Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Specialistica in
Ingegneria dell'Automazione

Task Assignment Distribuito su Rete Multi-Agente via Constraint Consensus

Laureando:
Mirko Fabbro

Relatore:
Ch.mo Prof. Angelo
Cenedese

Anno accademico 2013/2014

Sommario

I sistemi multi-agente pongono spesso problemi di natura decisionale e combinatoria, che sono tradizionalmente risolti tramite approcci centralizzati. Uno di questi problemi, che spesso si pone per scenari di agenti autonomi in grado di svolgere task eterogenei nelle tipologie e nelle priorità, è quello di stabilire un'assegnazione tra agenti e task in maniera ottima, ovvero in modo da massimizzare un determinato indice di benefit. Altrettanto frequentemente, si richiede che il sistema presenti buone caratteristiche di flessibilità e robustezza e che gli agenti siano in grado di gestire sè stessi in maniera autonoma.

La prima parte del presente lavoro indaga la possibilità di utilizzare strumenti risolutivi classici per il problema dell'assegnazione, quali il simplesso o il cosiddetto metodo ungherese, in maniera distribuita, appoggiandosi alla teoria dell'ottimizzazione astratta. La strategia (Constraint Consensus) prevede che ogni agente, a partire da una conoscenza parziale dei task disponibili e dello stato degli altri agenti, sia in grado, comunicando con questi ultimi, di ricostruire una rappresentazione minima del problema di partenza. A tale rappresentazione corrisponde una soluzione ottima, condivisa da tutti i nodi, del problema globale di partenza.

Una seconda parte fornisce alcuni esempi di applicazione dell'algoritmo ad un sistema di smart-camera con compiti di videosorveglianza distribuita. Infine sono riportate alcune simulazioni numeriche allo scopo di testare le prestazioni e la scalabilità dell'approccio proposto.

Ringraziamenti

Ora che sta per concludersi ufficialmente il mio percorso universitario, desidero, ringraziare tutti coloro che hanno contribuito, direttamente o indirettamente, alla stesura della tesi e chi mi sono stati vicini in questi anni. A loro va la mia gratitudine, a me spetta la responsabilità per ogni eventuale errore o imprecisione.

Ringrazio anzitutto il Professor Angelo Cenedese, che mi ha seguito nella redazione della tesi con cortesia e disponibilità, fornendomi preziosi spunti. Grazie anche al Professor Luca Schenato, per aver gettato le basi di questo lavoro nel corso di Progettazione dei Sistemi di Controllo.

Un enorme grazie ai miei genitori, per l'affetto e la fiducia che mi hanno sempre dimostrato, per avermi sostenuto dopo gli insuccessi, non meno di quanto hanno gioito con me dei successi. Grazie per il loro supporto. Se sono giunto a questo punto lo devo a loro.

Grazie ai parenti, per aver atteso con pazienza questo giorno, e agli amici, vicini e lontani, per le ore spensierate, lontano dal cruccio dello studio.

Infine grazie a Francesca: per essermi stata accanto in tutto questo tempo, per aver creduto in me, anche nei momenti in cui ero io stesso il primo degli scettici. Il mio scoglio nel mare in tempesta. Grazie.

Indice

1 Introduzione

I problemi di ottimizzazione combinatoria hanno ricevuto grande interesse a partire, a grandi linee, da metà del secolo scorso.

Come scegliere nel miglior modo possibile, tra un numero anche molto elevato di possibili alternative, è evidentemente un problema che si pone continuamente ed in tutti i campi teorici ed applicativi, non solo in ambito ingegneristico.

Diverse applicazioni, anche nell'ambito dei controlli, fanno sorgere problematiche di carattere decisionale la suddivisione di compiti, lo scheduling di attività, la logistica, spesso in forma dinamica.

Tali problemi sono stati tradizionalmente formulati e risolti, in modo sempre più efficiente, secondo modelli ed algoritmi centralizzati: definito uno spazio delle soluzioni $\mathcal{X}$, di un'opportuna metrica ϕ su di esso definita e disponendo di tutta l'informazione necessaria sull'istanza del problema, si applica una strategia adeguata con lo scopo di individuare un particolare elemento appartenente a $\mathcal{X}$, considerato ottimo ai sensi di ϕ.

1.0.1 Sistemi multi-agente

In tempi recenti si è assistito ad un crescente interesse verso i sistemi distribuiti e multi-agente, da parte di diverse discipline, che spaziano dalla tecnologia dell'informazione all'economia, dall'ingegneria alle scienze sociali, includendo campi come la biologia, le scienze naturali e la fisica.

In particolare, le reti multi-agente prendono le mosse da una metafora sociale, per cui ogni agente (o nodo della rete) è completamente autonomo nelle sue decisioni, che vengono prese sulla base di proprie percezioni (misure) dell'ambiente esterno e di informazioni comunicate dagli altri agenti della rete, facendo un marginale o alcun ricorso a strutture di coordinazione centralizzate.

L'obiettivo che perseguito nel design di questi sistemi è la sintesi di una logica, una legge di controllo, che, se applicata individualmente dagli agenti, dia luogo a comportamenti cooperativi (o competitivi) in grado di risolvere problemi complessi formulati in modo globale.

Tra i principali benefici ottenibili con questo tipo di approccio, tutti o in parte, si possono citare:

- **Distribuzione del carico computazionale**. Compiti complessi, non affrontabili computazionalmente dai singoli agenti, sono suddivisi tra diversi nodi, che si trovano così, individualmente, a manipolare e risolvere problemi di ordine ridotto.
- **Distribuzione dell'informazione**. La rappresentazione del problema globale può a volte essere di dimensioni molto elevate, problematiche da memorizzare e gestire da una singola entità. La distribuzione permette di suddividere l'informazione tra i nodi della rete senza pregiudicare la risolvibilità.
- **Ridondanza ed affidabilità**. Sia le informazioni che le competenze dei singoli agenti possono essere più di una mera partizione della loro unione. Questo aspetto permette di garantire, entro certi margini, che eventi di guasto, perdita di connettività o informazione da parte di un limitato numero di nodi non impatti sull'informazione globale e la capacità del sistema nel suo complesso di assolvere alle funzioni preposte.
- **Flessibilità ed estensibilità**. Esiste sempre la possibilità, nel caso le premesse di progetto abbiano subito variazioni (per esempio se sono richieste funzionalità eterogenee o la capacità di gestire istanze di dimensione superiore) di incrementare la rete con nuovi agenti mantenendo i vecchi nodi, senza dover ridefinire tutto il sistema.

1.0.2 Problemi di assegnazione

Questa tesi affronta, in particolare, il problema dell'assegnazione (task assignment problem). Dato un insieme di *agenti* ed una lista di compiti da eseguire (*task*), si vuole individuare una strategia ottima per affidare ad ogni agente uno dei task da svolgere. La risoluzione è, spesso, complicata dal fatto che ogni agente è compatibile con (è in grado si svolgere) un insieme limitato di task, piuttosto che con tutti i task presenti nella lista. Inoltre tra le possibili combinazioni derivanti dal vincolo appena espresso, si desidera identificarne una che ottimizzi un qualche indice opportunamente definito sulle possibili coppie task-agente. Per le ragioni discusse in precedenza, infine, la strategia risolutiva deve applicarsi convenientemente ad un sistema distribuito multi-agente.

1.1 Panoramica sulla letteratura.

La letteratura sull'argomento è piuttosto vasta ed affronta il problema con tecniche eterogenee. Per una trattazione generica sul problema dell'assegnazione, le sue varianti ed algoritmi risolutivi si segnalano, tra i tanti esempi, [5] e [16].

Più in dettaglio, per quanto riguarda l'assegnazione con metrica lineare (*linear assignment problem*) si distinguono le seguenti tecniche.

- **Programmazione lineare intera**. Un primo approccio modellistico rappresenta il problema dell'assegnazione come problema di programmazione lineare intera [13], cui è possibile applicare l'algoritmo generico del simplesso [6] in grado di ottenere una soluzione ottima in tempo esponenziale, nel caso pessimo.

- **Reti di flusso**. Un approccio alternativo si ottiene osservando che il problema dell'assegnazione in forma di grafo bipartito e riconducibile ad un problema di flusso, aggiungendo due nodi terminali (*source* e *sink*) e un certo numero di archi di *portata* opportuna. In questa forma sono direttamente applicabili svariati algoritmi di flusso massimo con complessità polinomiale, come l'algoritmo di Ford-Fulkerson [8].

- **Metodo ungherese**. Uno dei primi metodi risolutivi con complessità polinomiale (tuttora tra i più efficienti) definito appositamente per il problema dell'assegnazione, introdotto da Kuhn [12] e successivamente Munkres [15], funziona come segue: partendo da un'assegnazione ottima incompleta (con un numero ridotto di agenti), viene incrementata la cardinalità della stessa in maniera monotona sino ad ottenere la massima cardinalità e quindi l'ottimalità per il problema globale.

- **Stable marriage problem**. La rappresentazione prende spunto da modelli sociali come uomini e donne o datori di lavoro ed impiegati. Ad ogni soggetto viene attribuita una lista di preferenze relativa elementi dell'altro gruppo e si vuole ottenere un assegnazione completa in cui sia massimizzata la soddisfazione generale dei partecipanti [10],[14].
  Mantenendo la metafora uomini-donne, l'algoritmo dovuto a Gale e Shapley [9] prevede che un uomo scelga la donna in cima alla sua lista di preferenze e, in caso questa sia libera, la coppia si sposa, se la donna è invece già impegnata può decidere di lasciare l'attuale compagno per il nuovo pretendente o meno, coerentemente con le sue preferenze. Se un

uomo rimane non accoppiato passerà a valutare la donna successiva in lista e così via. In tempo finito si ottiene una serie di matrimoni detti "stabili", che massimizzano la soddisfazione di uomini e donne.

- **Algoritmi market-based**. Questo approccio, da cui discende un gran numero di implementazioni e varianti, prende spunto da dinamiche di mercato [1],[2], per cui ogni agente attribuisce un certo *valore* ai task e ad ogni task è assegnato un *prezzo* da parte di un *banditore*, secondo un meccanismo d'asta. Ogni agente, perseguendo una logica di massimo profitto, cerca di essere assegnato al task che massimizza la differenza valore-costo. Si tratta di una competizione tra gli agenti, che sono in grado di effettuare "rilanci" (di valore minimo fissato) sul prezzo per rendere meno appetibile il task preferito ai concorrenti, analogamente a quanto accade nelle comuni aste. Il processo termina con il banditore che definisce un'assegnazione globale quasi-ottima. È possibile agire sul valore della puntata minima per accelerare l'asta a scapito della precisione sull'assegnazione finale.

- **Ottimizzazione astratta e Constraint Consensus**. La teoria dell'ottimizzazione astratta si applica ad alcuni problemi combinatori (mo non solo) definendoli come un insieme di *vincoli*, che circoscrivono le soluzioni ammissibili, ed una metrica definita sul suo insieme delle parti, secondo cui determinare l'ottimalità delle stesse. Gli algoritmi risolutivi si occupano di identificare un sottoinsieme minimo, una *base*, dell'insieme dei vincoli, che, insieme alla metrica, determina univocamente la soluzione ottima. L'algoritmo di Constraint Consensus propone, per un sistema distribuito, una strategia di consenso sui vincoli che si conclude con il calcolo di una base ottima e condivisa per il problema di ottimizzazione astratta.

1.2 Struttura dell'elaborato

Si chiude l'introduzione presentando brevemente il contenuto dei capitoli successivi.

- **Capitolo 2 - Teoria**
  Si introduce un modello matematico formale per il problema dell'assegnazione e si definiscono alcune ipotesi di lavoro. Sono riportati brevemente il metodo del simplesso ed il metodo ungherese come soluzioni al problema centralizzato e sono descritti i risultati teorici alla base della teoria dell'ottimizzazione astratta e del constraint consensus. Infine è presentato l'algoritmo di Contraint Consensus distribuito e sono discusse le peculiarità di una sua applicazione all'assignment problem.

- **Capitolo 3 - Algoritmi ed implementazioni**
  È descritta in dettaglio la struttura della rete, degli agenti e dei task e se ne propone una rappresentazione *object-based*. Sono riportati in dettaglio l'implementazione del simplesso e l'algoritmo di Kuhn-Munkres nelle varianti adatte all'applicazione del Constraint Consensus. Si accenna all'estensione della comunicazione per la gestione ottima di assegnazioni dinamiche o situazioni di guasto.

- **Capitolo 4 - Scenario di esempio: video sorveglianza**
  Si presenta un semplice esempio applicativo per quanto introdotto nei capitoli precedenti: una rete di smart-camera con compiti di videosorveglianza. Viene presentata in dettaglio l'evoluzione dello stato della rete al verificarsi di eventi dinamici, come nascita o morte di task o situazioni di guasto improvviso.

- **Capitolo 5 - Simulazioni**
  Sono riportate alcune simulazioni numeriche su istanze casuali dell'assignment problem, utili ad indagare sia prestazioni e debolezze degli algoritmi sotto l'aspetto della velocità di convergenza, del carico di rete e della scalabilità, sia l'influenza di scelte progettuali come ad esempio il grado di connessione nel grafo di comunicazione.

- **Capitolo 6 - Conclusioni**
  Breve riepilogo di quanto svolto, con alcune considerazioni e qualche spunto per futuri approfondimenti.

2 Teoria

2.1 Assignment problem

In questa sezione si formalizza il problema dell'assegnazione e si introduce una notazione consistente, ove possibile, per il prosieguo della trattazione. Lo scenario che si vuole indagare è quello in cui un set di N *agenti* (o *nodi*), identificati come $\mathcal{A} = \{a_1, \ldots, a_N\}$ sono chiamati a svolgere una serie di compiti semplici, o *task*, appartenenti ad un insieme $\mathcal{T} = \{t_1, \ldots, t_M\}$, o *pool* dei task.

Per ogni task t_j è definito un valore $p_j \in \mathbb{R}_{\geq 0}$ di *priorità intrinseca*, che rappresenta l'utilità (o l'urgenza) associata all'esecuzione del task.

Tale situazione è ben rappresentata (figura 2.1) in forma di grafo pesato bipartito $\mathcal{G} = (\mathcal{A}, \mathcal{T}, \mathcal{H}, p)$ in cui $\mathcal{A}$ e $\mathcal{T}$ rappresentano gli insiemi disgiunti di agenti e task, $\mathcal{H} \subseteq \mathcal{A} \times \mathcal{T}$ è l'insieme dei possibili accoppiamenti, su cui è definita la funzione di peso $p : \mathcal{H} \to \mathbb{R}_{\geq 0}$.

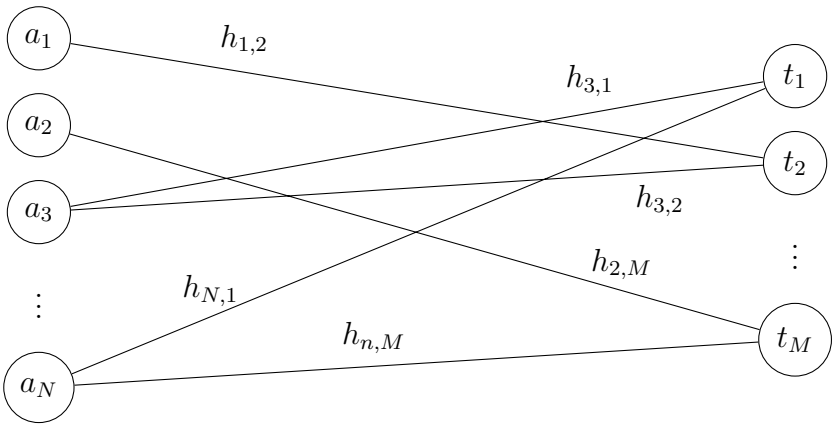


Figura 2.1: Problema dell'assegnazione: rappresentazione come grafo bipartito.

Si osservi che $\mathcal{G}$ rappresenta un'istanza del tutto generica in cui gli archi sono un sottoinsieme proprio del prodotto cartesiano $\mathcal{A} \times \mathcal{T}$, ovvero una parte dei task possono essere eseguiti da un insieme ristretto di agenti.

Definizione 2.1 (Assegnazione)**.** Si dice *assegnazione* una selezione $\mathcal{M}$ di archi appartenenti al grafo $\mathcal{G}$ per cui ogni arco incide su nodi distinti, ovvero

$$\mathcal{M} = \{h_{i,j} \in \mathcal{H} : h_{i1,j1} \neq h_{i2,j2} \implies a_{i1} \neq a_{i2} \wedge t_{j1} \neq t_{j2}\} \tag{2.1}$$

Posto, senza perdita di generalità, $N \leq M$, si parla di *assegnazione di cardinalità massima* se $|\mathcal{M}| = N$.

L'ipotesi $N \leq M$ è sempre verificata, volendo mantenere la distinzione task-agenti, a patto di introdurre in numero adeguato task *idle* fittizi con priorità nulla, a rappresentare gli agenti scarichi.

Proposizione 2.1.
Il numero di possibili assegnazioni, con $H = NM$, è

$$\Omega(N, M) = \sum_{k=0}^{N} \binom{N}{k} \mathcal{D}(M, k) = \sum_{k=0}^{N} \frac{N!M!}{k!(N-k)!(M-k)!} \tag{2.2}$$

Dimostrazione. Si consideri il caso $N \leq M$, senza perdita di generalità. In ogni assegnazione si possono identificare $0 \leq k \leq N$ agenti accoppiati con un task e di conseguenza $N - k$ agenti non occupati. Fissato k, gli effettivi agenti impegnati saranno una permutazione del vettore $v = (1, \ldots, 1, 0, \ldots, 0)$, con k uno e $N - k$ zeri. Le possibili permutazioni distinte di v sono esattamente $\binom{N}{k} = \frac{N!}{k!(N-k)!}$. Per ognuna di queste permutazioni è possibile ottenere (con k agenti occupati) $\mathcal{D}(M, k) = \frac{M!}{(M-k)!}$ disposizioni semplici di M task su k agenti. Sommando tutte le possibilità si ottiene la 2.2. □

Definizione 2.2 (Assegnazione ottima)**.** Un'assegnazione $\mathcal{M}^*$ su $\mathcal{G}$ è detta *ottima* se non esistono altre assegnazioni in cui la somma dei pesi (priorità) degli archi coinvolti è maggiore.

È evidente che, in generale, le soluzioni al problema dell'assegnazione ottima possono essere molteplici ed equivalentemente desiderabili a priori.

Un esempio è il problema, con $N = 3$ e $M = 4$, rappresentato in figura 2.2, dove i pesi degli archi dipendono solo dal nodo t_i cui sono incidenti ed è indicato a fianco al grafo dai valori p_i.

Si osserva che esistono due soluzioni distinte co-ottime; infatti

$$\mathcal{M}_1 = \{(1, 2), (2, 4), (3, 3)\} \text{ e } \mathcal{M}_2 = \{(1, 3), (2, 2), (3, 4)\}$$

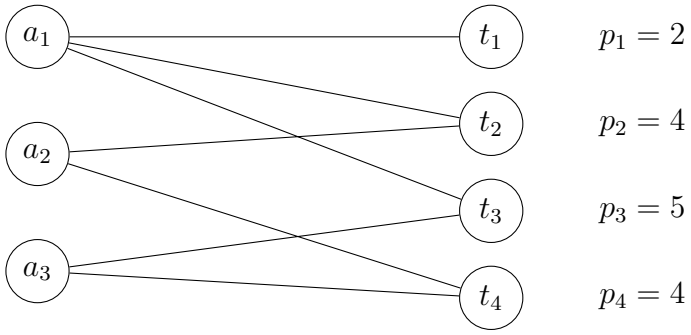


Figura 2.2: Esempio numerico di un'istanza di task assignment.

sono assegnazioni valide, di massima cardinalità e con somma dei pesi $p(\mathcal{M}_1 = p(\mathcal{M}_2) = 9$ e si verifica facilmente, per enumerazione, che tale valore è il massimo ottenibile per l'istanza data.

2.1.1 Rappresentazione PLI

Una rappresentazione algebrica alternativa alla teoria dei grafi è la seguente. Sia $\mathbf{x} \in \{0,1\}^{N \cdot M}$ un vettore (di stato) binario definito come

$$x_{i,j} = \begin{cases} 1 & \text{se l'agente } a_i \text{ esegue il task } t_j \\ 0 & \text{altrimenti.} \end{cases} \tag{2.3}$$

Allora, perché esso rappresenti un'assegnazione ammissibile, deve essere vincolato dalla definizione 2.1, ovvero ogni task può essere eseguito al più da un agente

$$\sum_{i=1}^{N} x_{i,j} \leq 1 \qquad 1 \leq j \leq M \tag{2.4}$$

e ogni agente può eseguire al più un task

$$\sum_{j=1}^{M} x_{i,j} \leq 1 \qquad 1 \leq i \leq N \tag{2.5}$$

o, equivalentemente, in forma matriciale compatta

$$\mathbf{Ax} \leq \mathbf{b} \tag{2.6}$$

con $\mathbf{x} = (x_{1,1}, x_{1,2}, \dots x_{1,M}, x_{2,1}, \dots x_{N,M})^\top$ e

$$\mathbf{A} = \begin{pmatrix} I_M & I_M & \cdots & I_M \\ \mathbf{1}_M^\top & \mathbf{0} & & \mathbf{0} \\ \mathbf{0} & \mathbf{1}_M^\top & & \\ \vdots & \mathbf{0} & \cdots & \vdots \\ & \vdots & & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & & \mathbf{1}_M^\top \end{pmatrix} \in \{0,1\}^{(N+M)\times NM} \tag{2.7}$$

e $\mathbf{b} = \mathbf{1}_{N+M}$.[1]

Definiamo formalmente

Definizione 2.3 (Problema primale)**.** Un problema di assegnazione in forma primale è espresso come:

$$\max f(\mathbf{x}) = \max \sum_{i,j} p_j \, x_{i,j} = \max \mathbf{c}^\top \mathbf{x} \tag{2.8}$$

con $\mathbf{c}$ *vettore dei costi*, di dimensioni uguali ad $\mathbf{x}$ e parimenti ordinato, che raccoglie le priorità p_j associate alle variabili $x_{i,j}$.

Vincolato da (*forma standard*):

$$\mathbf{A}\mathbf{x} \leq \mathbf{1} \tag{2.9a}$$

$$\mathbf{x} \geq \mathbf{0} \tag{2.9b}$$

o equivalentemente in *forma canonica* da:

$$\mathbf{A}\mathbf{x} + I_{N+M}\mathbf{s} = \mathbf{1} \tag{2.10a}$$

$$\mathbf{x}, \mathbf{s} \geq \mathbf{0} \tag{2.10b}$$

con $\mathbf{s} = \{s_1, \dots, s_N, s_{N+1}, \dots, s_{N+M}\}$, variabili *slack*.

In riferimento al precedente esempio numerico (figura 2.2), posto

$$\mathbf{x} = (x_{1,1}, x_{1,2}, x_{2,2}, x_{2,4}, x_{3,3}, x_{3,4})^\top$$

il vettore di stato che rappresenta gli archi del grafo bipartito, con valori definiti nella 2.3, il problema risulta

[1] Si intende con I_k e $\mathbf{1}_k$ rispettivamente la matrice identità di ordine k e il vettore formato da k uno in colonna.

il modello PLI in forma standard è dato da

$$\max f(\mathbf{x}) = \max (5, 1, 1, 2, 3, 4)^\top \mathbf{x}$$

vincolato da

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ & & & & & \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ x_{2,2} \\ x_{2,4} \\ x_{3,3} \\ x_{3,4} \end{pmatrix} \leq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \\ 1 \\ 1 \\ 1 \end{pmatrix} \tag{2.11}$$

$$x_{1,1}, x_{1,2}, x_{2,2}, x_{2,4}, x_{3,3}, x_{3,4} \geq 0 \tag{2.12}$$

Si noti che la matrice dei vincoli in 2.11 è ottenuta riducendo la matrice standard $\mathbf{A}$ della 2.7 eliminando le colonne relative a variabili (accoppiamenti) non possibili, ovvero ad archi non presenti nel grafo associato.

In entrambe le forme il numero di vincoli è $N + M$ più le H ($H + N + M$ per la forma canonica) disequazioni che impongono la positività delle variabili. In particolare nella forma canonica le variabili di slack $\{s_i : 1 \leq i \leq M\}$ sono interpretabili come

$$s_i = \begin{cases} 1 & \text{se il task } t_i \text{ non viene eseguito} \\ 0 & \text{altrimenti.} \end{cases} \tag{2.13}$$

e, banalmente, per $\{s_i : M + 1 \leq i \leq N + M\}$

$$s_i = \begin{cases} 1 & \text{se l'agente } a_i \text{ non esegue alcun task} \\ 0 & \text{altrimenti.} \end{cases} \tag{2.14}$$

2.1.2 Simplesso

Da un punto di vista geometrico le disequazioni nelle 2.9 formano un poliedro convesso e limitato nell'ortante positivo $\mathbb{R}_+$, detto *politopo*, il cui volume rappresenta la totalità delle soluzioni ammissibili.[2]

Ogni vertice del politopo identifica una *soluzione base* ammissibile che corrisponde alla soluzione (unica) di un sistema

$$B\mathbf{x}_B = \mathbf{b} \tag{2.15}$$

con $B \in \mathbb{R}_+^{N+M}$ una matrice ottenuta scegliendo $N + M$ colonne linearmente indipendenti da $\bar{A} = [\mathbf{A}, I_{N+M}]$ e $\mathbf{x}_B$ la corrispondente selezione di variabili dal vettore $\bar{\mathbf{x}}^\top = [\mathbf{x}^\top, \mathbf{s}^\top]$. La matrice B prende il nome di *base* e il vettore di stato $\mathbf{x}, \mathbf{s}$ è così partizionato in variabili *in base* $\mathbf{x}_B \geq 0$ e variabili *fuori base* $\mathbf{x}_F = 0$.

Un risultato fondamentale della programmazione matematica [6] afferma che

Proposizione 2.2.
Se il problema in def.2.3 è limitato e non banale (ovvero lo è il politopo associato) possiede almeno una soluzione ottima identificata da una soluzione base ammissibile e quindi insistente su un vertice del politopo associato.

Inoltre un'altra conveniente proprietà che si applica al problema dell'assegnazione è la seguente.

Proposizione 2.3.
In un problema di programmazione lineare, se la matrice dei vincoli $\mathbf{A}$ *è totalmente unimodulare (TUM), allora tutte le possibili soluzioni base ad essa associate hanno valore intero.*

Essendo la proprietà di totale unimodularità definita come segue, si dimostra [7] che essa è sempre valida per una matrice con struttura analoga alla $\mathbf{A}$ descritta nella 2.7.

Definizione 2.4 (Totale unimodularità)**.** Una matrice intera è *totalmente unimodulare* se ogni sua sottomatrice quadrata non singolare ha determinante unitario in modulo.

In breve, identificare la soluzione ottima al problema dell'assegnazione equivale a trovare il vertice del politopo che massimizza la $f(\mathbf{x})$.

[2] Soluzioni ammissibili al rilassamento continuo corrispondente, visto che non sono inclusi esplicitamente vincoli di interezza sulle variabili.

Il metodo risolutivo standard per questo genere di problemi è il metodo del simplesso ([6]). Esso prevede l'ispezione iterativa delle soluzioni base ammissibili (lungo un cammino di adiacenza dei vertici), che vede crescere in maniera monotona la funzione obiettivo $f(\mathbf{x})$ fino al raggiungimento di un ottimo. Di seguito si riporta uno schema di principio di tale metodo.

Algoritmo 1 Metodo del simplesso

Input: $\bar{A} = [\mathbf{A}, I_{N+M}]$, $\mathbf{b}$, $\mathbf{c}$, B: base ammissibile iniziale

1: **while** B identifica soluzione non ottima **do**
2: Selezionare una variabile *entrante* $x_{in} \in \mathbf{x}_F$ che incrementi $f(\mathbf{x})$.
3: Identificare una variabile *uscente* $x_{out} \in \mathbf{x}_B$ in modo da ottenere nuova soluzione base ammissibile
4: Effettuare operazione *pivot*:
5: $\mathbf{x}_B \leftarrow (\mathbf{x}_B \setminus x_{out}) \cup x_{in}$
6: $\mathbf{x}_F \leftarrow (\mathbf{x}_F \setminus x_{in}) \cup x_{out}$
7: **end while**
8: **return**

Definizione 2.5 (Degenerazione primale)**.** Un base B è detta *degenere* se corrisponde ad una soluzione base $\mathbf{x}_B$ con almeno una componente nulla.

In presenza di degenerazione primale c'è la possibilità che il metodo del simplesso percorra cicli (potenzialmente infiniti) tra basi differenti, senza incrementare il valore della funzione obiettivo [6]. Questo è vero, in particolare, se non si prendono alcune precauzioni nella scelta delle variabili entranti e uscenti ai passi 2 e 3, tra le quali si possono citare la regola di Bland [3], criteri randomici oppure lessicografici, presentati in seguito.

Si osservi che nel caso specifico del problema di assegnazione, con qualunque istanza per cui il numero di task eccede il numero di agenti ($N < M$), si è in presenza di degenerazione primale. Infatti le variabili in base sono necessariamente $N + M$, in numero strettamente maggiore degli N accoppiamenti corrispondenti a variabili di valore unitario.

Dualità

Tramite semplici manipolazioni delle matrici e dei vettori coinvolti nel problema in def.2.3 si ottiene il problema *duale* associato:

Definizione 2.6 (Problema duale)**.** Il problema primale in def.2.3 è equivalente al problema di programmazione lineare

$$\min f(\mathbf{y}) = \min \left(\sum_{i=1}^{N} u_i + \sum_{j=1}^{M} v_j \right) \tag{2.16}$$

Vincolato da (*forma standard*):

$$\mathbf{A}^\top \mathbf{y} \geq \mathbf{c} \tag{2.17a}$$
$$\mathbf{y} \geq \mathbf{0} \tag{2.17b}$$

o equivalentemente in *forma canonica* da:

$$\mathbf{A}^\top \mathbf{y} - I_H \mathbf{r} = \mathbf{c} \tag{2.18a}$$
$$\mathbf{y}, \mathbf{r} \geq \mathbf{0} \tag{2.18b}$$

con $\mathbf{y}^\top = (u_1, \ldots, u_N, v_1, \ldots, v_M)^\top$, e $\{r_1, \ldots, r_H\}$ variabili *surplus*.

Analogamente al problema primale si può definire una situazione di *degenerazione duale*, in presenza di soluzioni al problema duale che prevedono almeno una componente di $\mathbf{y}_B$ di valore nullo.

Inoltre si dimostra [3] che un problema dualmente degenere comporta soluzioni ottime multiple al problema primale.

La teoria della dualità garantisce la seguente.

Proposizione 2.4 (Condizioni di ottimalità (Complementary Slackness))**.** *Due soluzioni ammissibili al problema primale (def.2.3),* $\bar{\mathbf{x}} \in 0, 1^H$*, e duale (def.2.6),* $\bar{\mathbf{y}} \in \mathbb{R}^{N+M}$ *sono ottime se e solo se:*

$$\bar{\mathbf{y}}^\top (\mathbf{A}\bar{\mathbf{x}} - \mathbf{1}) = 0 \tag{2.19a}$$
$$\bar{\mathbf{x}}^\top (\mathbf{A}^\top \bar{\mathbf{y}} - \mathbf{c}) = 0 \tag{2.19b}$$

Al fine di garantire l'ottimalità della soluzione, risulta quindi equivalente lavorare sul problema primale o duale, a patto di rispettare, in ultima analisi, le 2.19.

2.2 Hungarian method

Il procedimento risolutivo per il problema dell'assegnazione noto come *metodo ungherese* (introdotto da Kuhn [12] e Munkres [15]) è un metodo cosiddetto *primale-duale*, che a differenza del simplesso evolve lungo soluzioni ammissibili del problema duale e soluzioni parziali al problema primale.

Più in dettaglio, nel simplesso si parte da una soluzione ammissibile al problema primale e ci si muove (si passa ad una soluzione base adiacente) verso soluzioni che determinano valori della funzione obiettivo non decrescenti (auspicabilmente crescenti strettamente)[3] imponendo il mantenimento dell'ammissibilità primale, fino a giungere alla base ottima, che per le condizioni 2.19, risulta ammissibile ed ottima anche per il problema duale associato.

Al contrario, il metodo ungherese impone di mantenere le soluzioni intermedie ammissibili per il problema duale, incrementando ad ogni iterazione il numero di variabili che risultano ammissibili per il problema primale, per arrivare, infine, alla soluzione ottima per l'istanza globale.

Si consideri nuovamente il problema di assegnazione ottima in def.2.2 su un grafo bipartito pesato $\mathcal{G} = (\mathcal{A}, \mathcal{T}, \mathcal{H}, p)$.

Sia definito $\bar{H} \subseteq \mathcal{H}$ l'insieme degli archi *assegnati* e, di conseguenza $\bar{A} \subseteq \mathcal{A}$ e $\bar{T} \subseteq \mathcal{T}$ gli insiemi di vertici *assegnati* incidenti agli archi di $\bar{H}$. Inoltre:

- Un *cammino alternante* è un cammino semplice in $\mathcal{G}$, i cui archi alternano elementi di $\bar{H}$ e $\mathcal{H} \setminus \bar{H}$.

- Un *albero alternante* radicato in $k \in \mathcal{A} \cup \mathcal{T}$ è un albero in cui tutti i cammini che partono dalla radice k sono alternanti.

- Un *cammino aumentante* è un cammino aumentante in cui i archi e vertici iniziali e finali non sono assegnati.

Da notare che per definizione un cammino aumentante di lunghezza n contiene $\lfloor \frac{n}{2} \rfloor$ archi indipendenti $\bar{H}$ e $\lfloor \frac{n}{2} \rfloor + 1$ archi indipendenti in $\mathcal{H} \setminus \bar{H}$. Questi ultimi se sostituiti ai primi costituiscono un'assegnazione di cardinalità strettamente maggiore.

Ad ogni vertice di $a_i \in \mathcal{A}$ sia assegnata un'*etichetta*, $u_i \in \mathbb{R}$, e per ogni $t_j \in \mathcal{T}$ sia definito, $v_j \in \mathbb{R}$. Consistentemente ai vincoli della forma duale 2.18a, si impone:

[3] Per formulazioni del problema in forma di massimo.

Definizione 2.7 (Labelling ammissibile)**.** Un'etichettatura

$$(u_1, \ldots, u_N, v_1, \ldots, v_M)$$

è detta *ammissibile* se, per ogni (i, j), vale

$$u_i + v_j \geq p_{ij} \tag{2.20}$$

Sia definito, infine, un sottografo $\mathcal{G}_0 = (\mathcal{A}, \mathcal{T}, \mathcal{H}_0 \subseteq \mathcal{H})$ i cui archi sono tutti e soli gli archi incidenti a nodi con etichettatura che soddisfa $u_i + v_j = p_{ij}$

A partire da un labelling ammissibile e da un'assegnazione iniziale $\mathcal{M} = \emptyset$ vuota, il metodo ungherese consiste in:

Algoritmo 2 Metodo ungherese

Input: $\mathcal{G} = (\mathcal{A}, \mathcal{T}, \mathcal{H}, p)$, $u = (u_1, \ldots, u_N), v = (v_1, \ldots, v_M)$: labelling valido

1: $\bar{H}, \bar{A}, \bar{T} \leftarrow \emptyset$
2: Crea $\mathcal{G}_0$ in accordo con u, v, p
3: **while** $|\bar{H}| < N$ **do**
4: Cerca cammino aumentante γ in $\mathcal{G}_0$
5: **if** $\exists\gamma$ **then**
6: Inverti lo stato di assegnazione degli archi di γ
7: Aggiorna $\bar{H}, \bar{A}, \bar{T}$
8: **else**
9: Modifica labelling u, v in modo da aumentare la dimensione di $\mathcal{H}_0$
10: **end if**
11: **end while**
12: **return**

Si noti come il funzionamento sia scandito dalla condizione allo step 3: dopo una prima fase di inizializzazione viene eseguito un controllo sulla cardinalità dell'assegnazione e se essa non è massima viene incrementata di un'unità, eventualmente manipolando l'etichettatura (operazione analoga al *pivot*, e quindi al cambio di base nel metodo del simplesso).

Nella pratica l'efficienza del metodo dipende molto da come se ne implementano le fasi principali.:

1. Creazione di un labelling iniziale (u, v) valido.

2. Ricerca del cammino aumentante γ (step 4).

3. Modifica del labelling per incrementare il grafo $\mathcal{G}_0$ (step 9).

L'algoritmo utilizzato per la presente è basato pricipalmente su [15], [4] e [13] è descritto in dettaglio nel capitolo 3.

2.3 Ottimizzazione astratta

Si consideri un problema di ottimizzazione (combinatoria) definito da una coppia $(\mathcal{H}, \phi)$, con $\mathcal{H}$ insieme finito di elementi denominati *vincoli* e

$$\phi : 2^{\mathcal{H}} \to \Phi : \mathcal{H} \supseteq G \mapsto \phi(G)$$

una funzione che mappa ogni sottoinsieme G di $\mathcal{H}$ in un insieme dotato di ordinamento $(\Phi, \leq)$.

Più concretamente, $\phi(G)$ identifica il valore ottimo (per esempio minimo) di una funzione obiettivo opportunamente definita su un dominio che soddisfa tutti i vincoli di G.

Definizione 2.8. Il problema di ottimizzazione appena introdotto è detto *problema astratto di ottimizzazione* se sono verificate le seguenti ipotesi:

1. *Monotonia*: $F \subset G \subseteq \mathcal{H} \implies \phi(F) \leq \phi(G)$

2. *Località*: se $F \subset G \subseteq \mathcal{H}$, con $\phi(F) = \phi(G)$, allora, $\forall h \in \mathcal{H}$:

$$\phi(G) < \phi(G \cup \{h\}) \implies \phi(F) < \phi(F \cup \{h\}) \tag{2.21}$$

Un insieme di vincoli $B \subseteq \mathcal{H}$ è *minimo* (o equivalentemente una *base*) se, per ogni sottoinsieme strettamente proprio $A \subset B$, si ha $\phi(A) < \phi(B)$. In altri termini, dato $G \subset \mathcal{H}$, B_G è una base per G se è un suo sottoinsieme minimo per cui $\phi(B_G) = \phi(G)$.

Una *soluzione* per il problema astratto $(\mathcal{H}, \phi)$ è una base $B_{\mathcal{H}} \subset \mathcal{H} : \phi(B_{\mathcal{H}}) = \phi(\mathcal{H})$.

2.3.1 Abstract linear programming

Di particolare interesse è verificare se il framework dell'ottimizzazione astratta appena introdotto si applica ai problemi di assegnazione, e, più in generale ai problemi di programmazione lineare. Le considerazioni che seguono dimostrano che è questo il caso, a patto di prendere qualche precauzione nella conversione.

Un problema di PLI (non necessariamente di assegnazione) definito su un vettore $\mathbf{x} \in \mathbb{R}^n$ con la consueta notazione[4]

$$\min f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x} \tag{2.22}$$

$$\begin{cases} a_i \mathbf{x} \leq b_i \\ \mathbf{x} \geq 0 \end{cases} \tag{2.23}$$

è riformulabile in termini di ottimizzazione astratta specificando la coppia $(\mathcal{H}, \phi)$.

I vincoli che compongono l'insieme $\mathcal{H}$ sono, banalmente, le singole disequazioni che compongono il politopo, o meglio i semipiani che ne sono identificati, quindi $h_i = \{\mathbf{x} \in \mathbb{R}^n : a_i \mathbf{x} \leq b_i\}$.

Per quanto riguarda ϕ non è invece sufficiente considerare la funzione obiettivo $\phi(G) = \mathbf{c}^\top \mathbf{x}$, in quanto violerebbe l'ipotesi di località.

Es. A titolo di esempio si consideri il seguente problema in $\mathbb{R}^2$, riportato anche in figura 2.3:

$$\min \ f(\mathbf{x}) = \min \ (-x_1 - x_2) = \min \ (-1 - 1)\mathbf{x} \tag{2.24}$$

$$\begin{cases} h_1 : \quad x_1 + x_2 \leq 3 \\ h_2 : \quad x_1 \leq 1 \\ h_3 : \quad x_2 \leq 1 \end{cases} \tag{2.25}$$

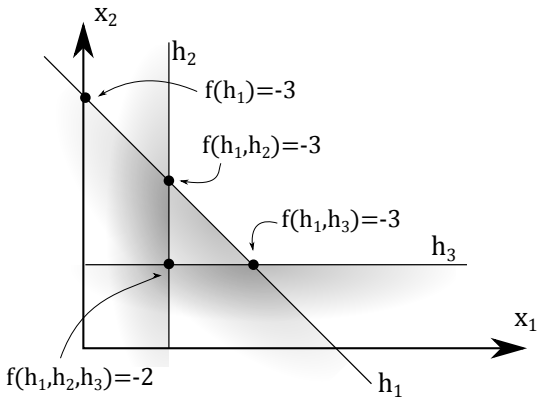


Figura 2.3: Esempio formulazione PLI che non rispetta la condizione di località.

[4]Nel caso di un problema di assegnazione $n = NM$.

Posto $F \subset G \subset \mathcal{H} = \{h_1, h_2, h_3\}$ e, rispettivamente, $F = \{h_1\}$ e $G = \{h_1, h_2\}$ è banale verificare che (con piccolo abuso di notazione)

$$f(F) = f(G) = -3 \tag{2.26}$$

$$f(G) < f(G \cup \{h_3\}) = -2 \tag{2.27}$$

e tuttavia $f(F)$=$f(F \cup \{h_3\}) = -3$, contraddicendo la 2.21. □

In effetti il problema non si presenta a patto di lavorare con un ordinamento stretto tra le basi. Questo equivale ad introdurre una perturbazione arbitraria della funzione obiettivo che induca soluzione ottima unica per ogni possibile politopo generato da sottoinsiemi dei vincoli del problema PLI.

Un'alternativa spesso considerata in letteratura ([6] e [11], per esempio) e che offre anche alcuni vantaggi di implementazione è la seguente.

Sia $\delta = (\delta_0, {\delta_0}^2, \ldots, {\delta_0}^n) \in \mathbb{R}^n$ una perturbazione da applicare ad $f(\mathbf{x})$ come segue:

$$f'(\mathbf{x}) = \left(\mathbf{c}^\top + \delta\right)\mathbf{x} \tag{2.28}$$

Si dimostra che, per $\delta_0 > 0$ sufficientemente piccolo, la perturbazione non modifica il valore della vecchia funzione obiettivo $f(\mathbf{x})$, e identifica in ogni caso una soluzione $\mathbf{x}$ che risolve ottimamente il problema 2.22.

Concretamente, si impone come soluzione ottima, nel problema 2.22, la soluzione con funzione obiettivo minima e che sia rappresentata rappresentata dalla base lessicograficamente minima (*lex-min*). Formalizzando, si dice che:

Definizione 2.9 (Ordinamento lessicografico)**.** Un vettore $\mathbf{z} = (z_1, z_2, \ldots, z_k)$ è *lessicograficamente positivo* (lex-positivo, in notazione $\mathbf{z}{>_{lex}}0$) se $\mathbf{z} \neq 0$ e la sua prima componente non nulla è positiva. Dati due vettori $\mathbf{p}, \mathbf{q} \in \mathbb{R}^k$, si definisce $\mathbf{p}{>_{lex}}\mathbf{q}$ se $\mathbf{p} - \mathbf{q}{>_{lex}}0$. Per estensione un vettore $\mathbf{q} \in Q$ si dice *lex-minimo* ($\min_{lex}$) se $\nexists \mathbf{p} \in Q : \mathbf{p}{<_{lex}}\mathbf{q}$.[5]

[5] Analogamente sono definite tutte le relazioni d'ordine lessicografico $\geq_{lex}, =_{lex}, <_{lex}, \leq_{lex}$ e l'operatore $\max_{lex}$.

2.4 Constraint consensus

Dato un problema di ottimizzazione astratta come precedentemente definito, la cui rappresentazione (in termini di vincoli) è partizionata e distribuita su una rete di agenti in grado di comunicare, è possibile stabilire una strategia di consenso in grado di fornire una soluzione ottima al problema globale. Tale soluzione è raggiunta in tempo finito da ogni singolo agente della rete.

2.4.1 Distributed abstract problem

Definizione 2.10. Un *problema astratto distribuito* (Distributed Abstract Problem, DAP nel proseguio) è un vettore di tre elementi $(\mathcal{G}, (\mathcal{H}, \phi), \mathcal{F})$ in cui

1. $\mathcal{G} = (\mathcal{A}, E_{com})$ è un grafo diretto che rappresenta una rete di comunicazione sincrona con $\mathcal{A}$ l'insieme dei nodi ed E_{com} l'insieme dei collegamenti;

2. $(\mathcal{H}, \phi)$ è un problema di ottimizzazione astratta, con $\mathcal{H}$ l'insieme dei vincoli e ϕ funzione di costo associata;

3. $\mathcal{F} : \mathcal{H} \to \mathcal{A}$ una relazione ovunque definita che associa ad ogni vincolo in $\mathcal{H}$ uno o più *agenti* tra i vertici del grafo di comunicazione.

Una *soluzione* per $(\mathcal{G}, (\mathcal{H}, \phi), \mathcal{F})$ si ottiene quando ogni agente in $\mathcal{A}$ ha calcolato una soluzione del problema $(\mathcal{H}, \phi)$.

Un vincolo $h \in \mathcal{H}$ *viola* un insieme $G \subseteq \mathcal{H}$ se e solo se $\phi(G \cup \{h\}) > \phi(G)$.

2.4.2 Consensus method

Il seguente semplice algoritmo (algoritmo 3) risolve il problema di ottimizzazione astratta distribuita $(\mathcal{G}, (\mathcal{H}, \phi), \mathcal{F})$ della definizione 2.10.

Siano definiti per ogni agente $a_i \in \mathcal{A}$:

1. $H^{[i]}_{LOC}$: insieme dei *vincoli locali*, ovvero un sottoinsieme di $\mathcal{H}$ attribuito all'agente a_i dalla funzione $\mathcal{F}$ precedentemente definita.

2. $H^{[i]}_{REM}(t)$: insieme dei *vincoli esterni* che vengono comunicati dagli altri agenti ad a_i durante il t-esimo intervallo di comunicazione.

3. $B^{[i]}(t)$: base, o insieme minimo dei vincoli, che risolve il problema astratto locale per l'agente a_i all'inizio del t-esimo intervallo di comunicazione.

Al termine di ogni intervallo di comunicazione ogni agente calcola un problema astratto locale $\mathcal{H}^{[i+1]}(t)$ in base alle informazioni di cui dispone (step 1) e ne estrae una base $B^{[i]}(t+1)$ (step 2) da condividere con gli altri agenti nella successiva iterazione (step 3).

Algoritmo 3 Constraint Consensus

Input: $H^{[i]}_{LOC}$, $H^{[i]}_{REM}(t)$, $B^{[i]}(t)$
1: $\mathcal{H}^{[i]}(t+1) \leftarrow H^{[i]}_{LOC} \cup H^{[i]}_{REM}(t) \cup B^{[i]}(t)$
2: $B^{[i]}(t+1) \leftarrow$ soluzione del problema astratto $(\mathcal{H}^{[i]}(t+1), \phi)$
3: Comunica $B^{[i]}(t+1)$ a tutti gli agenti direttamente raggiungibili.

Tale procedura fa sì che ogni agente converga, in un numero di iterazioni finito, ad una base stazionaria, che si dimostra essere soluzione per il il problema astratto distribuito di partenza.

2.4.3 Correttezza

Proposizione 2.5 (Correttezza)**.**
Si consideri $(\mathcal{G}, (\mathcal{H}, \phi), \mathcal{F})$ con $\mathcal{G}$ fortemente connesso,[6] *vale:*

(i) per ogni nodo i il valore $\phi(B_i(t))$ evolve in maniera monotona non decrescente e converge in tempo finito ad un valore costante $\bar{\phi}$;

(ii) l'algoritmo di consenso distribuito risolve il problema $(\mathcal{G}, (\mathcal{H}, \phi), \mathcal{F})$, ovvero in tempo finito le basi B_i sono soluzioni del problema astratto associato $(\mathcal{H}, \phi)$;

(iii) se $(\mathcal{H}, \phi)$ ha un'unica soluzione ottima $B_{\mathcal{H}}$, le basi di convergenza dei singoli nodi corrispondono a $B_{\mathcal{H}}$.

Dimostrazione. Dall'ipotesi di monotonia 2.21 e dalla limitatezza di $\mathcal{H}$ consegue direttamente la (i).

Per quanto riguarda il punto (ii) si supponga che ogni nodo abbia raggiunto la propria base finale (stazionaria) all'istante $\bar{t}$ e che per assurdo almeno due nodi, i e j, abbiano calcolato basi diverse, ovvero $\phi(B_i) \neq \phi(B_j)$. Si consideri un qualunque cammino semplice, insistente in $\mathcal{G}$, che collega i a j, rinominando, senza perdita di generalità, i nodi che lo compongono $i, i+1, i+2, \ldots, j$. Lungo tale percorso la base B_k raggiunta dal generico nodo k non è violata da alcun

[6] Un grafo diretto o indiretto è detto *fortemente connesso* se e solo se, per ogni sua coppia di vertici $v_h \neq v_k$ esiste un cammino da v_h a v_k.

vincolo in B_{k-1} (in caso contrario si contraddirebbe l'ipotesi di raggiunta base finale, perché il nodo k dovrebbe aggiornare i propri vincoli).

Si può quindi affermare che $\phi(B_{k+1}) \leq \phi(B_k)$ e, iterando lungo il cammino, si ha che

$$\phi(B_i) \leq \phi(B_{i+1}) \leq \cdots \leq \phi(B_j)$$

Le stesse considerazioni, dato che $\mathcal{G}$ è fortemente connesso si applicano a qualunque cammino da j a i, ottenendo che $\phi(B_j) \leq \cdots \leq \phi(B_i)$ e in definitiva che $\phi(B_j) = \phi(B_i)$, contraddicendo l'ipotesi iniziale. Ricapitolando, dall'istante $\bar{t}$ in poi

$$\phi(B_1) = \phi(B_2) = \cdots = \phi(B_n) = \bar{\phi}$$

Per induzione e sempre lungo un cammino che attraversa tutti i nodi, si dimostra che

$$\phi(B_1 \cup \cdots \cup B_N) = \bar{\phi} \tag{2.29}$$

Infatti, se $\phi(B_1) = \bar{\phi}$, $\phi(B_1 \cup B_2) = \bar{\phi}$ per costruzione dell'algoritmo: B_2 è una base per $B_1 \cup B_2$ e si è già dimostrato che $\phi(B_i) = \bar{\phi}$.

Se, per assurdo, $\phi(B_1, \ldots, B_k, B_{k+1}) > \bar{\phi}$, applicando l'ipotesi induttiva

$$\phi(B_1 \cup B_2 \cup \cdots \cup B_k) = \phi(B_1 \cup \cdots \cup B_{k+1})$$

si otterrebbe che

$$\phi(B_1 \cup \cdots \cup B_k) < \phi(B_1 \cup \cdots \cup B_k \cup B_{k+1})$$

e quindi dovrebbe esistere $h \in B_{k+1}$ tale che

$$\phi(B_1 \cup \cdots \cup B_k) < \phi(B_1 \cup \cdots \cup B_k \cup \{h\})$$

In questo caso, ricordando che $\phi(B_k) = \bar{\phi}$ e, per l'ipotesi induttiva, $\phi(B_1 \cup \cdots \cup B_k) = \bar{\phi}$, varrebbe anche, applicando il principio di località nella definizione 2.8, con $F = B_k$ e $G = B_1 \cup \cdots \cup B_k$,

$$\phi(B_k) < \phi(B_k \cup \{h\}) \tag{2.30}$$

e quindi, data la diretta comunicazione tra B_k e B_k, alla convergenza si avrebbe la contraddizione

$$\phi(B_k) = \phi(B_{k+1}) \geq \phi(B_k \cup \{h\}) > \phi(B_k)$$

con il $\geq$ dovuto all'ipotesi di monotonia ed il $>$ direttamente dall'equazione 2.30. Questo prova la 2.29.

Infine, siccome nessun vincolo $h \in \mathcal{H}$ viola l'insieme $B_1 \cup \cdots \cup B_N \subset \mathcal{H}$

$$\bar{\phi} = \phi(B_1 \cup \cdots \cup B_N) = \phi(\mathcal{H}) \tag{2.31}$$

ovvero dopo $\bar{t}$ ogni base B_i calcolata dai vari nodi, risolve il problema di ottimizzazione astratta $(\mathcal{H}, \phi)$, ed è quindi provato il punto (ii).

Il punto (iii) è banalmente vero. □

2.4.4 Terminazione

Per quanto riguarda le condizioni di uscita dall'algoritmo è sufficiente la conoscenza, da parte degli agenti, del *diametro* del grafo di comunicazione $\mathcal{G}$ per determinare quando si sia raggiunta la base ottima.

Definito $diam(\mathcal{G})$, il diametro di un grafo fortemente connesso $\mathcal{G}$, come il massimo tra i cammini minimi tra ogni coppia di vertici, si prova il seguente risultato.

Proposizione 2.6 (Terminazione)**.**
Si consideri $(\mathcal{G}, (\mathcal{H}, \phi), \mathcal{F})$ con $\mathcal{G}$ fortemente connesso, se un nodo ha raggiunto una soluzione stazionaria da $(2 \cdot diam(\mathcal{G}) + 1)$ non avrà ulteriori aggiornamenti della stessa. Tale nodo può interrompere l'algoritmo di consenso.

Dimostrazione. Per ogni arco di comunicazione diretto da a_k ad a_{k+1} si ha:

$$\phi(B_k(t)) \leq \phi(B_{k+1}(t+1)) \tag{2.32}$$

perché, dato il funzionamento dell'algoritmo, nessun vincolo della base $B_k(t)$ viola la base $B_{k+1}(t+1)$. Si ipotizzi che per gli istanti $t : t_0 \leq t \leq t_0 + 2diam(\mathcal{G})$ la base di a_k resti stabile: $B_k(t) = B$. Posto, per semplicità, $t_0 = 0$, iterando la 2.32 lungo un cammino di comunicazione diretta su $\mathcal{G}$ ti ottiene $\phi(B) \leq \phi(B_{k+1}(t+1)) \leq \phi(B_{k+2}(t+2)) \leq \ldots$ e, più in generale per ogni nodo $a_h \neq a_k$, che

$$\phi(B) \leq \phi(B_h(dist(a_k, a_h)))$$

Iterando nuovamente su un percorso da a_h ad a_k consegue

$$\phi(B) \leq \phi(B_h(dist(a_k, a_h))) \leq \phi(B_k(dist(a_k, a_h) + dist(a_h, a_k)))$$

Ricordando che vale sempre $dist(a_k, a_h) + dist(a_h, a_k) \leq 2diam(\mathcal{G})$ e, per ipotesi, $\phi(B_k(2diam(\mathcal{G})) = \phi(B)$, si conclude che

$$\phi(B) \leq \phi(B_h(dist(a_k, a_h))) \leq \phi(B)$$

e, quindi, per ogni a_h, $\phi(B_h) = \phi(B)$, ovvero non serve continuare l'algoritmo in quanto non sono più possibili miglioramenti di ϕ. □

2.4.5 Assignment Constraint Consensus

In questa parte si analizzano le peculiarità del Constraint Consensus applicato al problema dell'assegnazione 2.2.

Considerando il problema primale in forma canonica 2.3 si hanno N vincoli del tipo 2.5 e M del tipo 2.4, per un totale di $N+M$ a cui vanno ad aggiungersi gli $H + N + M$ vincoli di segno $x_{ij}, s_{ij} \geq 0$.

L'applicazione diretta dell'algoritmo LEX_SIMPLEX porta all'individuazione di esattamente $N + M$ variabili in base. Se il problema ha soluzione di cardinalità massima, la soluzione sarà composta da:

- N variabili $x_{ij} = 1$ che corrispondono ad effettive assegnazioni task-agente.
- $M - N$ variabili slack $s_{ij} = 1$ relative ai task *non* assegnati.
- Le rimanenti N variabili *degeneri* di valore nullo prese tra le x o le s rimanenti. Nel caso particolare in cui la base è *lex*-massima, le variabili degeneri in base saranno tutte $x_{ij} = 0$, in quanto vengono favorite per l'ingresso in base le colonne di indice minore.

In sintesi, a meno di una permutazione tra le prime $2N$ posizioni, la soluzione base sarà un vettore del tipo:

$$\bar{\mathbf{x}}_B = (\overbrace{\underbrace{1, \ldots, 1}_{N}, \underbrace{0, \ldots, 0}_{N}}^{x}, \overbrace{\underbrace{1, \ldots, 1}_{M-N}}^{s}) \tag{2.33}$$

Ciò che interessa, nota la base, è costruire un set minimo di disequazioni che le corrispondono, ovvero una base di vincoli per il problema $(\mathcal{H}, \phi)$.

Banalmente, la soluzione ottima al problema 2.3 è

$$\mathbf{x}_B = B^{-1}\mathbf{1}_{N+M} \tag{2.34}$$

e quindi una rappresentazione minima (e risolvibile in un passo) del problema è

$$\max(f(\mathbf{x})) = \max(\mathbf{c}_B^\top \mathbf{x}_B) \tag{2.35a}$$

$$\text{con:} \quad B\mathbf{x}_B = \mathbf{1}_{N+M} \tag{2.35b}$$

B è per costruzione un set di $N+M$ colonne indipendenti della matrice dei vincoli $\bar{A} = [\mathbf{A}, I_{N+M}]$. Ad ognuna di esse corrisponde, nel problema duale, una riga di $\bar{A}^\top$, ovvero il vettore dei coefficienti nella disequazione $u_i + v_j \geq c_{ij}$, se la colonna apparteneva ad $\mathbf{A}$ (prime $2N$ colonne), oppure $v_j \geq 0$. Concretamente, è possibile costruire un problema duale minimo:

Problema duale:

$$[B_1, \ldots, B_{2N}]^\top \mathbf{y} \geq \mathbf{c}_B \tag{2.36a}$$

$$[B_{2n+1}, \ldots, B_{N+M}]^\top \mathbf{y} \geq \mathbf{0} \tag{2.36b}$$

I vincoli del problema primale 2.35b sono (altro risultato notevole in programmazione lineare [6]) equivalenti ai duali dei duali e quindi:

Problema primale:

$$[B_1, \ldots, B_{2N}]\mathbf{x}_B \leq \mathbf{1}_{2N} \tag{2.37a}$$

$$[B_{2n+1}, \ldots, B_{N+M}]\mathbf{s}_B \geq \mathbf{0} \tag{2.37b}$$

Da notare che le colonne $\{B_{2n+1}, \ldots, B_\delta\}$ sono vettori canonici e identificano, quindi, imposizioni di segno (positivo) sulle variabili nel problema duale.

Osservando la tipologia dei vincoli ci si rende conto che, sia nella formulazione minima primale 2.37 che duale 2.36, vale la seguente proposizione:

Proposizione 2.7 (Set dei vincoli necessari e sufficienti)**.**
Il dato necessario e sufficiente per la ricostruzione del problema è un insieme $\mathcal{B} = \{h_1, \ldots, h_{2N}\}$ *i cui elementi sono*

$$h_k = (i, j, c_{ij}) \in \mathbb{N} \times \mathbb{N} \times \mathbb{R} \tag{2.38}$$

con i *e* j *che identificano l'accoppiamento (eventualmente degenere) agente-task in base e* c_{ij} *la priorità associata allo stesso.*

Infatti i vincoli di positività sulle variabili sono sempre inclusi automatica-

mente nelle formulazioni PLI[7] e $\mathcal{B}$ è sufficiente a costruire il vettore $\mathbf{c}_B$ e la matrice $[B_1, \ldots, B_{2N}]$.

[7] Violare la positività delle variabili, anzi, comporterebbe un'ulteriore aggiunta di variabili fittizie: $\alpha \gtreqless 0 \implies \alpha = \alpha^+ - \alpha^-$, con $\alpha^+, \alpha^- \geq 0$.

3 Algoritmi e Implementazione

In questo capitolo si scende più nel dettaglio degli algoritmi implementati e si presenta un'idea più precisa del funzionamento degli stessi.

3.1 Rappresentazione

Dal punto di vista implementativo si pone il problema di rappresentare in maniera efficiente un certo numero di entità: in particolare la struttura degli agenti e la rete di connettività, i singoli task ed i possibili accoppiamenti. Si è optato per un approccio orientato agli oggetti.

3.1.1 Vincoli

A titolo di esempio si consideri l'istanza che induce il grafo bipartito in figura 3.1.

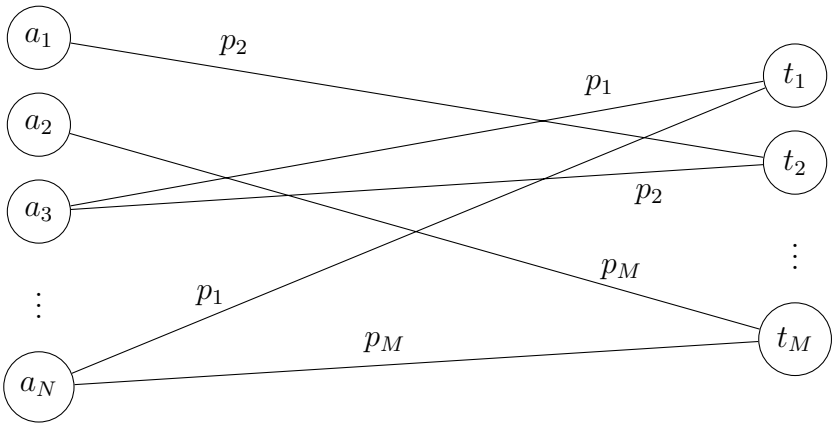


Figura 3.1: Problema dell'assegnazione: rappresentazione come grafo bipartito.

Una sua possibile rappresentazione è una matrice di adiacenza del tipo

$$\mathbf{V}: \quad \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \\ \\ a_N \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & & t_M \end{array} \\ \begin{pmatrix} 0 & p_2 & \cdots & 0 \\ 0 & 0 & & p_M \\ p_1 & p_2 & & 0 \\ \vdots & & \ddots & \\ p_1 & 0 & & p_M \end{pmatrix} \end{array} \tag{3.1}$$

i cui elementi v_{ij} sono definiti come

$$v_{ij} = \begin{cases} p_j & \text{se l'agente } a_i \text{ è in grado di eseguire il task } t_j \\ 0 & \text{altrimenti} \end{cases}$$

con p_j la priorità del task t_j, come noto.

In alternativa, un'estensione a grafo completo, con $\hat{v}_{ij} = -\infty$ per gli archi che identificano accoppiamenti impossibili, è indicata nella 3.2:

$$\hat{\mathbf{V}}: \quad \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \\ \\ a_N \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & & t_M \end{array} \\ \begin{pmatrix} -\infty & p_2 & \cdots & -\infty \\ -\infty & -\infty & & p_M \\ p_1 & p_2 & & -\infty \\ \vdots & & \ddots & \\ p_1 & -\infty & & p_M \end{pmatrix} \end{array} \tag{3.2}$$

È evidente che nel caso in esame, con un numero di possibili accoppiamenti H molto inferiore a NM, le rappresentazioni 3.1 o 3.2 possono risultare onerose da un punto di vista di ingombro, anche nell'ottica dello scambio di dati tra agenti necessario al Constraint Consensus. In altre parole $\mathbf{V}$ risulta una matrice sparsa.

Un'alternativa che risulta più pratica (applicata al grafo 2.1) è, invece

$$\mathcal{H} = ((1,2,p_2), \ldots (2,M,p_M), (3,1,p_1), (3,2,p_2), \ldots, (N,1,p_1), \ldots (N,M,p_M))$$

e quindi, in generale, una rappresentazione compatta del problema è data dall'insieme delle associazioni possibile (ovvero dagli archi del grafo associato) $\mathcal{H} = (h_1, \ldots, h_H)$ nella forma

$$h_k = (i, j, p_j) \qquad \in \mathbb{N} \times \mathbb{N} \times \mathbb{R} \tag{3.3}$$

che identificano puntualmente l'agente a_i ed il task t_j coinvolti e la priorità p_j associata.

Si osservi che h_k contiene le informazioni necessarie e sufficienti a descrivere un *vincolo* nell'ambito dell'ottimizzazione astratta introdotta in sezione 2.4.5.

Un'implementazione dei vincoli è quindi data dalla seguente tabella 3.1

Tabella 3.1: Classe `Constraint`

	`Constraint`
`ida`	Intero univoco identificativo dell'agente.
`idt`	Intero univoco identificativo del task.
`p`	Valore reale positivo corrispondente alla priorità del task.

È utile definire anche una rappresentazione formale per l'insieme dei vincoli $\mathcal{H}$, con una serie di operazioni comuni ad esso applicati, come in tabella 3.2

La relazione tra le classi `Constraint` e `Hset` è quella definita dal diagramma UML in figura 3.2.

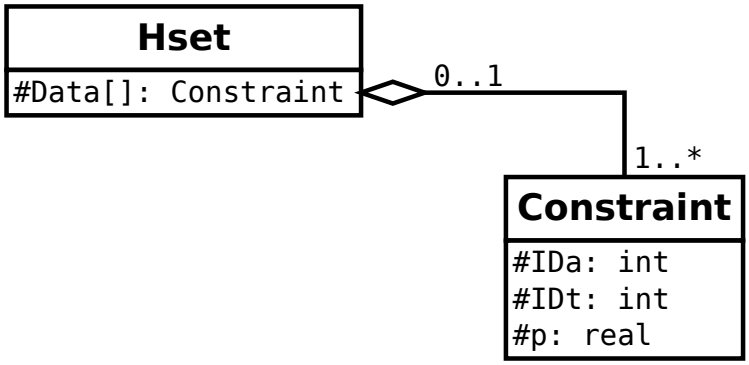


Figura 3.2: Relazione tra le classi `Hset` e `Constraint`

Tabella 3.2: Classe `Hset`

	`Hset`
`data`	Array ordinato (lessicograficamente) di oggetti `Constraint`
`add(H)`	Funzione che aggiunge, in senso insiemistico e quindi senza ripetizioni, a `data` un insieme di vincoli H (sia esso un `Hset`, o un array di `Constraint`)
`remove(h)`	Funzione che cerca e rimuove da `data` un vincolo identificato da `h`, sia esso un `Constraint` oppure una coppia `ida,idt`
`compare(H)`	Verifica se `H` è uguale a questo `Hset`
`isIn(H)`	Verifica se `data` è incluso in `H`
`(A,c,idas,idts) = toPLI`	Funzione che restituisce la matrice `A` dei vincoli ed il vettore `c` dei costi relativi al problema PLI associato ad $\mathcal{H}$, e i vettori `idas` e `idts` contenenti gli id degli agenti e dei task coinvolti.
`(Q,idas,idts) = toTab`	Funzione che restituisce la matrice di adiacenza completa (come in 3.2) `Q` dei vincoli e i vettori `idas` e `idts` contenenti gli id degli agenti e dei task coinvolti.
`fromPLI(A,c,idas,idts)`	Funzione complementare a `toPLI` aggiunge a `data` i vincoli relativi al problema PLI `(A,c,idas,idts)`.
`fromTab(Q,idas,idts)`	Funzione complementare a `toTab` aggiunge a `data` i vincoli della matrice di adiacenza `Q` e i relativi id `(idas,idts)`.

3.1.2 Agenti

Per quanto riguarda il grafo di comunicazione $\mathcal{G}$, le informazioni ad esso relative sono suddivise tra i vari agenti. Nello specifico, ogni agente è in possesso di una lista di nodi (e di relativi indirizzi IP, nel caso la rete sia basata su TCP/IP), con cui può comunicare direttamente. L'unione di tutte le liste descrive in maniera univoca la topologia di comunicazione.

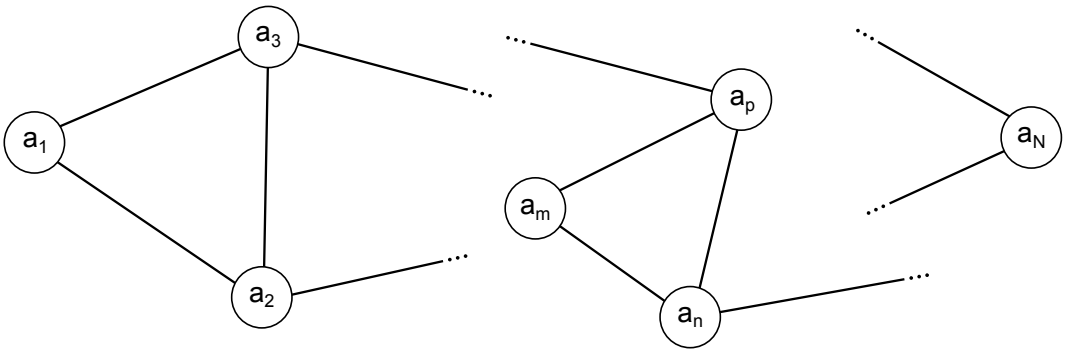


Figura 3.3: Grafo di comunicazione tra agenti $\mathcal{G}$

Analogamente, guardando ai task in istanza e relativi vincoli $\mathcal{H}$, essi sono divisi tra i nodi coerentemente alla definizione di problema astratto distribuito, secondo una mappa $\mathcal{F} : \mathcal{H} \to \mathcal{A}$.

La distribuzione iniziale più logica, quanto meno in uno scenario statico, è che ogni agente sia a conoscenza dei vincoli che lo riguardano direttamente. Quindi una buona mappa iniziale, per ogni $h_k = (i, j, p_j)$, è offerta da

$$\mathcal{F} : h_k \mapsto a_i$$

Naturalmente questa scelta non è a priori vincolante.

La struttura di una classe `Agent` è riportata in tabella 3.3

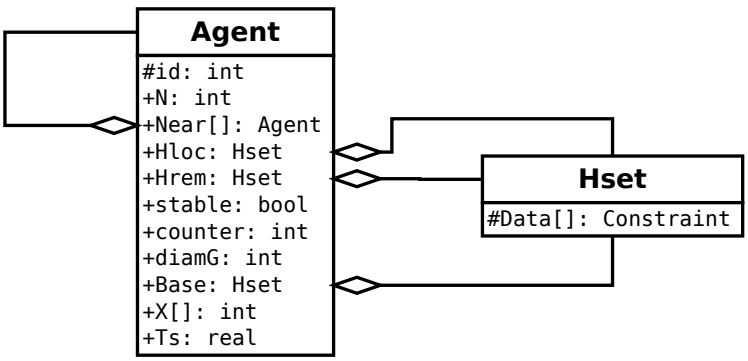


Figura 3.4: Relazione tra le classi `Agent` e `Hset`

Tabella 3.3: Classe `Agent`

	`Agent`
`id`	Intero univoco identificativo dell'agente.
`N`	Numero di agenti nella rete.
`Near`	Lista di agenti con cui è possibile comunicare (in trasmissione) direttamente.
`Hloc`	`Hset` dei vincoli *locali*, ovvero che coinvolgono l'agente.
`Hrem`	`Hset`, buffer in ingresso per le comunicazioni (vincoli base) inviate da altri nodi.
`stable`	Variabile booleana che identifica se la base locale è rimasta costante per un certo numero ($2d_{\mathcal{G}}$) di step.
`counter`	Contatore di iterazioni con base costante.
`diamG`	Diametro del grafo di comunicazione $d_{\mathcal{G}}$.
`Base`	`Hset` Base ottima calcolata localmente.
`X`	Assegnazione ottima calcolata localmente.
`Ts`	Intervallo di aggiornamento.
`onTimeout`	Routine di funzionamento principale che viene eseguita periodicamente (periodo `Ts`)
`(x,base) = computeBase`	Funzione che calcola, a partire dai dati locali, l'assegnazione ottima e la base minima di vincoli che la identifica.
`sendBase(base,ag)`	Funzione che invia la `base` calcolata all'agente `ag`

Al momento di creare un oggetto `Agente` (o nel configurarlo e metterlo online in uno scenario reale), è conveniente inizializzare manualmente alcuni parametri, tra cui l'ID univoco dell'agente, il numero totale di agenti, la lista dei *vicini* nella rete di comunicazione, il diametro del grafo di comunicazione (G) e il set di vincoli locali `Hloc`, che, per rispettare l'ipotesi $N \leq M$ deve contenere da subito almeno un vincolo relativo al task *idle*: per a_i, quindi, $(i, i, 0) \in$ `Hloc`. Da notare, quindi, che i task $(t_1, \ldots, t_N)$ saranno task a priorità nulla.

Naturalmente, le informazioni locali sul numero totale di agenti e sui nodi in ascolto possono facilmente essere ottenute ed aggiornate durante l'esecuzione, attraverso tradizionali comunicazioni Echo.

La funzione `onTimeout` è la principale routine per la selezione del task da svolgere e per il coordinamento con gli altri nodi della rete. In essa è im-

plementata la logica alla base del Constraint Consensus discussa in sezione 2.4.

Il codice dell' algoritmo 4 ne riporta il funzionamento:

- L'agente non ancora stabile verifica la presenza di nuovi vincoli non contenuti nella propria base locale.
- In questo caso calcola una nuova base per il set di vincoli

$$H = H_{LOC} \cup H_{REM} \cup B$$

  e la confronta con la base precedente.
  Se non si rilevano variazioni, il contatore è incrementato, altrimenti il conteggio ricomincia da zero.
- Successivamente è verificata la condizione di terminazione (prop. 2.6): se la base risulta inalterata da $diam(\mathcal{G}) + 1$ chiamate, si può considerare stabile e non verrà aggiornata ulteriormente.
- Raggiunta o meno una base stabile, viene svuotato il buffer `Hrem` e sono spediti i vincoli della base locale a tutti i nodi direttamente raggiungibili.

Con il raggiungimento di una base stabile l'agente ha concluso un'assegnazione ottima globale ed in base ad essa darà il via all'esecuzione del task che gli compete.

La funzione `sendBase` si occupa di comunicare i vincoli della base ai nodi vicini. Per mantenere contenuto il carico sulla rete se si adottano comunicazioni point-to-point (a differenza di quanto avverrebbe con un broadcast su tutti i link in uscita) vengono inviati solo i vincoli che è noto non appartengono già al set di vincoli locali `Hloc` del nodo ricevente.

La funzione `comuputeBase` è il nucleo della risoluzione del problema di task assignment. In essa è implementata la ricerca di un set minimo di basi, che, condiviso con tutti i nodi della rete porti al consenso tra gli stessi e all'ottimizzazione della funzione obiettivo. Sono seguiti e confrontati due approcci: il metodo del simplesso lessicografico ed il metodo ungherese.

Algoritmo 4 Agent.onTimeout

```
1  function onTimeout
2
3  if ~this.stable
4     if isempty(this.Base) || ...
5      ~this.Hrem.isIn(this.Base)
6        %Componi H (set di vincoli da ottimizzare):
7        H = Hset(this.Base);
8        H.add(this.Hloc);
9        H.add(this.Hrem);
10
11       %Calcola assegnazione:
12       [x, base] = computeBase(H);
13
14       %Aggiorna contatore:
15       if base.compare(this.Base)
16          this.counter = this.counter+1;
17       else
18          this.X = x;  this.Base = base;
19          this.counter = 0;  this.stable = 0;
20       end
21    else
22       this.counter = this.counter+1;
23    end
24
25    %Verifica stabilita':
26    if this.counter > 2*this.diamG
27       this.stable = 1;  this.counter = 0;
28    end
29
30    this.Hrem = Hset();  %Svuota buffer
31
32    %Invia dati ai vicini:
33    for ag = this.Near
34       sendBase(this.Base,ag);
35    end
36 end
```

Algoritmo 5 Agent.sendBase

```
1 function sendBase(base,ag)
2 for h = base.data
3    if h(1) ~= ag.id
4       ag.Hrem.add(h)
5    end
6 end
```

3.2 Lex-simplex

L'algoritmo del simplesso lessicografico qui considerato ricalca la struttura del simplesso modificato tradizionale con l'aggiunta di alcune condizioni che devono essere rispettate.

Con riferimento ad un problema di programmazione matematica del tipo

$$\begin{aligned} \min & \mathbf{c}^\top \mathbf{x} \\ \mathbf{A}\mathbf{x} &\leq \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0}, \end{aligned}$$

in particolare, la condizione di ammissibilità delle basi (considerate durante l'esecuzione) è resa più restrittiva dal concetto di ammissibilità lessicografica.

Definizione 3.1 (Ammissibilità lessicografica)**.** Una selezione di colonne di $\mathbf{A}$ linearmente indipendenti $\mathbf{B}$ è *lessicograficamente ammissibile* se ogni riga della matrice $[\mathbf{B}^{-1}\mathbf{b}, \mathbf{B}^{-1}]$ è *lex-positiva.*[1]

Quando si renda necessario un cambiamento di soluzione base, fissata la base (lex-ammissibile) ed una variabile entrante candidata, la fase di selezione della variabile uscente va modificata opportunamente. Si deve infatti garantire che l'operazione di pivot sia conservativa rispetto alla lex-ammissibilità.

La strategia che si adotta, denotando con h l'indice della colonna entrante e r quello della colonna uscente, è

$$r = \arg\min_{i \ \ lex} \left\{ \frac{1}{\bar{A}_{ih}} [\bar{b}_i, (\mathbf{B}^{-1})_{i\cdot}] : \bar{A}_{ih} > 0 \right\} \tag{3.4}$$

con $\bar{b} = \mathbf{B}^{-1}\mathbf{b}$, $\bar{A}_{\cdot h} = \mathbf{B}^{-1}\mathbf{A}_h$ e $(\mathbf{B}^{-1})_{i\cdot}$ la i-esima riga di $\mathbf{B}^{-1}$.

Inoltre, affinché l'algoritmo termini con una soluzione ottima *univoca*, resta da applicare la perturbazione lessicografica in 2.28, che influisce sulla scelta della colonna entrante. Un'applicazione diretta di $\delta = (\delta_0, {\delta_0}^2, \ldots, {\delta_0}^n)$, tuttavia, è problematica per l'instabilità numerica che si presenta anche per n piuttosto modesti.

Un buon sistema per aggirare questa limitazione, risulta osservando che la forma del vettore dei costi residui [6] può essere espressa, riordinando secondo

[1] Vedi definizione 2.9

gli indici delle colonne in base, in forma di prodotto come

$$\begin{aligned}\bar{c}_h &= c_h - \mathbf{c}_B^\top B^{-1} A_h = c_h - \mathbf{c}_B^\top \bar{A}_h \\ &= [c_h, \mathbf{c}_B^\top] \begin{bmatrix} 1 \\ -\bar{A}_h \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{c}_{\{\beta_i<h\}} & c_h & \mathbf{c}_{\{\beta_i>h\}} \end{bmatrix} \begin{bmatrix} -\bar{A}_{\{i,h:i<h\}} \\ 1 \\ -\bar{A}_{\{i,h:i>h\}} \end{bmatrix}\end{aligned}$$

ovvero

$$\bar{c}_h = \mathbf{c}_{B\cup h}^\top \cdot \rho_{B\cup h} \tag{3.5}$$

Quindi, per il costo perturbato $f'(\mathbf{x}) = \left(\mathbf{c}^\top + \delta\right)\mathbf{x}$, una colonna h diventa ammissibile se $\bar{c}_h = \rho_{B\cup h}^\top \mathbf{c}_{B\cup h} + \rho_{B\cup h}^\top \delta < 0$, che equivale ad imporre

$$(\rho_{B\cup h}^\top \mathbf{c}_{B\cup h}, \rho_{B\cup h}^\top) <_{lex} 0 \tag{3.6}$$

L'implementazione in algoritmo 6 tiene conto delle considerazioni precedenti ed ottiene una soluzione base lessicograficamente minima [11].

Algoritmo 6 LEX_SIMPLEX

Input: $\mathbf{A}$, $\mathbf{b} \geq 0$, $\mathbf{c}$, $\{\beta_1, \ldots, \beta_m\}$ ▷ indici delle colonne in base (ammissibile)

1: `unbounded` $\leftarrow$ `false`
2: `optimal` $\leftarrow$ `false`
3: **while** `not(optimal or unbounded)` **do** ▷ Loop principale
4: $B \leftarrow [A_{\beta_1}|\ldots|A_{\beta_m}]$ ▷ Base
5: $\mathbf{c}_B^\top \leftarrow (c_{\beta_1}, \ldots, c_{\beta_m})$
6: **for all** $h \notin \{\beta_1, \ldots, \beta_m\}$ **do** ▷ Ricerca tra le variabili fuori base
7: $\bar{A}_h \leftarrow B^{-1}A_h$
8: **for all** $i \in \{\beta_1, \ldots, \beta_m\}$ **do**
9: $\rho_{ih} \leftarrow -\bar{A}_{ih}$
10: **end for**
11: $\rho_{hh} \leftarrow 1$
12: $\bar{c}_h \leftarrow (\rho^\top \mathbf{c}_{B\cup h}, \rho)$ ▷ Composizione vettore eq. 3.6
13: **end for**
14: **if** $\nexists \bar{c}_h <_{lex} 0$ **then** ▷ Verifica condizioni lex-ottimalità
15: `optimal` $\leftarrow$ `true`
16: **else**
17: $h \leftarrow \arg\min\{h : \bar{c}_h < 0\}$
18: $\bar{b} \leftarrow B^{-1}\mathbf{b}$
19: **if** $\exists \bar{b}_i < 0$ **then** ▷ Verifica limitatezza del problema
20: `unbounded` $\leftarrow$ `true`
21: **else** ▷ Ricerca variabile uscente
22: $r \leftarrow \arg\min_{i}{}_{lex} \left\{ \frac{1}{\bar{A}_{ih}} [\bar{b}_i, (\mathbf{B}^{-1})_{i\cdot}] : \bar{A}_{ih} > 0 \right\}$
23: $\beta_r \leftarrow h$
24: **end if**
25: **end if**
26: **end while**
27: **if** `optimal` **then**
28: $\mathbf{x} \leftarrow B^{-1}\mathbf{b}$ ▷ Soluzione ottima
29: $z \leftarrow \mathbf{c}^\top \mathbf{x}^\star$ ▷ Valore della soluzione ottima
30: **return** $z, \mathbf{x}$
31: **end if**

3.3 Hungarian

La seguente implementazione tabulare si basa su [4]. Si definiscono in anticipo alcune variabili:

- $Q \in \{\mathbb{R}_{\geq 0} \cup \{-\infty\}\}^{N \times M}$: matrice rettangolare delle priorità con coefficienti finiti positivi (in caso di priorità negative basta aggiungere un offset adeguato). Gli elementi che la compongono sono:

$$q_{ij} = \begin{cases} p_{ij} & \text{se } h_{ij} \in \mathcal{H} \\ -\infty & \text{altrimenti} \end{cases}$$

- $R \in \{\mathbb{R}_{\geq 0} \cup \{\infty\}\}^{N \times M}$ priorità *residue*.

$$r_{ij} = u_i + v_j - q_{ij}$$

  Gli elementi $r_{ij} = 0$ identificano gli archi del sottografo $\mathcal{G}_0$ su cui ricercare un matching (non pesato) di cardinalità massima.

- 0^* archi del sottografo $\mathcal{G}_0$ selezionati per il matching.

- $0'$ archi del sottografo $\mathcal{G}_0$ candidati a far parte di un cammino aumentante.

- c_{row} e C_{col} vettori booleani, di dimensione rispettivamente N ed M, i cui elementi contrassegnano righe e colonne *coperte*. c_{row} e C_{col} evolvono tenendo coperti gli 0^* e si rivelano utili ad escludere alcuni vertici durante la ricerca e nell'aggiornamento dell'etichettatura (u, v).

Un esempio di applicazione dell'algoritmo ungherese, relativo ad una matrice delle priorità

$$Q = \begin{pmatrix} 4 & 3 & 1 & 2 \\ 8 & -\infty & 2 & 4 \\ 12 & -\infty & 3 & 6 \end{pmatrix}$$

è riportato in figura 3.5.

L'assegnazione data dagli accoppiamenti $\mathcal{M} = \{(a_1, t_2), (a_2, t_4), (a_3, t_1)\}$ identifica la soluzione ottima con valore della funzione obiettivo

$$f(\mathcal{M}) = q_{1,2} + q_{2,4} + q_{3,1} = 3 + 4 + 12 = 19$$

.

Algoritmo 7 HUNGARIAN

Input: **Q** ▷ Matrice dei pesi
Output: *starred* ▷ Assegnazione finale
$u_i \leftarrow \max_j(q_{ij})$ ▷ Labelling iniziale
$v_j \leftarrow 0$
$r_{ij} \leftarrow u_i + v_j - q_{ij}$
$C_{row}, C_{col}, starred, primed \leftarrow \emptyset$
$complete \leftarrow$ `false`
while `not`(*complete*) **do** ▷ Loop principale:
    $(r, c) \leftarrow$ `findFreeZero()` ▷ Ricerca zero libero
    if $(r, c) = \emptyset$ **then**
        `expandGraph`() ▷ Espansione del grafo modificando il labellig
    else
        `prime`(r, c)
        if $\exists (i, j) \in starred : i = r$ **then**
            $C_{row} \leftarrow C_{row} \cup \{r\}$
            $C_{col} \leftarrow C_{col} \setminus \{c\}$
        else
            $P \leftarrow$ `augmentPath`(r, c) ▷ Ricerca cammino aumentante
            $starred \leftarrow starred \setminus (P \cap starred) \cup (P \cap primed)$
            $C_{row}, C_{col}, primed \leftarrow \emptyset$
            $C_{col} \leftarrow \{c : (r, c) \in starred\}$
        end if
    end if
    if $|starred| = N$ **then** ▷ Verifica ottimalità assegnazione:
        $complete \leftarrow$ `true`
    end if
end while
return

Algoritmo 8 `findFreeZero`

Input: $\mathbf{R}, C_{row}, C_{col}$
Output: (r, c)
for all $(i, j) : i \notin C_{row} \wedge j \notin C_{col}$ **do** ▷ Ricerca negli elementi scoperti
    if $r_{ij} = 0$ **then** ▷ Se (i, j) è un arco di $\mathcal{G}_0$:
        $(r, c) \leftarrow (i, j)$
        return (r, c) ▷ Indici del primo 0 libero trovato
    end if
end for
return $\emptyset$

Algoritmo 9 `expandGraph`

Input: $\mathbf{R}, C_{row}, C_{col}, u, v$

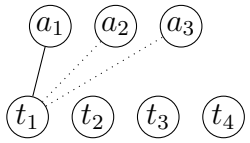
$\theta \leftarrow \min_{(i,j)}\{r_{ij} : i \notin C_{row} \wedge j \notin C_{col}\}$ ▷ Ricerca minimo elemento scoperto
if $\theta = \emptyset$ `or` $\theta = \infty$ **then**
    $complete \leftarrow$ `true` ▷ Massima cardinalità dell'assegnazione raggiunta
else
    for all $(i,j) : i \notin C_{row} \wedge j \notin C_{col}$ **do**
        $r_{ij} \leftarrow r_{ij} - \theta$ ▷ Aggiunta nuovi archi a $\mathcal{G}_0$
    end for
    for all $(i,j) : i \in C_{row} \wedge j \in C_{col}$ **do**
        $r_{ij} \leftarrow r_{ij} + \theta$ ▷ Rimozione archi non più utilizzabili
    end for
    for all $i : i \notin C_{row}$ **do**
        $u_i \leftarrow u_i - \theta$ ▷ Aggiornamento u
    end for
    for all $j : j \notin C_{col}$ **do**
        $v_j \leftarrow v_j + \theta$ ▷ Aggiornamento v
    end for
end if
return

Algoritmo 10 `augmentPath`

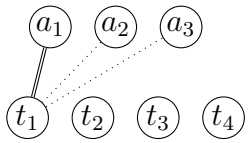
Input: $(r,c), primed, starred$
Output: P ▷ Cammino aumentante per $\mathcal{G}_0$

$P \leftarrow \{(r,c)\}$ ▷ Primo arco (libero)
$r \leftarrow i : (i,c) \in starred$ ▷ Ricerca arco (assegnato) co-incidente al task t_c
while $r \neq \emptyset$ **do**
    $P \leftarrow P \cup \{(r,c)\}$
    $c \leftarrow j : (r,j) \in primed$ ▷ Ricerca arco (libero) co-incidente all'agente a_r
    $P \leftarrow P \cup \{(r,c)\}$
    $r \leftarrow i : (i,c) \in starred$ ▷ Ricerca arco (assegnato) co-incidente al task t_c
end while
return P ▷ Cammino alternate e aumentante per $\mathcal{G}_0$

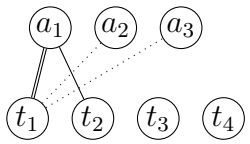
	0	0	0	0
4	0’	1	3	2
8	0	-	6	4
12	0	-	9	6



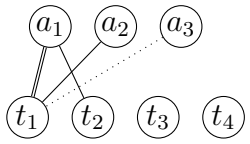
	0	0	0	0
4	0*	1	3	2
8	0	-	6	4
12	0	-	9	6



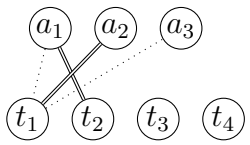
	1	0	0	0
3	0*	0’	2	1
7	0	-	5	3
11	0	-	8	5



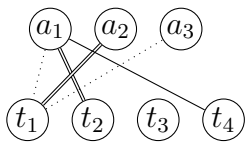
	1	0	0	0
3	0*	0’	2	1
7	0’	-	5	3
11	0	-	8	5



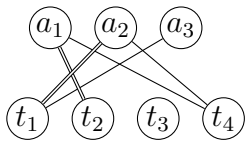
	1	0	0	0
3	0	0*	2	1
7	0*	-	5	3
11	0	-	8	5



	2	1	0	0
2	0	0*	1	0’
6	0*	-	4	2
10	0	-	7	4



	4	1	0	0
2	2	0*	1	0’
4	0*	-	2	0’
8	0	-	5	2



	4	1	0	0
2	2	0*	1	0
4	0	-	2	0*
8	0*	-	5	2

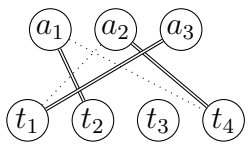


Figura 3.5: Applicazione Hungarian Algorithm su un’istanza di esempio. A sinistra la rappresentazione tabulare dello stato dell’algoritmo, a destra il grafo corrispondente. I vincoli sono indicati con tratti via via più marcati (tratteggio, continuo, doppio) a seconda che l’arco appartenga al grafo ridotto $\mathcal{G}$, sia un candidato nel calcolo di un cammino aumentante ($0'$) oppure un’assegnazione (0^*).

Si nota facilmente come, da un'assegnazione parziale, tramite un cammino aumentante si ottenga un'assegnazione di cardinalità strettamente superiore, fino alla massima.

3.4 Evoluzione dinamica

Quanto visto sin qui si occupa di risolvere in maniera ottima un'istanza statica di task assignment: gli agenti ed i task sono dati e fissati durante l'esecuzione dell'algoritmo. In un contesto reale il numero di agenti e il *pool* (l'insieme) dei task a cui gli agenti attingono sarà un'entità variabile e dinamica, infatti:

- Nuovi task vengono generati con il passare del tempo o in conseguenza al verificarsi di particolari eventi.
- I task che vengono assegnati richiedono, in genere, un tempo finito per essere completati. Al termine dell'esecuzione vanno rimossi dal sistema (o contrassegnati come completati) e non partecipano ad ulteriori assegnazioni.[2]
- Si può prevedere che i task abbiano un tempo di attesa massimo, prima di essere considerati obsoleti, anche nel caso di non assegnazione. Scaduto tale intervallo il task viene scartato dal sistema (*dropping*). Questo accorgimento, oltre ad essere un ragionevole parametro di progetto del sistema, previene la crescita incontrollata del *pool* (e quindi la saturazione del sistema), in particolare nei casi in cui il tasso di nascita di nuovi task supera la capacità di esecuzione complessiva degli agenti.
- È sempre da tenere in considerazione la possibilità che uno o più agenti subiscano guasti o risultino *offline* per un periodo rilevante, variando quindi la struttura di comunicazione e la capacità esecutiva del sistema. Altresì in alcuni casi è da contemplare l'inserimento di nuovi agenti nel sistema al *runtime*.

L'ipotesi di lavoro che si pone è che la dinamica di nascita e morte dei task (e di ingresso e uscita degli agenti dal sistema) sia "lenta" rispetto alla velocità di convergenza dell'algoritmo di assegnazione, ovvero che non ci siano variazioni nell'insieme dei task durante l'esecuzione del constraint consensus.

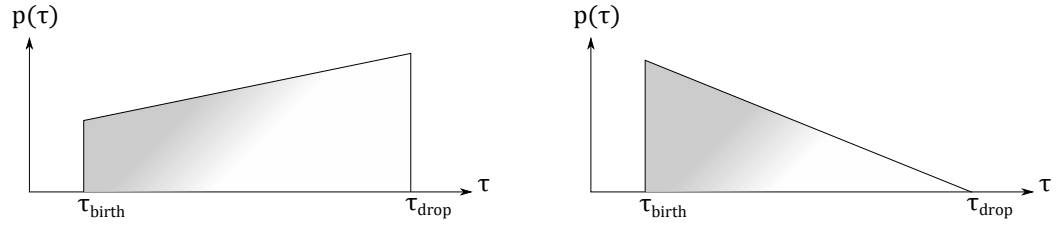
Un altro aspetto da tenere in considerazione è il tempo di attesa dei task prima di essere assegnati. In genere (ma non necessariamente) è desiderabile che un task, anche se di bassa priorità, non venga rinviato fino a diventare inutile e quindi scartato dal sistema, ma si ricerca un compromesso tra massimizzazione delle priorità (dei task) in esecuzione e minimizzazione del task

[2] Alcuni task, a bassa priorità, possono comunque essere definiti come non completabili. Si pensi, ad esempio ad attività che vengono eseguite di default quando un agente è libero: task di tipo *idle*, *listening*, ecc... a seconda dell'applicazione.

dropping. In un picco di generazione di task che satura la capacità di esecuzione degli agenti, potrebbe essere preferibile ritardare leggermente anche i task ad alta priorità per impedire un eccessivo tasso di scarti.

Specularmente, invece, posso esserci applicazioni in cui l'utilità di un task è tanto maggiore quanto più in fretta viene posto in esecuzione, mentre con il passare del tempo perde gradualmente interesse.

Le due situazioni sono esemplificate nelle curva di priorità dinamica rispettivamente in figura 3.6a e 3.6b. La prima viene ripresa ed ampliata leggermente con un esempio in sezione 4.



(a) Dinamica con peso positivo del tempo di attesa.

(b) Utilità decrescente con l'età del task.

Figura 3.6: Possibili dinamiche della priorità di un task.

Aggiornamento dello stato del sistema

All'istante iniziale, tipicamente, si presume che gli n agenti conoscano solo gli n task *idle* (o assimilabili) che li coinvolgono, presenti come vincoli in `Hloc`, e che questi compongano complessivamente l'istanza da risolvere.

Man mano che il sistema evolve, il verificarsi di eventi identificati dagli agenti (o per iniezione da un supervisore esterno) comporta la formazione di nuovi task, che vanno comunicati dal nodo di creazione al resto del sistema. Lo stesso vale per i task la cui esecuzione viene completata: tale evento va reso noto agli altri agenti, in modo da permetterne la rimozione e segnalare la necessità di riavvio dell'algoritmo di assegnazione. Inoltre un sistema di rilevamento dei guasti, del diametro istantaneo del grafo di comunicazione (in caso di topologia variabile) e in generale di ingresso o uscita di agenti, deve essere affiancato alle comunicazioni definite per lo scambio di vincoli. Una soluzione (non necessariamente l'unica, né la migliore, a seconda dell'applicazione) è quella di implementare uno scambio di informazioni periodico (di tipo *hearthbeat*) per verificare lo stato dei nodi, affiancato a pacchetti in broadcasting per la notifica di eventi come nascita o morte di task.

4 Scenario di esempio: video sorveglianza

In questa sezione si illustra un possibile scenario applicativo del task assignment via contraint consensus, per quanto semplice, utile a chiarirne la dinamica e le peculiarità.

Si consideri, a titolo di esempio, il problema dell'assegnazione in un contesto di videosorveglianza. In particolare si ipotizzi che una certa area debba essere sottoposta a sorveglianza da parte di una rete di videocamere *Pan Tilt Zoom* (PTZ).[1] Queste, si suppone, dispongono di una certa capacità computazionale, spazio di archiviazione locale e siano collegate tra di loro su rete TCP/IP (wireless o cablata a seconda dei requisiti progettuali) e secondo una certa topologia di comunicazione, assunta statica solo per semplicità.

Per quanto riguarda l'area da controllare, si definisca un partizionamento univoco della stessa, ad esempio sfruttando le intersezioni tra le visuali dei singoli agenti (*Field Of View*, FOV) in caso di agenti a posizione fissa, oppure a griglia, di dimensioni opportune, nel caso alcuni agenti siano dotati di una certa mobilità che non consente di fissare a priori un FOV statico (si pensi ad esempio ad un drone in volo).

Un'istanza di esempio è rappresentata in figura 4.1: sono considerati quattro agenti (camere) a_1, a_2, a_3, a_4 collegati secondo un grafo di comunicazione $\mathcal{G}_{com}$ (statico) ad anello. L'area di interesse è partizionata in otto zone $(z_1, \ldots, z_8)$ distinte, di pertinenza degli agenti in questo modo: a_1 copre $\{z_1, z_2, z_3\}$, a_2 copre $\{z_3, z_4, z_5\}$, a_3 copre $\{z_5, z_6, z_7\}$ e a_4 copre $\{z_1, z_7, z_8\}$.

In questo scenario è possibile identificare dei semplici task svolgibili dagli agenti. Alcuni esempi:

- *Patrolling*: dato un singolo elemento della partizione d'area (una precisa *locazione*) si richiede che almeno un agente tra quelli fisicamente in grado di coprirla, che la contengono, cioè, nel proprio FOV osservi la

[1] Si pensi in genere a strutture aeroportuali, scolastiche, ospedaliere o istituzionali, magazzini, parcheggi, complessi industriali..

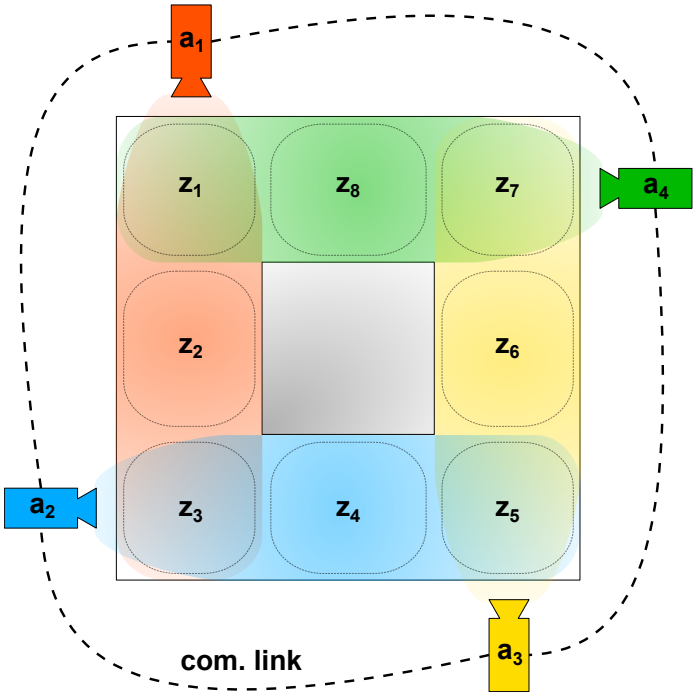


Figura 4.1: Possibile scenario di applicazione: videosorveglianza.

zona per rilevare eventi di interesse o semplicemente per disporre di una registrazione della scena che si svolge.
In alternativa il patrolling può essere definito per un preciso agente, piuttosto che ad un'area, ovvero si impone all'agente di mantenere sotto controllo tutte le aree che è in grado di coprire.

- *Tracking*: nota la presenza di un evento in corso in una particolare zona, si desidera che almeno un agente la riprenda inseguendo, per esempio, un soggetto in movimento con il massimo dettaglio possibile.

- *Funzionamento manuale*: si impone che un particolare agente segua direttamente i comandi di un operatore, per quanto riguarda il brandeggio (movimento di Pan-Tilt) e lo zoom.

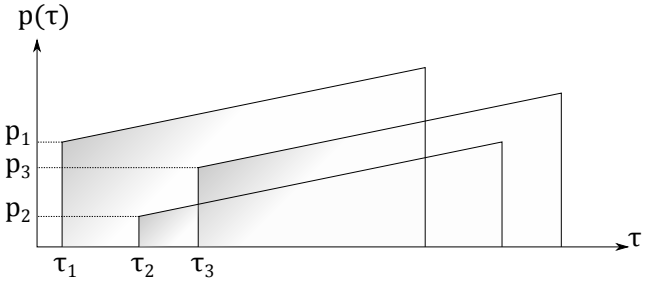
Ogni task generato sarà dotato di una *priorità intrinseca* determinata dal tipo particolare di task (per esempio il funzionamento manuale dovrebbe essere una tipologia a massima priorità, essendo comandato direttamente da un supervisore) ed eventualmente da sotto-casi specifici: se l'evento da tracciare è potenzialmente pericoloso per l'incolumità dei presenti o minaccia strutturalmente l'edificio, risulta più urgente di un parcheggio in zona vietata. Altresì, la priorità che si desidera massimizzare potrebbe essere una combinazione tra

la priorità intrinseca, come definita sopra, ed un termine tempo-variante che tenga conto dell'anzianità del task, del suo tempo di vita, in modo che un task a bassa priorità intrinseca abbia comunque un'alta probabilità di essere eseguito dopo un certo periodo, di attesa anche in presenza di molti task ad alta priorità. Una possibile realizzazione è indicata nella 4.1.

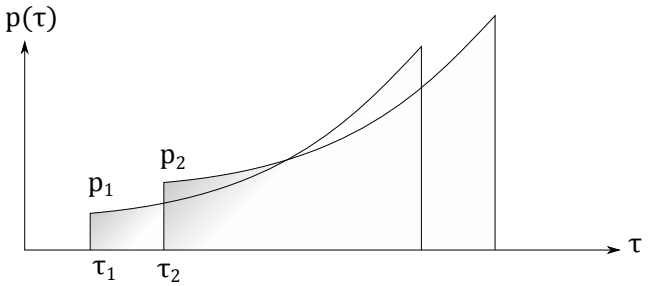
$$p_j(\tau) = \alpha p_j + (1 - \alpha)(\tau - \tau_j) \tag{4.1}$$

dove τ e τ_j indicano, rispettivamente, l'istante corrente e di nascita del task t_j e $0 < \alpha \leq 1$ un coefficiente modulante il peso della tipologia di task sul tempo di attesa, stabilito a priori.

Si noti che una legge lineare è necessaria per preservare l'ordinamento relativo tra i task. In tal modo la differenza di priorità tra gli stessi è fissata al momento della loro nascita: possono essere "scavalcati" solo da nuovi task, come rappresentato in figura 4.2a. In caso contrario si assisterebbe a molteplici variazioni di ordine tra i task (figura 4.2b), potenzialmente in grado di perturbare le assegnazioni.



(a) Variazione lineare: mantiene l'ordine reciproco tra i task.



(b) Variazione non lineare: ordine reciproco tra i task instabile.

Figura 4.2: Esempio di andamento temporale delle priorità.

Infine, in un contesto reale si suppone che dopo un certo tempo un task non eseguito perda di interesse e venga quindi scartato dal sistema (*task dropping*).

4.1 Evoluzione

Si consideri il sistema di videosorveglianza già introdotto in figura 4.1 con una particolare istanza di task, come rappresentato in figura 4.3, in cui sono dislocati 7 task così definiti:

- $t_1, \ldots, t_4$: task di patrolling a bassa priorità $p = 1$, localizzati sulle zone z_2, z_4, z_6, z_8 e quindi di competenza esclusiva, rispettivamente, di a_1, a_2, a_3, a_4;
- t_5, t_7: task di patrolling a media priorità ($p = 2$) localizzati in z_1 e z_3;
- t_6: task di tracking di un evento, ad alta priorità ($p = 3$), in zona z_3.

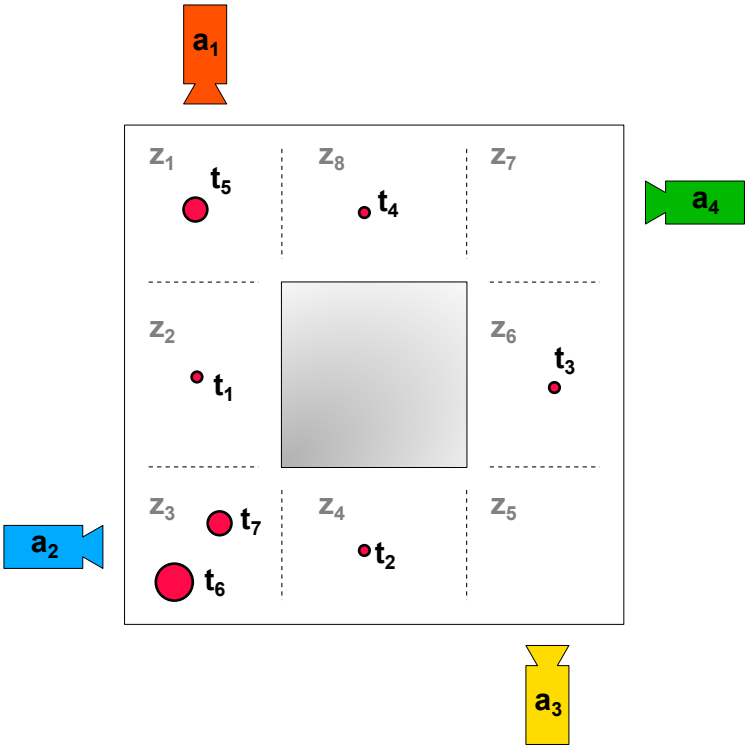


Figura 4.3: Esempio con $M = 7$ task attivi. La priorità dei task è rappresentata dalla dimensione del cerchio associato (maggior priorità, maggior diametro).

La matrice di adiacenza associata all'istanza è

$$\hat{\mathbf{V}}: \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{array} \begin{array}{c} \begin{array}{ccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \end{array} \\ \begin{pmatrix} 1 & -\infty & -\infty & -\infty & 2 & 3 & 2 \\ -\infty & 1 & -\infty & -\infty & -\infty & 3 & 2 \\ -\infty & -\infty & 1 & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & 1 & 2 & -\infty & -\infty \end{pmatrix} \end{array}$$

4.1.1 Aggiunta di nuovi task

La rappresentazione seguente riporta l'evoluzione delle informazioni che ogni agente possiede sullo stato del sistema e le relative manipolazioni attuate al fine di giungere ad un'assegnazione ottima condivisa.

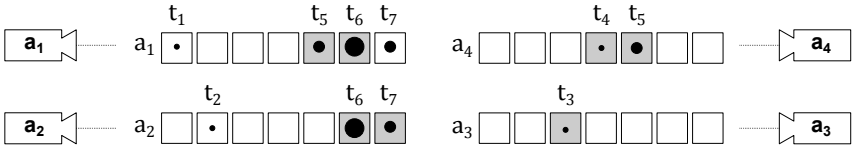


Figura 4.4: STEP 0: Situazione iniziale.

Nella fase iniziale (figura 4.4) ogni agente conosce solo le informazioni riguardanti i task che è in grado di svolgere. I vincoli (la possibile coppia task-agente) sono rappresentati graficamente come cerchi di dimensione proporzionale alla priorità, per una più immediata leggibilità. Partendo dalle informazioni locali, l'agente calcola una base per il problema (vincoli con sfondo grigio).

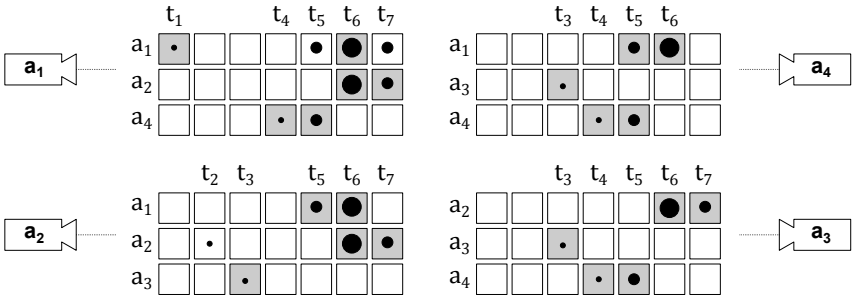


Figura 4.5: STEP 1

Dopo un primo step di comunicazione (figura 4.5), gli agenti sono a conoscenza, oltre che delle informazioni iniziali, anche delle basi calcolate dai nodi adiacenti (sul grafo di comunicazione), ed in funzione delle nuove informazioni aggiornano la propria base locale.

Più in dettaglio, in figura 4.6, è schematizzata logica di aggiornamento per il nodo a_2. L'agente mantiene ad ogni iterazione i suoi vincoli locali $H^{[2]}_{LOC}$ e costruisce un problema di ottimizzazione astratta $\mathcal{H}^{[2]}$ unendo ad essi i vincoli esterni $H^{[2]}_{REM}$ (l'unione delle basi comunicate dai vicini $B^{[1]}, B^{[3]}$) e la propria base precedentemente calcolata $B^{[2]}$ (vincoli in grigio):

$$\mathcal{H}^{[2]} = H^{[2]}_{LOC} \cup H^{[2]}_{REM} \cup B^{[2]}$$

A questo punto è in grado di calcolare per sè una nuova base che andrà poi a notificare agli altri nodi.

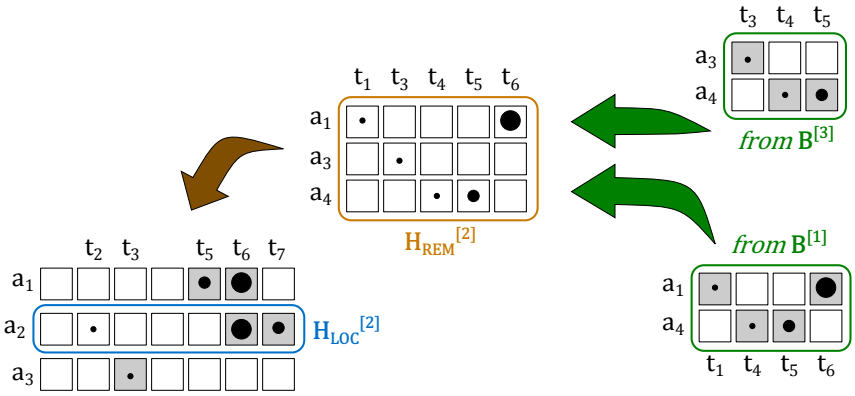


Figura 4.6: STEP 1.5: Aggiornamento del problema locale

Iterando il procedimento, come riportato in figura 4.8, si giunge ad uno stato in cui tutti gli agenti hanno calcolato la stessa base, che si è visto essere ottima, e su di essa possono calcolare l'effettiva assegnazione per iniziare lo svolgimento dei task, cui corrisponde lo schema in figura 4.7.

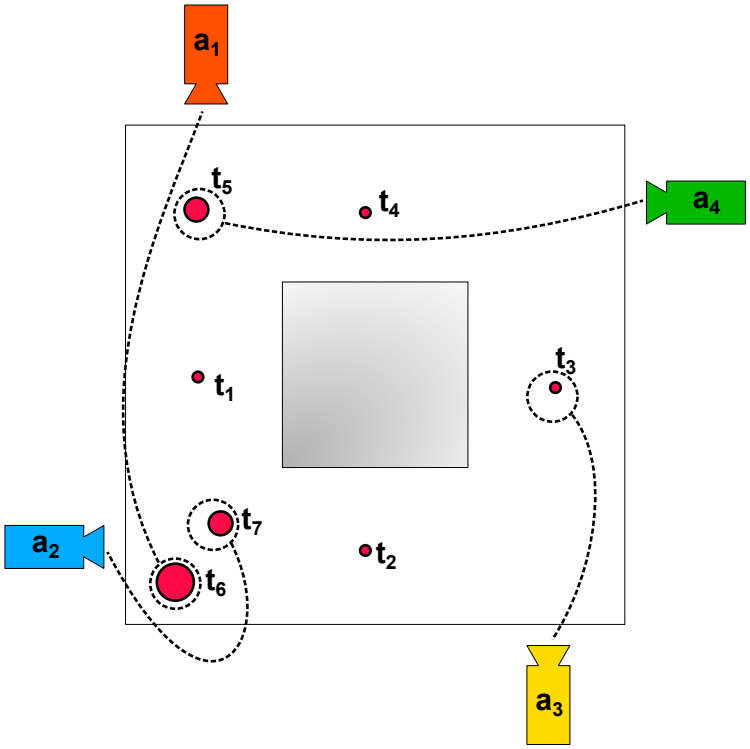
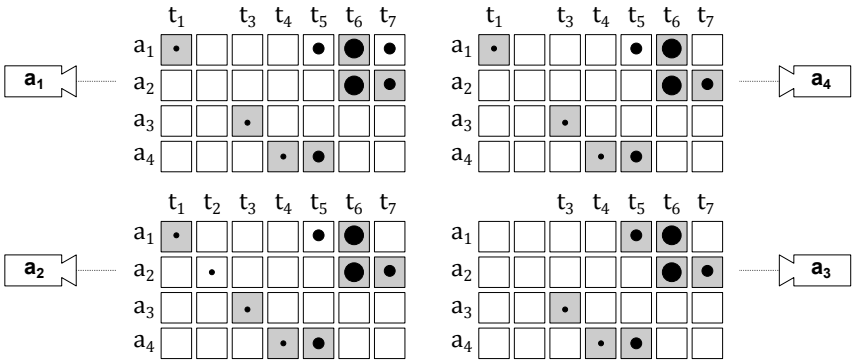
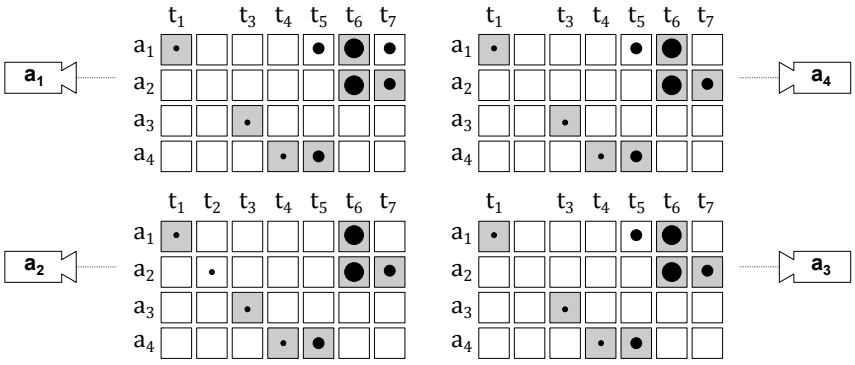


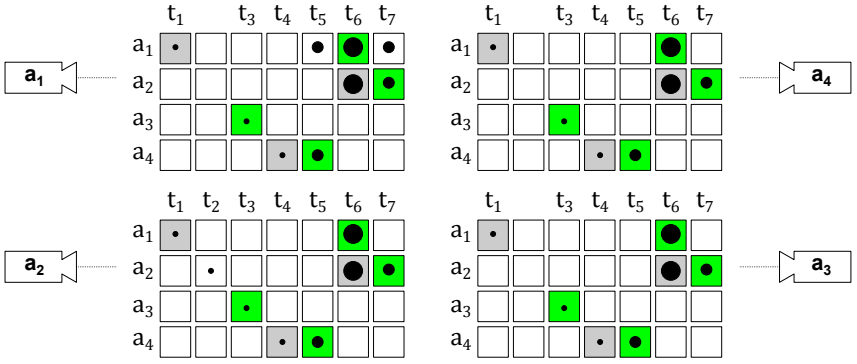
Figura 4.7: Assegnazione finale ottima (corrispondente allo STEP 4 in fig. 4.8c).



(a) STEP 2



(b) STEP 3



(c) STEP 4: Convergenza delle basi ed assegnazione.

Figura 4.8: STEP 2–4

4.1.2 Termine di un task

A partire dalla situazione raggiunta allo step finale del precedente esempio, si suppone che dopo un certo intervallo di tempo l'agente a_2 abbia completato il task che stava svolgendo, t_7, e comunichi alla rete l'evento.

In risposta all'aggiornamento del sistema gli agenti rimuovono dal proprio *pool* locale il task contrassegnato come concluso (figura 4.9) e danno il via ad un nuovo ciclo di consenso, per riassegnare i compiti rimanenti in maniera ottima.

Come si può osservare dal risultato finale (figura 4.13), l'agente a_1 cede il task che stava eseguendo ad a_2 e passa al task a bassa priorità t_1: tra le soluzioni con funzione obiettivo massima è scelta la prima in senso lessicografico.

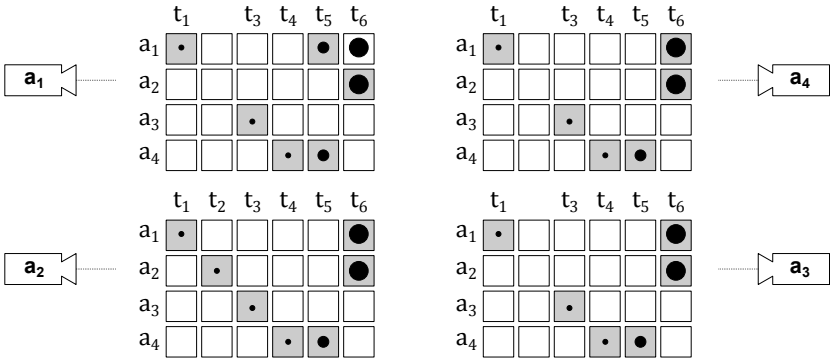


Figura 4.9: STEP 0

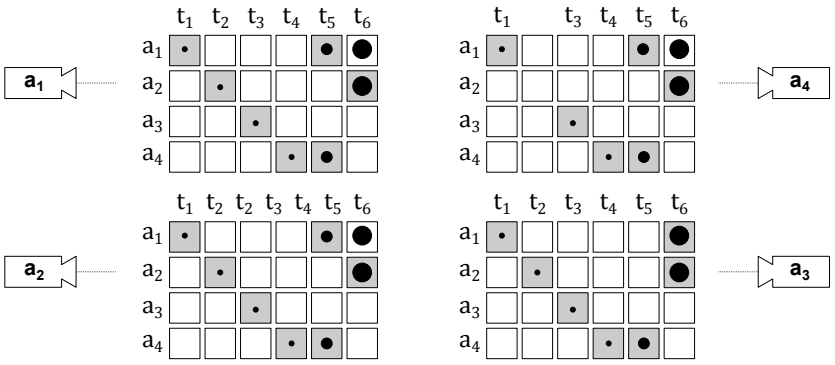


Figura 4.10: STEP 1

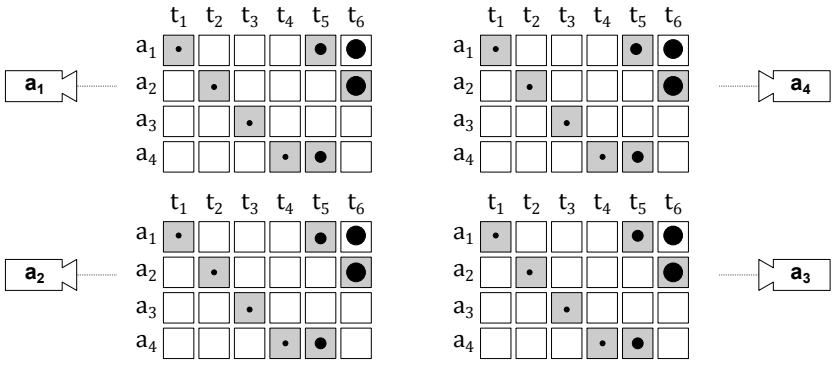


Figura 4.11: STEP 2

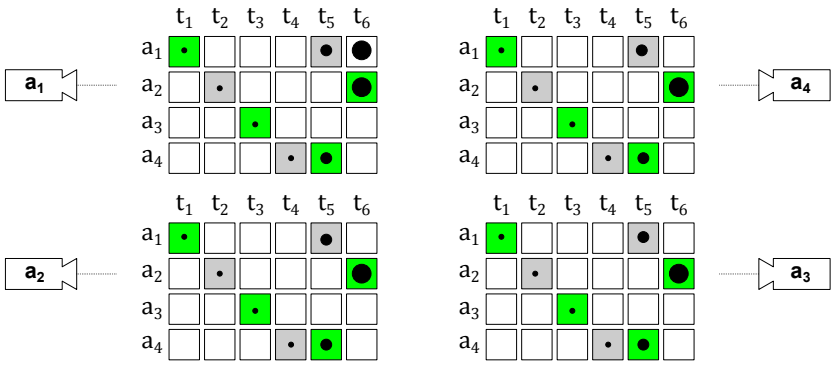


Figura 4.12: STEP 3: Convergenza e nuova assegnazione.

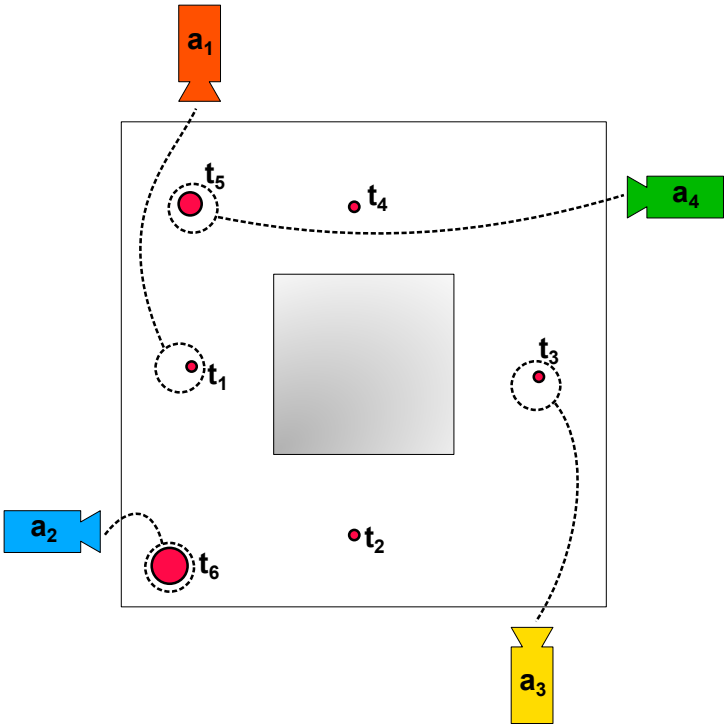


Figura 4.13: Soluzione ottima aggiornata dopo la conclusione di t_2 (corrispondente allo STEP 3 in fig. 4.12).

4.1.3 Guasto

Un ultimo scenario interessante da indagare è cosa accade quando un agente soffre di un guasto o viene per qualche motivo escluso dalla rete di comunicazione. In questo caso gli altri agenti sono in grado di intercettare l'evento non ricevendo più comunicazioni periodiche dal nodo guasto ed adeguano l'assegnazione di conseguenza.

Proseguendo con l'esempio dalla situazione raggiunta in figura 4.13, si ipotizzi che per qualche motivo l'agente a_2 non sia più raggiungibile.

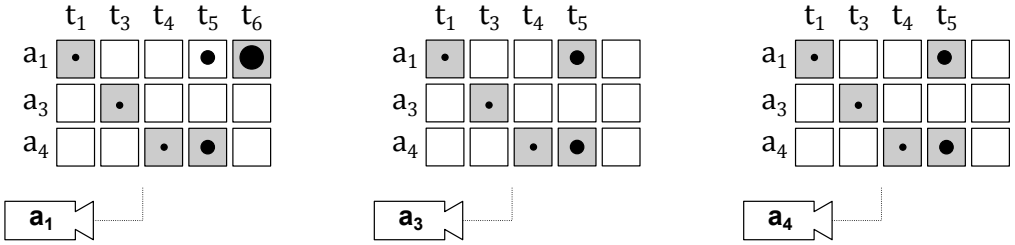


Figura 4.14: STEP 0: Rimozione dell'agente a_2.

Gli agenti rimanenti, rilevato il guasto, rimuovono completamente i dati relativi all'agente non raggiungibile (figura 4.14) e avviano nuovamente una sessione di constraint consensus tra loro, per ottenere una soluzione ottima al problema ridotto, con un set di agenti $\mathcal{A}_{red} = \{a_1, a_3, a_4\}$ ed i soli task compatibili $\mathcal{T}_{red} = \{t_1, t_3, t_4, t_5, t_6\}$.

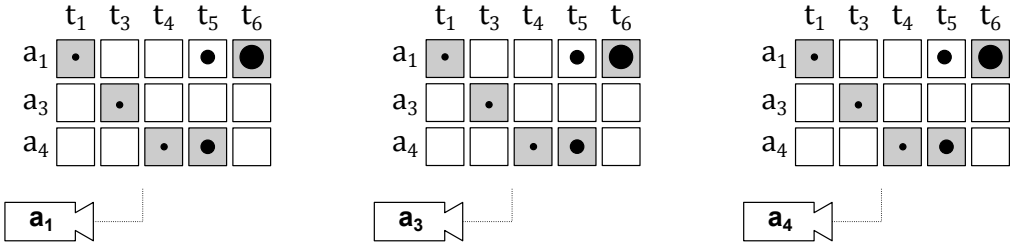


Figura 4.15: STEP 1

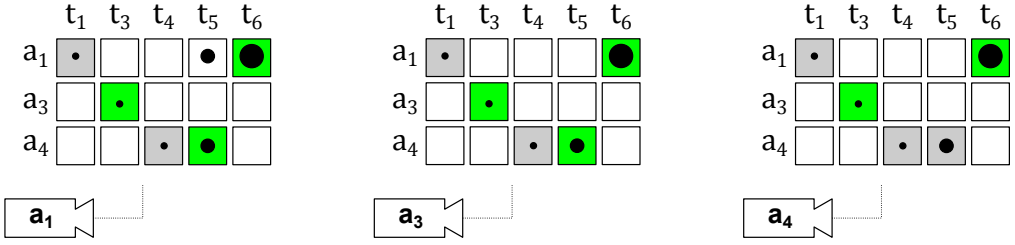


Figura 4.16: STEP 2: convergenza e nuova assegnazione ottima.

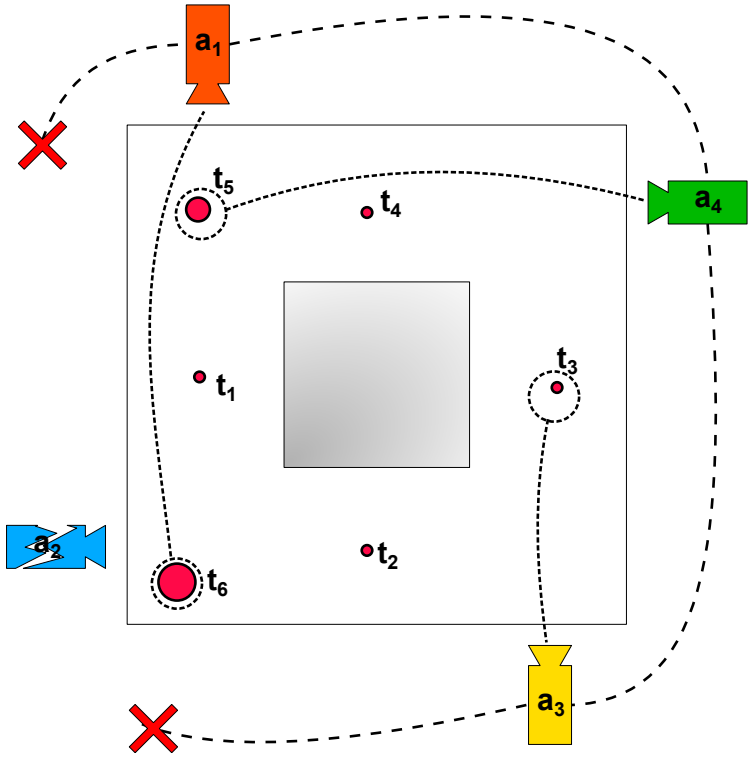


Figura 4.17: Nuova soluzione ottima per il sistema ridotto $\{a_1, a_3, a_4\}$ (corrispondente allo STEP 2 in fig. 4.16).

Si osservi che, per come è stato definito il vettore sei vincoli locali $H_{LOC}^{[i]}$, esso contiene tutti i task compatibili con l'agente a_i. Se si verifica un guasto, la perdita di informazioni immagazzinate nell'agente offline, determina al più l'eliminazione dal sistema rimanente dei soli task che sarebbe in grado da solo di eseguire. Rifacendosi all'esempio appena discusso, il sottosistema dato dagli agenti funzionanti $\mathcal{A}_{red} = \{a_1, a_3, a_4\}$ ha perso traccia del solo task t_2, di competenza esclusiva dell'agente guasto, quindi inutile ai fini dell'ottimalità per gli agenti rimanenti.

5 Simulazioni

Nella presente sezione sono riportate alcune simulazioni numeriche utili a comprendere le prestazioni effettive e la scalabilità degli approcci descritti in precedenza.

Setup

L'impianto simulativo realizza un sistema di N agenti autonomi in grado di dialogare tra loro in maniera semi-asincrona[1] secondo un grafo di comunicazione $\mathcal{G}_{com}$ statico, definito, nello specifico, da una matrice circolante $C_{com} \in \{0,1\}^{N\times N}$.

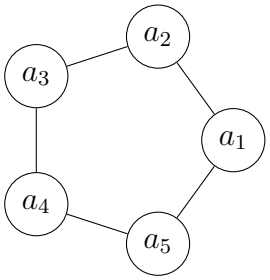


Figura 5.1: Esempio di connessione ad anello bidirezionale per $N = 5$.

Ove non diversamente specificato si considera una topologia di collegamento ad anello, in cui ogni nodo è connesso bidirezionalmente ai nodi precedente e successivo come rappresentato, per esempio, nel grafo in figura 5.1, la cui matrice di comunicazione risulta, quindi,

$$C_{com} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{5.1}$$

[1] È inteso che la comunicazione possa essere asincrona, purché sia possibile definire un limite superiore T_{COM} all'intervallo tra due aggiornamenti di stato consecutivi per ogni nodo. In simulazione si è imposto esattamente un aggiornamento di stato da parte degli agenti per ogni intervallo di durata T_{COM}.

Il set di task su cui effettuare l'assegnazione consiste di $M \geq N$ elementi generati tenendo conto delle seguenti ipotesi:

(i) La priorità di ogni singolo task è scelta in maniera casuale secondo una distribuzione discreta uniforme $\mathcal{U}([1, P_{max}], P_{max})$, con $P_{max} = M$ ove non diversamente specificato, ed è un valore intrinseco al task, ovvero non c'è differenza, in termini di incremento di utilità, tra esecuzione del task da parte di un agente a_i piuttosto che da a_j. La variabile di interesse è se il task sia o meno assegnato.[2]

(ii) La compatibilità con un agente, ovvero la capacità dello stesso di eseguire il task è determinata anch'essa in maniera aleatoria con probabilità costante:

$$P[(a_i, t_j) \text{ compatibili}] = P_{TA} \qquad \forall\ 1 \leq i \leq N, 1 \leq j \leq M$$

il tasso medio di task per agente P_{TA} è posto, salvo variazioni, a 50%.

La probabilità di compatibilità al punto (ii) indica che il numero effettivo di vincoli (o di archi del grafo associato) è, in media, $H = P_{TA}NM$ e, di conseguenza il numero medio di task disponibili per agente sarà $P_{TA}M$.

Grandezze d'interesse

Ciò che maggiormente interessa tenere sotto controllo, oltre ad avere un riscontro numerico di quanto teorizzato, è il modo in cui un sistema come quello considerato scala all'aumentare della dimensione.

La velocità di convergenza dell'algoritmo in termini di step di comunicazione necessari al raggiungimento di una soluzione comune è il dato primario su cui si vuole verificare l'impatto di variazioni del problema di riferimento quali:

- numero di agenti nel sistema;
- numero di task disponibili;
- ridotta capacità di comunicazione tra i nodi.

[2] La distribuzione discreta determina un'alta probabilità di avere più task con pari priorità, cui consegue una maggiore degenerazione del problema duale associato e quindi una condizione in generale più sfavorevole agli algoritmi di consenso. Lo stesso principio si applica all'indipendenza della priorità dall'agente che esegue il task.

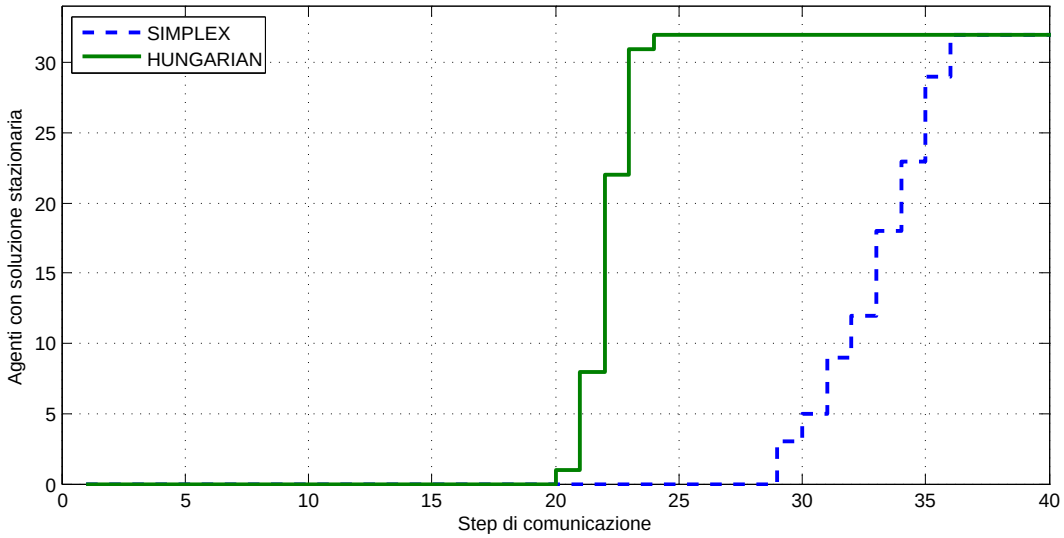
L'ottimalità della soluzione finale al problema di assegnazione (dal punto di vista delle priorità) è di fatto un vincolo al problema e non una misura delle prestazioni.

È di interesse, infine, accennare all'impatto sulle comunicazioni dovuto alla tipologia di algoritmo per il calcolo della base ed al numero di agenti.

5.1 Velocità di convergenza e scalabilità

In figura 5.2 è riportato un esempio di applicazione dell'algoritmo ad un sistema di $N = 32$ agenti per un'istanza di $M = 64$ task.[3] Sono poste a confronto le due implementazioni simplesso lessicografico e metodo ungherese per il calcolo dell'assegnazione e della base. In ascissa sono rappresentate le singole iterazioni dell'algoritmo, mentre l'ordinata indica il numero di agenti che ha raggiunto una base stabile (la soluzione ottima). In particolare, nell'intervallo tra un passo e il successivo, ogni agente ha avuto modo di inviare la propria base locale ai vicini nel grafo di comunicazione.

Come si osserva, per un certo numero di iterazioni (circa 20 per `HUNGARIAN` e 29 per `SIMPLEX`)i nodi computano basi variabili ed integrano l'informazione con i messaggi ricevuti dalla rete. Nel momento in cui un agente raggiunge una base stabile, essa rimane invariata, andando ad incrementare il numero di agenti che hanno concluso l'algoritmo; come prevedibile la curva risulta essere monotona non decrescente fino ad N agenti stabili.



N	M	P_{max}	P_{TA}
32	64	M	50%

Figura 5.2: Esempio di evoluzione (istanza singola) delle basi per 32 agenti.

La scelta dell'algoritmo per il computo della base va ad influire sulla velocità di convergenza, ma lascia inalterata l'evoluzione del consenso da un punto di

[3] Insieme al grafico sono richiamati i parametri della simulazione in forma tabellare, come numero N di agenti, numero M di task, e le grandezze definite nella sezione 5. Quando possibile, la tabella dei parametri è indicata anche nei grafici che seguono.

vista qualitativo.

È interessante indagare quali siano le grandezze che influiscono direttamente sulla velocità di convergenza, e quale la sensitività ad alcuni parametri simulativi scelti arbitrariamente.

In figura 5.3 sono rappresentati i gli step necessari alla convergenza per varie combinazioni di N ed M. Le priorità dei task sono estratte nell'intervallo $[0 \dots M]$ e $[0 \dots 100M]$.

Ciò che si osserva è che si ha correlazione tra il numero di agenti ed il tempo di convergenza, mentre non ci sono dipendenze significative dalla varianza delle priorità ed i tempi di convergenza.

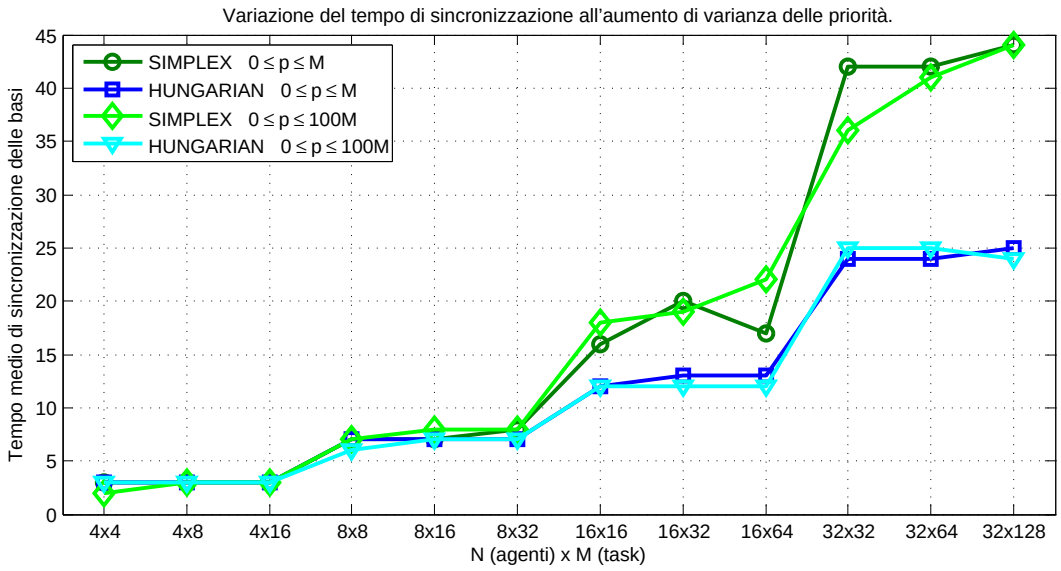


Figura 5.3: Sensibilità alla varianza della priorità ($P_{TA} = 50\%$).

Le figure 5.4 e 5.5 tengono conto di tre scenari differenziati da un diverso tasso di compatibilità medio tra gli agenti ed i task. A valori di P_{TA} alti corrisponde il caso di task poco differenziati, ovvero che hanno alta probabilità di essere compatibili con un gran numero di agenti. Viceversa al decrescere di P_{TA} ci si sposta verso la situazione di task compatibili con un insieme ristretto di agenti: si pensi, per esempio, al caso in cui agenti e task sono vincolati da una localizzazione spaziale ben precisa che non permette ai primi di svolgere task "lontani".

Anche in questo caso le curve sono quasi sovrapposte ed evidenziano una sostanziale insensibilità al parametro dal punto di vista della velocità di convergenza.

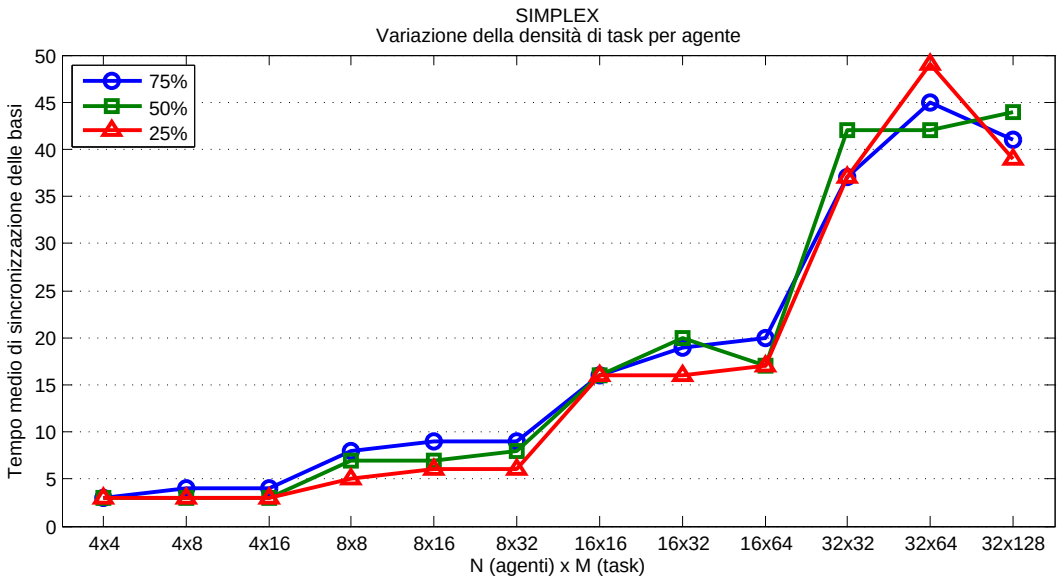


Figura 5.4: Sensibilità al numero di task per agente: implementazione con simplesso lessicografico ($P_{max} = M$).

L'andamento a gradini suggerisce anche che il numero di task disponibili M (quindi il livello di saturazione del sistema) sia parimenti ininfluente sul numero di iterazioni necessarie.

Tale tendenza risulta sostanzialmente confermata dalle simulazioni con un più ampio intervallo nel numero di task, mantenendo N costante, in figura 5.6.

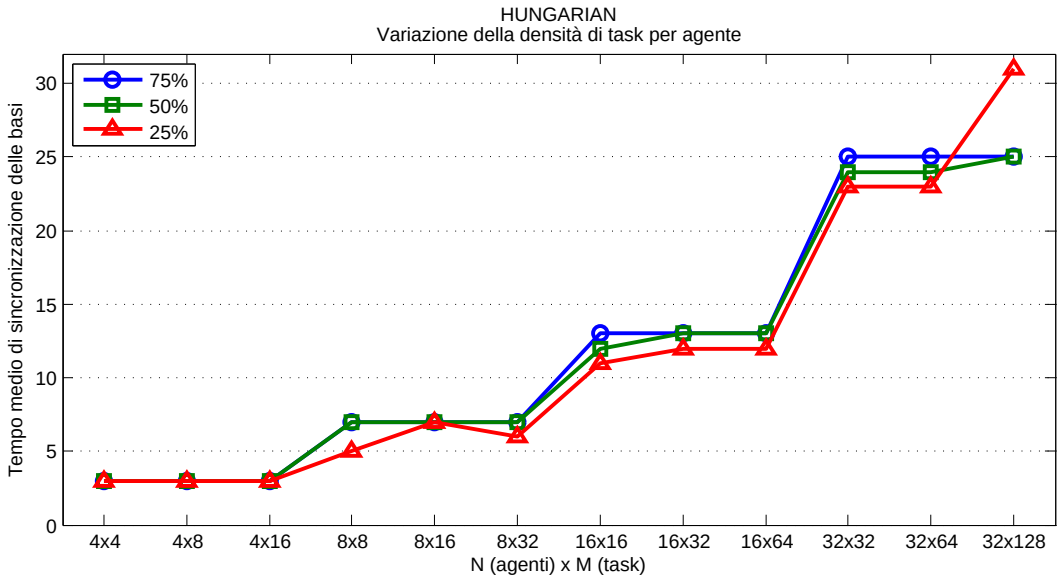
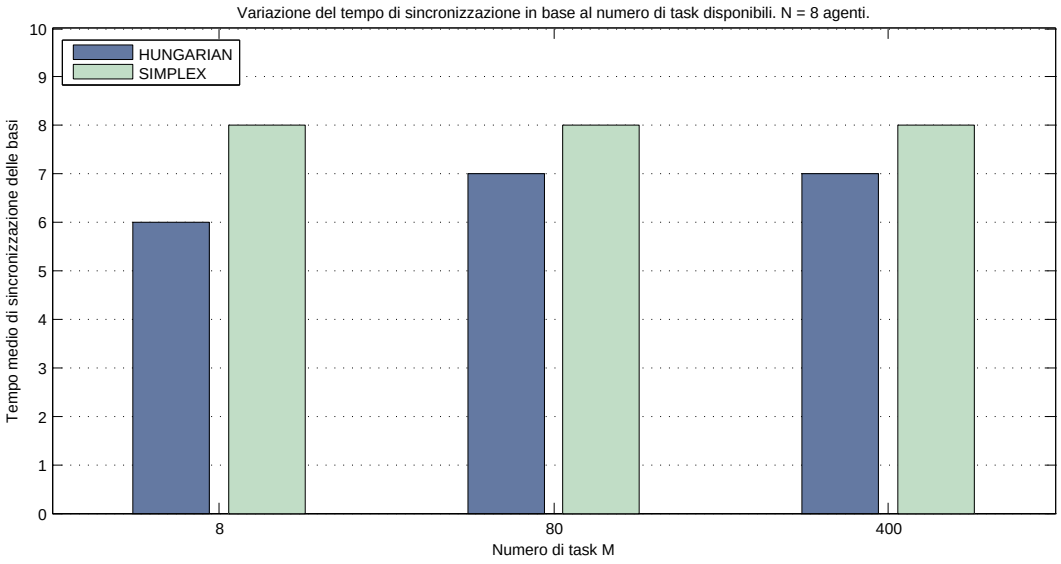


Figura 5.5: Sensibilità al numero di task per agente: implementazione con metodo ungherese ($P_{max} = M$).

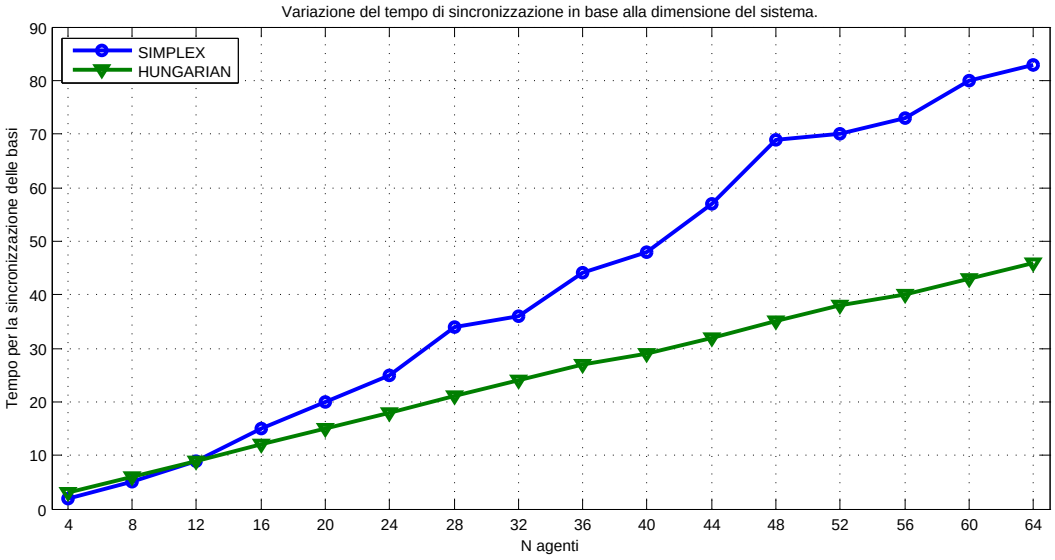


N	P_{max}	P_{TA}
8	M	50%

Figura 5.6: Velocità di convergenza in un sistema di 8 agenti all'aumentare dei task.

Scalabilità

Il parametro fondamentale che determina la dimensione del sistema è, banalmente, il numero di agenti attivi. Le precedenti simulazioni e, più in dettaglio, il grafico in figura 5.7 mostrano come i tempi di convergenza crescano in maniera quasi lineare con l'aumento del numero di nodi.[4]



M	P_{max}	P_{TA}
N	M	25%

Figura 5.7: Velocità di convergenza allo scalare della dimensione (N agenti, N task).

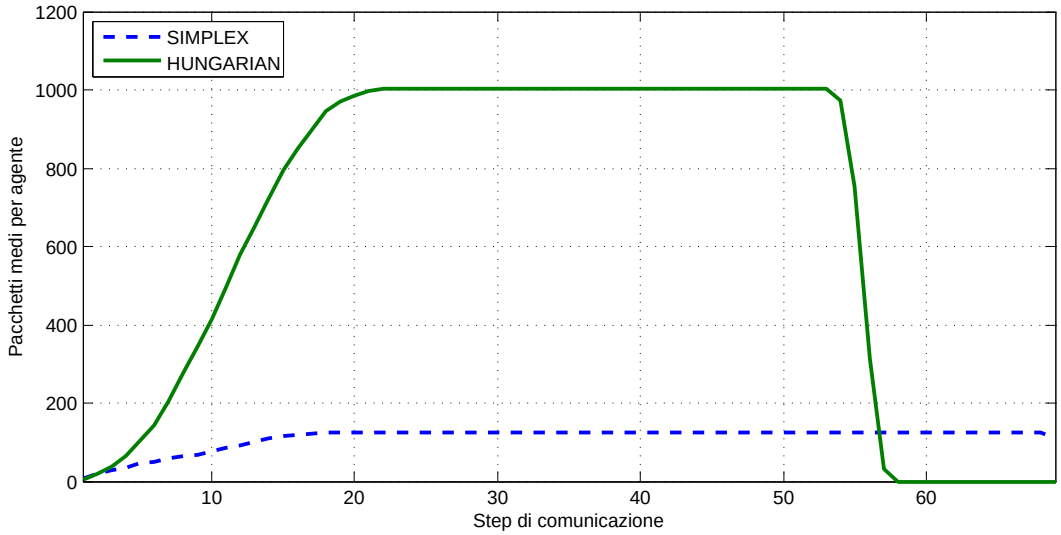
In generale, comunque, si osserva una maggior velocità di convergenza (specie all'aumentare della dimensione) utilizzando il metodo ungherese, rispetto al simplesso lessicografico. Questo aspetto è da attribuire al fatto che il set di vincoli calcolato nel primo caso ha una dimensione maggiore, sovrabbondante, rispetto alla base minima calcolabile con il simplesso. In questo modo, il numero di vincoli (e quindi l'informazione complessiva) che vengono inviati ai vicini ad ogni round di comunicazione è maggiore e porta in minor tempo alla costruzione di una base condivisa.

[4]Si osservi che l'ipotesi di lavoro è $N \leq M$, ma si è già accertato che il numero di task M non incide sulle prestazioni (fig. 5.6), quindi il grafico 5.7 riporta le simulazioni per $N = M$.

5.2 Costo di comunicazione

Un altro aspetto da tenere in considerazione è il carico imposto alla rete di comunicazione sotto forma di pacchetti-vincolo.

In figura 5.8 è riportato l'andamento del traffico di rete (in vincoli spediti per agente) durante l'evoluzione dell'algoritmo. Si osserva immediatamente che per il metodo ungherese si ha un rapido incremento del traffico fino ad una soglia abbastanza elevata che viene mantenuta per un certo periodo, per poi interrompersi man mano che gli agenti considerano stabile la propria base. Al contrario con il simplesso lo scambio di pacchetti viene mantenuto più a lungo (si è visto che occorre un maggior tempo per ottenere il consenso sulla base), ma è di entità considerevolmente più bassa.



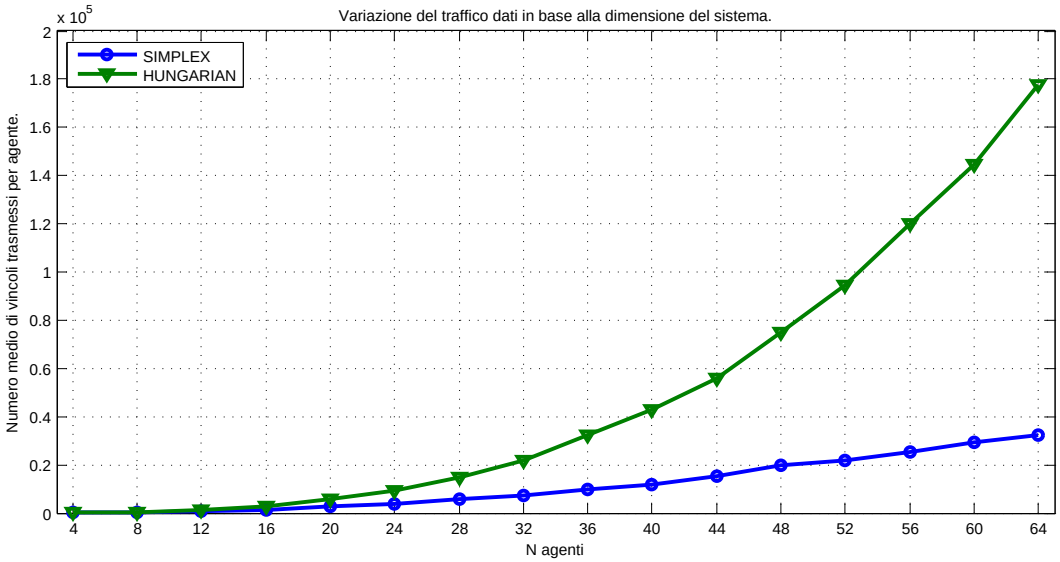
N	M	P_{max}	P_{TA}
32	64	M	50%

Figura 5.8: Carico di rete durante l'esecuzione dell'algoritmo in un sistema di $N = 32$ agenti.

La disparità di traffico tra i due algoritmi è ben evidenziata in figura 5.9, all'aumentare della dimensione del sistema. Come era prevedibile i vincoli scambiati applicando il metodo ungherese sono limitati da una curva asintotica di tipo $O(N^2)$, mentre il simplesso mantiene una caratteristica lineare $O(N)$.

Densità del grafo di comunicazione

Nelle simulazioni seguenti è sempre utilizzato una topologia ad anello (quindi definita da una matrice circolante), questa volta con collegamenti direzionali.



M	P_{max}	P_{TA}
N	M	25%

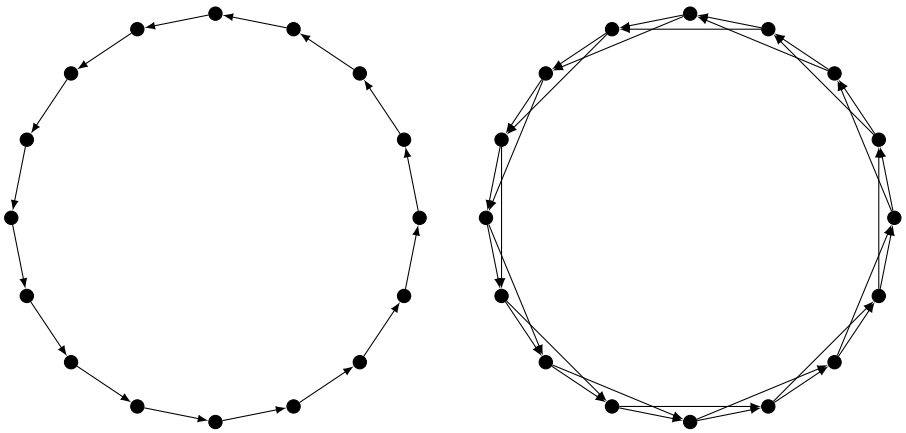
Figura 5.9: Numero medio di vincoli trasmessi per agente all'aumentare della dimensione del sistema (N agenti, N task).

La casistica è espressa in funzione del parametro δ_{out}, ovvero il numero di link in uscita da ogni singolo nodo. In questo scenario il diametro del grafo di comunicazione è determinato direttamente dall'espressione

$$d_{\mathcal{G}} = diam(\mathcal{G}_{com}) = \left\lceil \frac{(N-1)}{\delta_{out}} \right\rceil \tag{5.2}$$

In figura 5.10 sono rappresentati alcuni esempi per un sistema di 16 agenti.

I grafici in figura 5.11 e 5.12 riportano la variazione del tempo di convergenza, rispettivamente per `SIMPLEX` ed `HUNGARIAN`, all'aumentare delle connessioni per link, ed alla conseguente riduzione del diametro del grafo di comunicazione. Si osserva che per entrambe le implementazioni (ungherese e simplesso) si ha un consistente miglioramento delle prestazioni passando da uno e tre-quattro link, mentre il guadagno diventa marginale per δ_{out} maggiori.



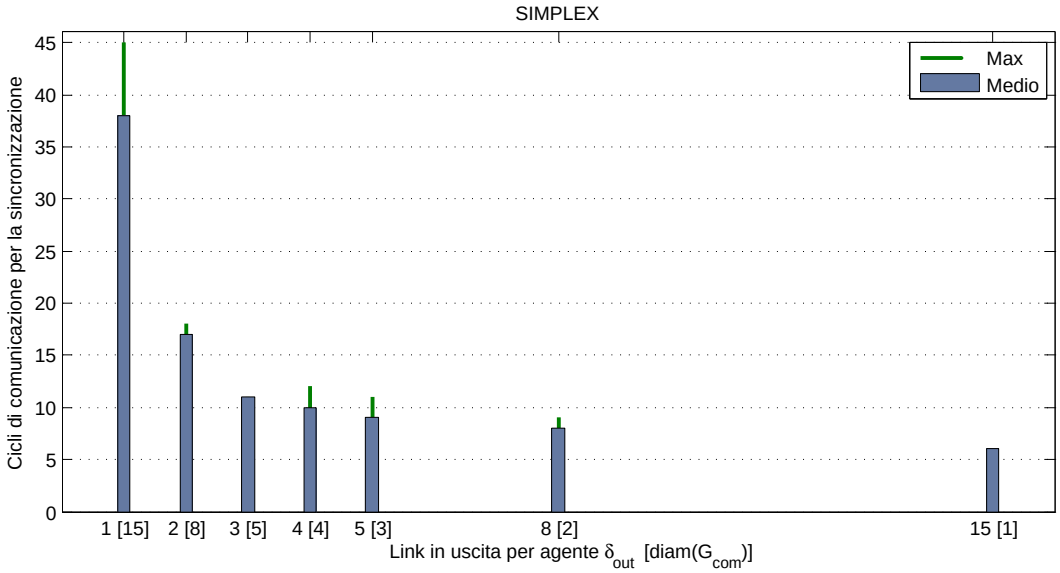
(a) $\delta_{out} = 1,\ d_{\mathcal{G}} = 15$

(b) $\delta_{out} = 2,\ d_{\mathcal{G}} = 7$

(c) $\delta_{out} = 4,\ d_{\mathcal{G}} = 4$

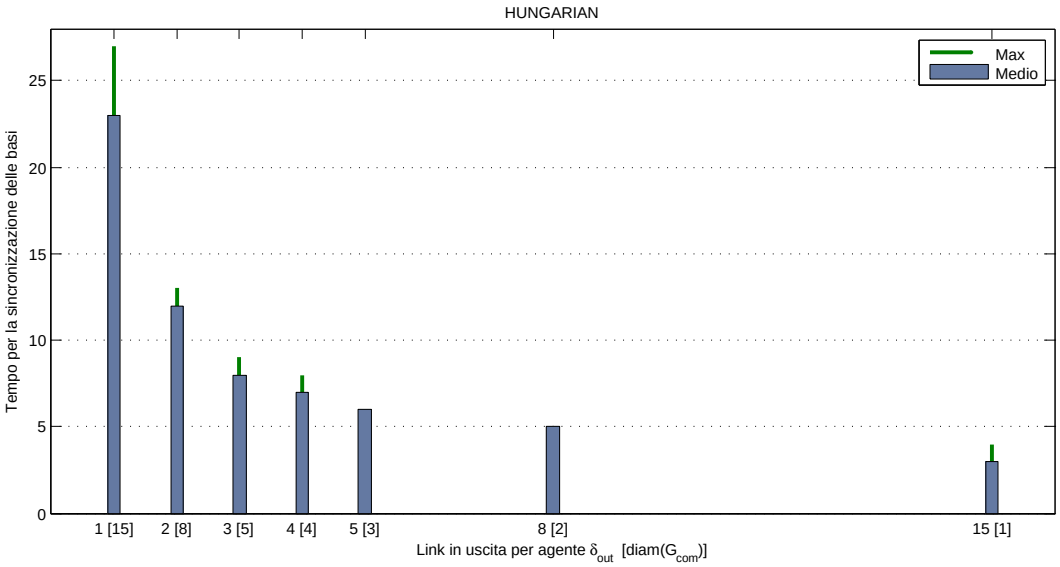
(d) $\delta_{out},\ d_{\mathcal{G}} = 3$

Figura 5.10: Grafo di comunicazione diretto per $N = 16$ e diverse densità di collegamento.



N	M	P_{max}	P_{TA}
16	64	M	50%

Figura 5.11: Influenza della densità di connessioni sul tempo di convergenza: implementazione con simplesso lessicografico.



N	M	P_{max}	P_{TA}
16	64	M	50%

Figura 5.12: Influenza della densità di connessioni sul tempo di convergenza: implementazione con metodo ungherese.

6 Conclusioni

Gli obiettivi prefissati per la presente tesi erano l'individuazione di un algoritmo distribuito per la soluzione al problema del task assignment su rete multi-agente, specialmente in presenza di un elevato numero di task rispetto agli agenti.

Partendo da tecniche risolutive esatte di tipo centralizzato presenti in letteratura si è passati all'adozione di un algoritmo distribuito. L'approccio adottato è stato quello di partizionare l'informazione relativa al problema globale sui singoli agenti, in forma di vincoli semplici, e di utilizzare una strategia di consenso basata sulla comunicazione per la ricostruzione di una rappresentazione minima del problema di partenza, a livello locale in ogni nodo.

Il comportamento complessivo del sistema si è rivelato soddisfacente, risolvendo in modo ottimo l'assegnazione e presentando un tempo di convergenza che, nelle simulazioni numeriche, appare grossomodo lineare con le dimensioni del sistema. Si è verificato, numericamente, che la velocità dell'algoritmo è scarsamente influenzata da scelte modellistiche come la varianza delle priorità nei task o la densità di accoppiamenti compatibili, fissato il numero di task e agenti (da cui, invece, dipende sostanzialmente). Inoltre il numero di task disponibili nel sistema, quindi la congestione dello stesso, non degrada l'efficienza del sistema.

Per la soluzione del problema locale sono stati proposti due diversi algoritmi, simplesso e metodo ungherese. Quest'ultimo ha evidenziato, in fase simulativa, una maggior velocità di convergenza a discapito di un maggior traffico di comunicazione, che lo rende problematico dal punto di vista della scalabilità per sistemi con molti agenti. Emerge in tal senso un possibile criterio di trade-off nella valutazione dell'implementazione più conveniente.

Infine si è osservato come la rete, implementato un semplice sistema di notifica degli eventi, risulti reattiva in scenari di assegnazione dinamica, con processi di nascita e morte dei task e sia affidabile a fronte di eventi di guasto che coinvolgano un insieme ridotto di agenti.

6.1 Possibili approfondimenti

Un interessante spunto di approfondimento è il seguente: verificare la possibilità di modifica dell'algoritmo ungherese, oppure di uno degli algoritmi polinomiali per il task assignment, in modo da ottenere come risultato la soluzione ottima al problema perturbato lessicograficamente, evitando problemi di stabilità numerica analogamente a quanto fatto per il simplesso. In questo modo sarebbe possibile mantenere i pro di un algoritmo compatto e performante come il Kuhn-Munkres, che la ottime proprietà di scalabilità relative al simplesso nel Constraint Consensus.

Inoltre, data la maggior velocità di convergenza associata all'incrementata taglia delle comunicazioni rispetto alla base minima del simplesso, è forse possibile identificare un'euristica per la notifica di vincoli fuori base aggiuntivi da inviare contestualmente alla base minima per ottenere, in parte o in toto, lo stesso effetto.

Infine, un altro aspetto indagabile riguarda la struttura della matrice di compatibilità task-agente, che in molte applicazioni tende ad alla forma diagonale a blocchi (salvo permutazioni di righe e colonne) e suggerisce una possibile clusterizzazione o una gerarchizzazione degli agenti. In questo contesto è forse possibile definire una strategia di Constraint Consensus tra gruppi, anziché tra singoli agenti, con un conseguente incremento delle prestazioni.

Bibliografia

[1] Dimitri P Bertsekas. «A new algorithm for the assignment problem». In: *Mathematical Programming* 21.1 (1981), pp. 152–171.

[2] Dimitri P Bertsekas. «The auction algorithm for assignment and other network flow problems: A tutorial». In: *Interfaces* 20.4 (1990), pp. 133–149.

[3] Robert G Bland. «New finite pivoting rules for the simplex method». In: *Mathematics of Operations Research* 2.2 (1977), pp. 103–107.

[4] Francois Bourgeois e Jean-Claude Lassalle. «An extension of the Munkres algorithm for the assignment problem to rectangular matrices». In: *Communications of the ACM* 14.12 (1971), pp. 802–804.

[5] Rainer E Burkard, Mauro Dell'Amico, Silvano Martello et al. *Assignment Problems, Revised Reprint*. Siam, 2009.

[6] George Bernard Dantzig. *Linear programming and extensions*. Princeton university press, 1965.

[7] Matteo Fischetti. *Lezioni di ricerca operativa*. Libreria Progetto, 1999.

[8] Lester R Ford e Delbert R Fulkerson. «Maximal flow through a network». In: *Canadian journal of Mathematics* 8.3 (1956), pp. 399–404.

[9] David Gale e Lloyd S Shapley. «College admissions and the stability of marriage». In: *American Mathematical Monthly* (1962), pp. 9–15.

[10] William Hunt. *The stable marriage problem*. URL: `www.csee.wvu.edu/~ksmani/courses/fa01/random/lecnotes/lecture5.pdf`.

[11] Colin N Jones, Eric C Kerrigan e Jan M Maciejowski. «Lexicographic perturbation for multiparametric linear programming with applications to control». In: *Automatica* 43.10 (2007), pp. 1808–1816.

[12] Harold W Kuhn. «The Hungarian method for the assignment problem». In: *Naval research logistics quarterly* 2.1-2 (2006), pp. 83–97.

[13] Eugene L Lawler. *Combinatorial optimization: networks and matroids*. DoverPublications. com, 2001.

[14] David F Manlove et al. «Hard variants of stable marriage». In: *Theoretical Computer Science* 276.1 (2002), pp. 261–279.

[15] James Munkres. «Algorithms for the assignment and transportation problems». In: *Journal of the Society for Industrial & Applied Mathematics* 5.1 (1957), pp. 32–38.

[16] David W Pentico. «Assignment problems: A golden anniversary survey». In: *European Journal of Operational Research* 176.2 (2007), pp. 774–793.