui la selezione di una azione è proporzionale al proprio peso corrente. Questo metodo di selezione permettono al cacciatore un comportamento valido in un sistema stocastico. Successivamente il cacciatore genera la propria azione scelta a_t, e controlla se l'azione è positiva o meno (in termini

di ricerca della preda). Se l'azione non porta a trovare alcuna preda, il cacciatore salva la coppia stato azione (σ_t, a_t) nella sua memoria come una *regola* e continua finché non trova la preda, indicata come R. Si definisce episodio il periodo che intercorre tra l'inizio della ricerca e l'ottenimento di R. Una volta ottenuta R, il cacciatore rinforza le regole contenute nella sua memoria con una funzione di assegnamento $f(R, t)$ che soddisfa il Teorema di Razionalità[7] come ad esempio la funzione di decrescita geometrica $f = R(1/(L-1))^{T-t}$, con T= tempo in cui si trova il goal e t=0 tempo iniziale, L=numero delle azioni disponibili $\forall$ step. Questo teorema afferma che R non dovrebbe essere assegnata ad una azione inconsistente che provoca il movimento in loop di un agente, a favore di una azione che fa muovere l'agente senza loop.

4.2 Sperimentazione e Risultati

Per la sperimentazione viene utilizzata la funzione $f = R(0.3)^{T-t}$ per assegnare R ad ogni coppia stato-azione all'interno della memoria. In ogni condizione di sperimentazione, i cacciatori imparano 1000000 di episodi come prova e la lookup table di ogni cacciatore è resettata dopo ogni prova. Vengono iterate 10 prove per valutare la media della risposta e la deviazione standard. Per valutare la performance dei cacciatori senza conoscenza globale, si prende in considerazione la condizione base in cui un singolo agente globale scheluda la sequenza per catturare una preda. In questo caso all'agente globale è data l'informazione inerente la posizione di tutte le prede e di tutti i cacciatori, per cui scegli la preda in base a

$$Prey_j = argmin_{j \in prey} \sum distance(Prey_j, Hunter_i)$$

Tutti i cacciatori si focalizzano sulla preda target, che l'agente globale decide di catturare e non considera le altre prede: in questo caso un cacciatore ignora tutte le prede se non quella target, anche se sono nella sua visuale. Dopo aver catturato la prima preda, l'agente globale decide l'obiettivo successivo e continua fino alla cattura di tutte le prede.

Come si desume dal grafico (Figura 2), la performance del sistema con un agente globale è migliore rispetto a quella senza agente globale, perché l'obiettivo dei cacciatori viene sempre preso all'interno del loro ambiente, che ha dimensione costante. Tuttavia, il numero di step necessari per catturare una

[7] Miyazaki K. , YamamuraM., and KobayashiS. , 1994. On the Rationality of Profit Sharing in Reinforcement Learning

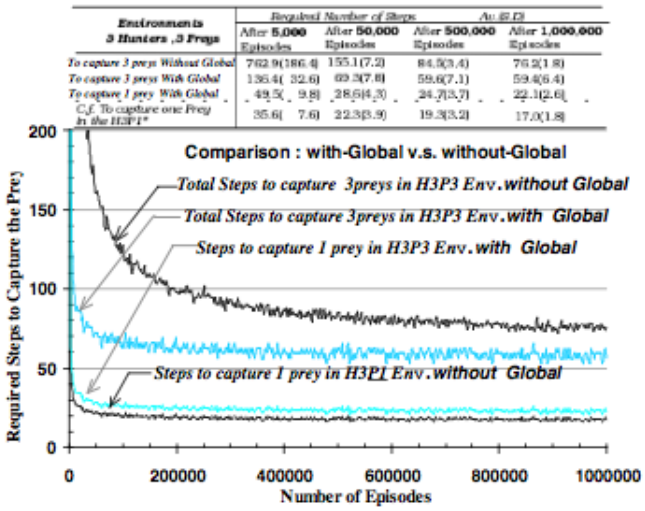

Figura 2: Risultati sperimentazione

singola preda è minore nella versione senza agente globale, presumibilmente per il fatto che l'agente globale, forzando la scelta della preda, porta tutti i cacciatori in aliasing di percezione impedendo a loro di muoversi liberamente e soprattutto verso la preda più vicina.

5 Un approccio ai sistemi di scheduling Multi-Agente usando una coalizione di formazione [8]

In questo elaborato, viene affrontato il tema dello scheduling basato su sistemi multiagenti analizzando due meccanismi chiamati nemawashi (manovrazione dietro le scene) e settoku (persuasione) e mostrando gli effetti di questi meccanismi in sistemi di scheduling pratici. Nei giochi basati sulla coalizione, un agente calcola la propria utilità mediante una funzione caratteristica, e partecipa alla coalizione incrementando la propria utilità. Si definisce fun-

[8] Takayuki Ito, Toramatsu Shintai

zione caratteristica $v(S)$ basata sulla dimensione degli eventi $|S|$, sugli eventi privati U_e, sulle relazioni umane H e sulle relazioni tra gli agenti A

$$v(S) = |S| - U_e + H + A$$

Un sistema di scheduling viene usato per creare uno scheduling coerente per tutti i membri del sistema: ogni agente viene assegnato ad un utente e conosce lo scheduling privato del proprio utente. Sono gli agenti che collaborano per la creazione della coalizione. Nelle applicazioni pratiche, è difficile raggiungere un consenso solo con protocolli di negoziazione convenzionali, per questo nascono i due meccanismi nemawashi e settoku.

5.1 Processo di negoziazione

Nei processi di negoziazione, le preferenze degli agenti vengono immagazzinate in un protocollo[9]. Per raggiungere consenso in modo efficace, il processo di negoziazione viene diviso in due parti: negoziazione fondamentale e meccanismo soft settoku. La prima fase viene eseguita da agenti free(che non hanno eventi privati schedulati), la seconda fase invece è eseguita da agenti busy. Nella negoziazione fondamentale, gli agenti free dichiarano le proprie preferenze basandosi sulla funzione caratteristica (considerando solo H e A); successivamente, basandosi su queste preferenze, gli agenti busy dichiarano le proprie preferenze utilizzando la funzione $v(S)$ nel meccanismo settoku.

Nel processo del meccanismo nemawashi, un agente immagazzina informazioni inerenti le preferenze di un altro agente per sequenze concorrenti. In base alle informazioni disponibili, l'agente può restringere le preferenze dell'altro agente. Quando le preferenze di un agente non tendono a raggiungere un consenso, un meccanismo settoku forzato obbliga l'agente in minoranza a deselezionare le proprie sequenze preferite e deve cambiare preferenze, anche con l'intervento di agenti più influenti nel processo di negoziazione.

6 Conclusioni

Dopo aver presentato questi approcci a problemi di scheduling, è giusto considerare l'efficienza di un utilizzo di MAS a discapito di un controllo centralizzato. Nello scheduling, spesso, i task che devono essere eseguiti sono già decisi

[9] Optimal Task Allocation by Circulation Board Protocol

e nella pratica, lo scheduling tende a focalizzarsi su algoritmi per problemi di dominio specifico. Nella pratica, i sistemi multiagente possono modellarsi in numerosi ambienti (dall'industria bellica a quella spaziale, dall'informatica alla robotica e così via) e spesso risulta difficile motivare un approccio multi agente in quanto la versione centralizzata risulta più efficiente (si veda il caso del Pursuit Game): è necessario dunque codificare metodi per comprendere quando e come decentralizzare i sistemi. Inoltre vi è da considerare che lo scheduling molto spesso collassa in problemi di planning, che in ottica multiagente significa supportare azioni concorrenti e adattare algoritmi di planning flat agli agenti, quindi il planning multiagente non può essere considerato solo come una semplice soluzione di problemi di planning, ma occorre comprendere anche come gli agenti debbano comportarsi e interagire, date le proprie capacità organizzative.