Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Allocazione di compiti in un team di robot autonomi mediante l'algoritmo Max-Sum

Candidato:
Elena Zanotto
Matricola: 1041131 - IF

Relatore:
Ch.mo Prof. Enrico Pagello

Correlatore:
Ing. Alessandro Farinelli

Anno Accademico 2013 - 2014

"Siamo fatti anche noi della materia di cui son fatti i sogni;
e nello spazio e nel tempo d'un sogno è racchiusa la nostra breve vita"

William Shakespeare, *La tempesta*
Prospero: atto IV, scena I

A te,
la fonte di tutta la forza che non ha mai smesso di accompagnarmi
in ogni battaglia vinta o persa, ogni giorno di questi cinque lunghi anni.

Abstract

Lo scopo di questa tesi è quello di proporre una soluzione al problema dell'assegnazione di task in un sistema multi robot, nell'ambito di una challenge robotica, utilizzando l'algoritmo *max-sum* e la struttura del *factor graph*.
A tal fine la tesi prevede un primo capitolo di introduzione ai problemi di coordinamento distribuito, al progetto `RoCKIn` ed in particolare alla challenge `RoCKIn@work` in riferimento alla quale è stato ideato questo lavoro. Segue un capitolo che descrive gli strumenti matematici utilizzati per lo sviluppo della soluzione proposta, factor graph e algoritmo max-sum. Il terzo capitolo riassume la procedura seguita per modellare il sistema, ne descrive l'implementazione e fornisce alcuni esempi di applicazione. Seguono il quarto capitolo, che riporta vari casi di applicazione di *max-sum* trattati nella letteratura recente, ed il quinto, che riporta analisi qualitative e possibili estensioni del lavoro svolto.

Indice

Capitolo 1

Introduzione

Lo scopo di questa tesi è quello di fornire una soluzione al problema della ripartizione di task all'interno di un sistema multi robot, utilizzando l'algoritmo max-sum applicato al modello grafico definito dal factor graph.
Si procede quindi a contestualizzare il problema fornendo una breve introduzione a tutti gli argomenti trattati.

1.1 Distributed Constraint Optimization Problems

La soluzione di problemi di ottimizzazione nell'ambito di sistemi multiagente, trae vantaggio dall'esistenza di vincoli, poiché essi possono essere interpretati come elementi che manifestano conoscenza dei particolari del sistema stesso.
La categoria di problemi entro la quale ricade quello affrontato in questa tesi è denominata DCOPs ovvero Distributed Constraint Optimization Problems: in questo contesto un insieme di agenti cerca un accordo, attraverso l'utilizzo di una forma di negoziazione, per decidere quale azione intraprendere al fine di ottenere la soluzione migliore per il sistema [4]. L'aspetto distribuito del problema si concretizza nel fatto che ogni agente cerca di negoziare localmente, cioè solo con quel sottoinsieme di altri agenti che può influenzare il suo comportamento.

Definizione 1.1. Una rete vincolata $\mathcal{N}$ è una tupla $<X,D,C>$ in cui:
- $X = \{x_1, x_2, \cdots, x_n\}$ è un insieme di variabili discrete
- $D = \{D_1, D_2, \cdots, D_n\}$ è l'insieme dei domini delle variabili
- $C = \{C_1, C_2, \cdots, C_m\}$ è un insieme di vincoli. Essi possono essere *hard constraint* o *soft constraint*.

Definizione 1.2. Un *hard constraint* C_i^h è una relazione R_i definita su un sottoinsieme delle variabili S_i. Questo definisce l'ambito del vincolo ed R_i elenca tutti i validi assegnamenti di valori alla variabili.
Un *soft constraint* C_i^s è una funzione F_i definita sull'insieme S_i che mappa ogni assegnamento delle variabili in un valore reale.

Se i vincoli sono tutti di tipo hard, il problema è detto **Constraint Satisfaction Problem** (CSP) e la soluzione cercata è un assegnamento di valori alle variabili che soddisfi tutti i vincoli; se alcuni, invece, sono di tipo soft è detto **Constraint Optimization Problem** (COP) e la soluzione cercata è la migliore ovvero quella che, oltre a soddisfare tutti i vincoli, ottimizza la funzione globale.

Definizione 1.3. La funzione globale $F(\bar{a})$ per una rete $\mathcal{N}$ è l'aggregazione delle funzioni locali $F_i(\bar{a})$ che rappresentano i *soft constraint*. Sia $\bar{a} = (a_1, a_2, \cdots, a_n)$ con $a_j \in D_j$ un assegnamento di valori alle variabili e $\bar{a_i}$ la sua restrizione a S_i, allora

$$F(\bar{a}) = \sum_i F_i(\bar{a}_i)$$

Un COP può essere di massimizzazione o minimizzazione ovvero,rispettivamente, essi cercano l'assegnamento $\bar{a}^*$ tale che:

- $\bar{a}^* = \text{argmax}_{\bar{a}} \sum_i F_i(\bar{a}_i)$ per un problema di massimizzazione
- $\bar{a}^* = \text{argmin}_{\bar{a}} \sum_i F_i(\bar{a}_i)$ per n problema di minimizzazione

Ogni CSP può essere visto come un COP in cui si cerca la soluzione che minimizza il numero di vincoli violati.

Definizione 1.4. Un DCOP può essere rappresentato da una rete $\mathcal{N}$=<X,D,C> con *soft constraints* e un insieme di agenti $A = (A_1, A_2, \cdots, A_k)$, ognuno dei quali può controllare solo un sottoinsieme delle variabili $X_i \subseteq X$, e ciascuna variabile è assegnata ad un solo agente.

In generale gli agenti si considerano interessati ad ottimizzare la funzione globale piuttosto che le loro singole utilità.
La ricerca della soluzione ottima per un DCOP è un problema NP-Hard.

1.1.1 Tecniche per la risoluzione di un DCOP

Gli algoritmi che risolvono i problemi in oggetto sono di due tipi: esatti e approssimati.

Algoritmi esatti

Questi algoritmi trovano sempre la soluzione che ottimizza la funzione globale e possono essere suddivisi in due gruppi:

- **Search-based**: ricercano la soluzione a ritroso a partire dalla migliore assegnazione di valori per ogni funzione locale o si avvalgono di tecniche branch-and-bound. Possono essere suddivisi in due sotto categorie:
  - **sincroni**: se ogni agente attende la ricezione di tutti i messaggi dagli agenti a lui vicini prima di procedere al calcolo del proprio. Questi algoritmi permettono che ogni agente disponga sempre delle informazioni più aggiornate.
  - **asincroni**: se l'agente non attende, ovvero calcola ed invia il messaggio a prescindere dai suoi vicini. Queste tecniche possono essere applicate solo in sistemi in cui agli agenti è permesso prendere decisioni senza consultare i vicini.
- **Dynamic Programming**: lavorano per fasi alla ricerca del valore ottimo di una variabile per volta.

Algoritmi approssimati

Sono preferiti in quei casi in cui i sistemi coinvolgono dispositivi fisici che hanno vincoli molto stretti di memoria e capacità di calcolo (come robot o sensori), ma non garantiscono sulla qualità della soluzione ottenuta.
Due dei sottoinsiemi in cui si dividono questi algoritmi sono:

- **Metodi greedy**: considerano inizialmente un assegnamento random di valori alle variabili e modificano i valori localmente, cercando di massimizzare la funzione obiettivo. La qualità della soluzione dipende dalla presenza di massimi locali, tuttavia, i costi di esecuzioni sono bassi e la scalabilità buona.
- **Approcci Generalised Distributive Law**: la GDL è un framework comune che bypassa il problema degli ottimi locali che si presenta nella tecnica precedente. Si applica su anelli semi commutativi e, a seconda delle specifiche operazioni definite sull'insieme, se ne distinguono molte varianti, due delle quali sono esposte a partire da Sezione 2.1.4.

1.2 RoCKIn Project

RoCKIn, acronimo di **Robot Competitions Kick Innovation in cognitive systems and robotics**, è progetto finanziato dall'Unione Europea (sviluppato dai soggetti in Figura 1.1) nell'ambito del *Seventh Framework Programme* per il triennio 2013-2015, che si propone di promuovere il progresso scientifico nella robotica e nei sistemi intelligenti, attraverso il design e l'implementazione di *competitions* ed il loro *benchmarking* [9]. Contemporaneamente, il progetto persegue l'obiettivo di aumentare la consapevolezza pubblica, in merito allo stato dell'arte nell'ambito della robotica e, al potenziale innovativo delle sue nuove applicazioni, al fine di favorire la competitività della aziende europee (@Work) e migliorare la qualità della vita della popolazione (@Home).

Figura 1.1: Partner del progetto RoCKIn

La scelta di strutturare il progetto intorno alla definizione di competizioni nasce da alcune considerazioni:

- esse sono in grado di attrarre un vasto ed eterogeneo pubblico, dal ricercatore all'appassionato di tecnologia, fino all'azienda in cerca di nuove idee da sviluppare o applicare al proprio settore
- le competizioni danno la possibilità di provare sul campo studi e ricerche spesso sviluppate solo a livello teorico
- esse danno la possibilità di confrontare la preparazione di team diversi, premiare e diffondere specifiche soluzioni particolarmente soddisfacenti

Il secondo punto focale del progetto è quello del *benchmarking*, esso infatti è un aspetto trascurato, o del tutto assente, dalle numerose competizioni già esistenti. Senza la possibilità di raccogliere ed analizzare i dati, viene persa la possibilità di valutare e comparare in modo obiettivo i risultati di approcci diversi che si propongono di risolvere il medesimo problema.
É altresì importante che i dati raccolti siano ottenuti a partire dalle medesime condizioni iniziali ed è per questo che viene posta molta attenzione nel fornire ai team partecipanti lo stesso ambiente di sviluppo.

Le competizioni ideate sono due: una nell'ambito della robotica di servizio dal nome *RoCKIn@Home* ed una per la robotica industriale *RoCKIn@Work*.
Nonostante il nome scelto sia simile a quello delle ben più note *RoboCup@Home* e *RoboCup@Work*, esse si contraddistinguono, oltre che per le specifiche tecniche, anche per il sopracitato benchmarking.

Il progetto è attuato attraverso camp e competizioni, al momento due dei tre camp previsti si sono già svolti mentre le competizioni si terranno nel Novembre 2014 e 2015 entrambe nell'ambito della *European Robotics Week*.

Ho avuto la possibilità di partecipare ad entrambi i camp e brevemente ne riassumo contenuto e finalità:

- **RoCKIn Camp 2013**: si è tenuto a Luglio 2013 ad Eindhoven (Paesi Bassi) durante la *RoboCup 2013*, il suo scopo era quello di introdurre i partecipanti al progetto e supportarne la diffusione. [14]

- **RoCKIn Camp 2014**: si è tenuto nel Gennaio 2014 a Roma (Italia) ed ha accolto un totale di 19 team di cui 11 per la lega *@Home* e 8 per la lega *@Work*. Ho costituito, insieme ad un dottorando ed un borsista, il team IAS-Lab che ha partecipato alla lega *@Work* insieme ad un team di dottorandi dell'Università La Sapienza di Roma, ricevendo un attestato per la miglior prova pratica nella sezione perception. Questo evento aveva lo scopo di consentire ai team la prova sul campo delle proprie competenze, fornendo le specifiche basi teoriche necessarie e l'hardware a coloro che ne erano sprovvisti.

1.2.1 RoCKIn@Home

Uno dei più interessanti problemi per quanto riguarda l'ambito sociale, è quello della gestione delle problematiche legate all'anzianità, la qualità della vita in quel periodo e la possibilità di fornire adeguata assistenza a chi ne necessita.
Il progresso degli ultimi anni in questo campo è stato inferiore rispetto alle aspettative e, la maggior parte delle soluzioni a problemi di questo genere, non sono attualmente forniti dalla robotica [11].

Lo scenario in cui si pone RoCKIn@Home prevede una signora anziana vedova ed in pensione, che vive nel suo appartamento in compagnia di due gatti e ha la necessità di essere aiutata nei movimenti, di prendere regolarmente dei medicinali, seguire una dieta, monitorare il suo stato di salute e accogliere ospiti. Questi compiti, di notevole complessità, non devono essere eseguiti necessariamente da un unico robot e possono usufruire di sensoristica di vario genere disposta nella casa.
L'appartamento tipo è pensato in modo da riflettere la tipica abitazione europea (Figura 1.2) ed è composto da una cucina, una sala da pranzo e salotto, una camera da letto ed un bagno, il tutto collegato da un corridoio.

Le categorie di task che il/i robot deve/devono essere in grado di eseguire sono le seguenti:

- **Servire i pasti**: preparare e servire i pasti e le bevande, preparare la tavola e sparecchiare

- **Assistenza alla mobilità**: aiutare la persona a sedersi ed alzarsi, guidarla all'interno della casa

- **Supporto alla terapia farmacologica**: assicurarsi che la persona assuma le medicine prescritte nella modalità corretta

- **Aiuto nelle relazioni sociali**: aiutare la persona a tenere i contatti con amici e familiari fornendo supporto nell'utilizzo del telefono, email e simili
- **Gestione accoglienza ospiti**: accogliere gli ospiti e guidarli all'interno dell'appartamento, interagire con loro attraverso la comunicazione
- **Intrattenimento**: fornire assistenza nell'utilizzo dei media, giocare a carte
- **Supporto al comfort personale**: controllare la temperatura della stanza e l'intensità della luce
- **Cura degli animali domestici**: fornire loro cibo e acqua
- **Gestione di emergenze**: ricerca oggetti perduti, sostituzione di quelli rotti, assistenza in caso di cadute o incidenti
- **Gestione inventario**: mantenere un inventario degli oggetti necessari in casa, provvedere ad ordinarli quando necessario
- **Supporto nei lavori di pulizia**: provvedere alla pulizia e a mantenere l'ordine

Figura 1.2: Due possibili varianti dell'arena di RoCKIn@Home

1.2.2 RoCKIn@Work

La competizione in oggetto è stata sviluppata a partire dall'analisi del corrispettivo in RoboCup, ma da essa si differenzia per le motivazioni che hanno determinato la scelta dei task da svolgere: mentre in RoboCup i task sono stati scelti in modo da essere il più generali possibile, in RoCKIn, essi nascono dall'analisi delle esigenze del settore industriale [10].

Le industrie Europee si trovano ad affrontare un mercato in continua e veloce evoluzione: mentre le aziende multinazionali, o di dimensioni molto elevate, possono trasferire o delegare le fasi logistica/manifatturiera a terzi, le piccole e medie imprese non hanno la medesima opportunità ed è con riferimento ad esse che è

stata sviluppata la challenge.
Nel loro caso, un sistema completamente automatizzato non è una soluzione proponibile, non solo a causa del notevole investimento economico richiesto, ma anche poiché la flessibilità richiesta nella produzione (in termini di varietà e complessità delle operazioni da eseguire) non ne permetterebbe l'adozione.
Il settore della robotica industriale in Europa è convinto che i recenti sviluppi nell'ambito dei manipolatori mobili, manipolatori leggeri e mano robotiche sempre più abili, possano aprire la strada a nuove applicazioni e nuovi mercati. Al fine di contribuire al convincimento delle industrie manifatturiere ad investire in sistemi robotici è necessario fornire loro dimostrazioni delle effettive capacità dei robot e delle loro possibili applicazioni. Un modo utile per farlo, è quello di utilizzare la challenge RoCKIn@Work.

Lo scenario pensato per la gara prevede una fabbrica specializzata nella produzione di parti meccaniche di piccole e medie dimensioni che si trova a gestire un volume giornaliero di produzione sempre diverso in funzione degli ordini, che deve inoltre gestire il loro assemblaggio, la spedizione e le restituzioni.
L'ambiente prevede diverse workstation in cui uomini, o eventualmente robot, dispongono le parti necessarie all'assemblaggio secondo gli ordini di uno scheduler centrale. Ogni prodotto richiede un differente insieme di parti, tutte queste sono riposte in dei contenitori appositi.

Figura 1.3: Visione dall'alto dello scenario RoCKIn@Work

I task che i robot devono risolvere sono suddivisi in quattro categorie:

- **Accettazione ordine e assemblaggio lotto**: i robot devono provvedere alla raccolta degli oggetti richiesti per l'assemblaggio dal magazzino

- **Gestione restituzioni**: un uomo si occupa di aprire di pacchi restituiti, i robot quindi devono estrarre le parti, determinarne il tipo e, secondo quest'ultimo, eseguire una specifica azione

- **Personalizzazione ordini**: nell'ottica di dover assemblare scatole regalo, gli oggetti da posizionarvi andranno riposti in modo appropriato al fine di poter confezionare il regalo stesso
- ***Processing* consegne alla fabbrica**: solitamente le consegne sono effettuate attraverso lotti che riuniscono materiale di diversa tipologia, i robot devono provvedere alla suddivisione gli oggetti per tipologia in apposite scatole.

Lo scenario sopra descritto è schematizzato in Figura 1.3: esso prevede delle workstation (in basso nell'immagine), i riquadri rossi indicano scatole di componenti e i blu le scatole vuote che andranno riempite seguendo le specifiche richieste dell'ordine. Le scatole rosse e blu più grandi (in alto), rappresentano rispettivamente il posto in cui rifornirsi dei componenti ed il luogo in cui depositare l'oggetto assemblato; tra le due si trova un nastro trasportatore nel quale riporre i prodotti pronti per la spedizione.

L'idea di scenario si concretizza nella challenge in un ambiente in scala, ovvero un'arena suddivisa in due zone, una per l'assemblaggio ed una di produzione, separate da uno o più scaffali (Figura 1.4).

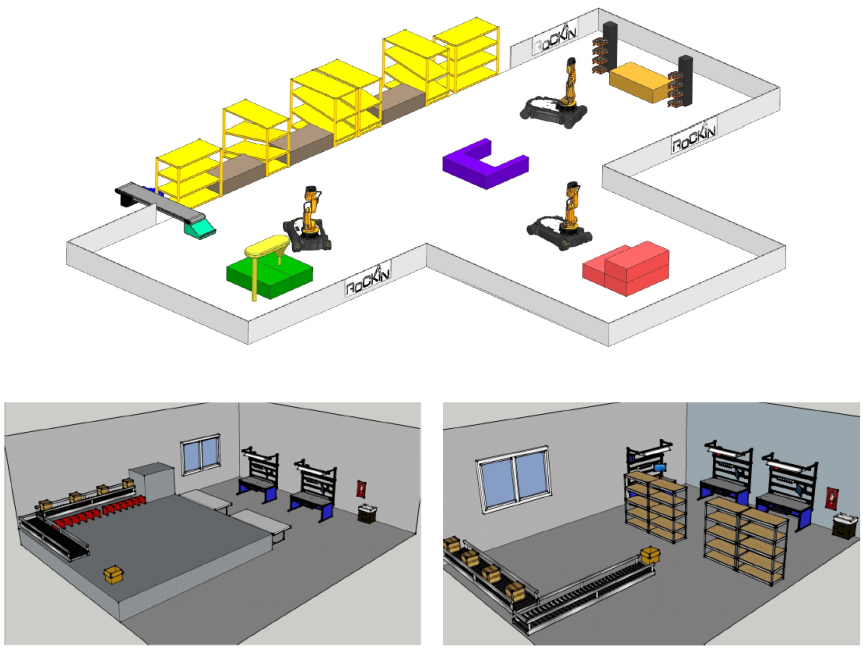


Figura 1.4: Alcune possibili varianti dell'arena di RoCKIn@Work

La challenge è stata creata in modo che i task potessero essere eseguiti da manipolatori mobili, ovvero robot dotati di una piattaforma mobile su ruote con almeno un braccio ed un gripper, o una mano, di piccole dimensioni quali ad esempio il KUKA youBot o il Little Helper della Aalborg University.

KUKA® youBot

Lo youBot di KUKA, azienda tedesca di fama internazionale impegnata nella ricerca di soluzioni avanzate nei settori industriale e dell'automazione, è un robot manipolatore concepito principalmente a scopo didattico e per la ricerca.
Esso è fornito di interfacce opensource che permettono l'accesso al sistema alla totalità dei livelli. Il software di riferimento sono le Application Programming Interface *KUKA® youBot API*, disponibili per i framework più diffusi (ROS e OROCOS) e l'integrazione con il simulatore Gazebo e VREP.

Il robot, giunto attualmente alla seconda versione, è composto da due parti principali controllabili anche in modo autonomo: la base ed il braccio (o i bracci) dotate di sensoristica di vario tipo.

- **Piattaforma omnidirezionale**: la base del robot è costituita da una piattaforma mobile montata su quattro ruote omnidirezionali, telaio, motore, alimentazione e un onboard-PC

Figura 1.5: Piattaforma omnidirezionale di KUKA youBot.

- **Braccio**: dotato di cinque gradi di libertà, su di esso è montato un gripper a due dita. Può essere controllato sia dal pc interno alla base che da un pc esterno attraverso un collegamento con cavo Ethernet.

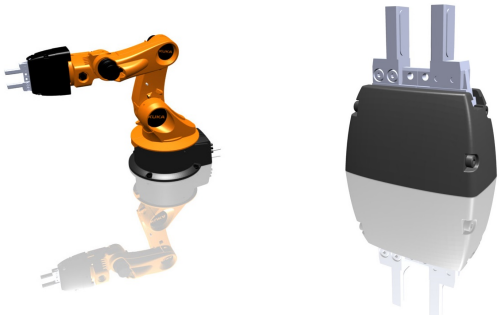


Figura 1.6: Braccio e gripper di KUKA youBot

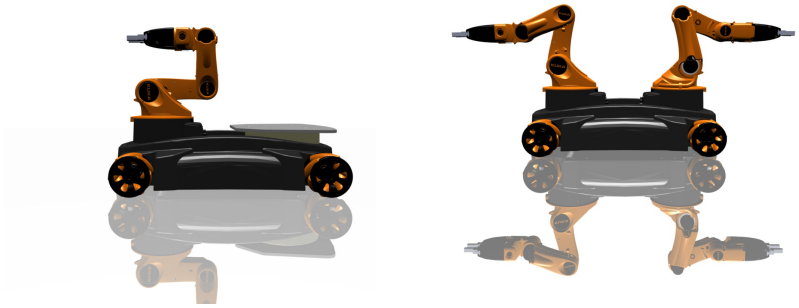


Figura 1.7: Configurazione di youBot con uno e due bracci

Little Helper - Aalborg University

Little Helper è un robot realizzato presso la Aalborg University (DK), all'interno di un progetto di ricerca del Dipartimento di Ingegneria Meccanica e Manifatturiera, a partire dal 2007. Lo scopo del progetto è quello di realizzare un robot mobile, impiegabile in ambito industriale, in grado di gestire interazioni sia con l'uomo che con altre macchine [24] [25].
La struttura del robot è giunta ora alla terza generazione (Figura 1.8), essa è pensata in modo da essere estremamente modulare, di facile utilizzo per mantenere il più basse possibili le barriere d'ingresso.

Figura 1.8: Little Helper di 3^a generazione

Il robot è dotato di un onboard-pc montato su piattaforma mobile (attualmente non omnidirezionale), un braccio (un KUKA® LBR) e diversi sensori (laser, ultrasuoni, telecamera). I componenti sono disaccoppiati in modo tale da consen-

tire che quelli non necessari allo svolgimento dello specifico task siano disattivati durante l'esecuzione (ad esempio il braccio fermo durante lo spostamento, o la piattaforma ferma durante la manipolazione).
I task che il Little Helper può eseguire sono suddivisi in due categorie:

- **tech-based**: applicazioni in cui il robot si muove utilizzando una precedente calibrazione
- **vision-based**: applicazioni in cui gli spostamenti del manipolatore avvengono con riferimento al sistema di visione

Nonostante le specifiche della challenge non prevedano strettamente l'utilizzo esclusivo di uno di questi due robot, la totalità dei team in gara durante RoboCup@work 2013 e il Camp RoCKIn@Work 2014 ha adottato lo youBot di KUKA®.

Capitolo 2

Strumenti matematici

Il *factor graph*[1] è uno strumento grafico utile quando si ha a che fare con funzioni di molte variabili: in questo caso è vantaggioso scomporre la funzione principale nel prodotto di sotto funzioni, il cui valore dipende da un sottoinsieme delle variabili di partenza.
L'algoritmo *sum-product* e la sua variante di interesse per questo lavoro, *max-sum*, calcolano il valore che massimizza la funzione globale sfruttando la struttura del factor graph. Termine e concetto di f.g. furono originariamente introdotti alla fine degli anni '90 da Brendan Frey in riferimento a problemi statistici in [34].

2.1 Factor graph

Esistono vari modi di definire e rappresentare la struttura di un factor graph, di seguito ne saranno esaminate due: la prima definita sulla base di [32] e la seconda di [35].
Prima di procedere nella trattazione dei f.g. vengono riepilogati i concetti base della teoria dei grafi.

Definizione 2.1. Un grafo G è individuato da una coppia di insiemi G=(V,E) tali che $E \subseteq V \times V$. V rappresenta l'insieme dei vertici (o nodi) del grafo mentre E è l'insieme dei lati.

Definizione 2.2. Dato un lato $e \in E$ e due vertici v_1, v_2 tali che $e = (v_1, v_2)$, si dice che v_1 e v_2 sono tra loro adiacenti. L'insieme dei vertici adiacenti al nodo v è indicato come $A(v)$.

Definizione 2.3. Il grado di un nodo è pari al numero di vertici ad esso adiacenti.

Definizione 2.4. Dato un lato $e \in E$ e due vertici v_1, v_2 tali che $e = (v_1, v_2)$, si dice che v_1 e v_2 sono incidenti in e, analogamente e è detto incidente con v_1 e v_2. L'insieme dei vertici incidenti in un lato e è indicato come $I(e)$, analogamente l'insieme dei lati incidenti in un vertice v è $I(v)$. Inoltre $v \in I(e) \Leftrightarrow e \in I(v)$.

Definizione 2.5. Un grafo bipartito è un grafo il cui insieme di vertici V può essere partizionato in due sottoinsiemi, V_X e V_Y. tali che $E \subseteq V_X \times V_Y$.

[1] D'ora in avanti indicato anche con le abbreviazioni f.g. o FG

2.1.1 Funzioni, fattorizzazioni e marginali

Siano $x_1, x_2, ..., x_n$ un insieme di variabili tali che per ogni x_i sia definito un dominio A_i. Sia $g(x_1, x_2, ..., x_n)$ una funzione a valori reali definita sul dominio $S = A_1 \times A_2 \times ... \times A_n$ con codominio $\mathbb{R}$.

Definizione 2.6. Una *configurazione* è un particolare assegnamento di valori a tutte le variabili. Il *configuration space* Ω è l'insieme di tutte le configurazioni, ovvero il dominio della funzione globale f.

Si supponga che $g(x_1, x_2, ..., x_n)$ sia fattorizzabile nel prodotto di un insieme di funzioni, ognuna definita su un sottoinsieme delle variabili $x_1, x_2, ..., x_n$. Ovvero, dati J, insieme di indici discreti, e $X_j \subset \{x_1, x_2, ..., x_n\}$ e definita $f_j(X_j)$ come la funzione che accetta in ingresso gli elementi di X_j, allora si può esprimere:

$$g(x_1, x_2, ..., x_n) = \prod_{j \in J} f_j(X_j) \tag{2.1}$$

Esiste sempre una fattorizzazione banale ad unico fattore, la stessa f e con $S = A_1 \times A_2 \times ... \times A_n$.

Assumendo che in S la somma sia ben definita, allora ad ogni funzione $g(x_1, \cdots, x_n)$ sono associate n funzioni marginali $g_i(x_i)$ tali che per ogni a, $g_i(a)$ è ottenuto sommando i valori di $g(x_1, x_2, ..., x_n)$ su tutte le configurazioni di variabili tali che $g_i = a$, ovvero:

$$g_i(a) = \sum_{x_1 \in A_1} \sum_{x_2 \in A_2} \cdots \sum_{x_{i-1} \in A_{i-1}} \sum_{x_{i+1} \in A_{i+1}} \cdots \sum_{x_n \in A_n} (x_1, x_2, ...x_{i-1}, a, x_{i+1}, ..., x_n) \tag{2.2}$$

si introduce quindi la seguente notazione per gli indici della sommatoria:

$$g_i(x_i) = \sum_{\sim\{x_i\}} (x_1, ..., x_n) \tag{2.3}$$

Definizione 2.7. Una fattorizzazione $\prod_{k=1}^{K} f_k(S_k)$ di una funzione $f(x_1, x_2, ..., x_n)$ contiene un ciclo di lunghezza $L \geq 2$ quando esiste un insieme di L coppie distinte $\{X_{i_1}, X_{i_2}\}, \{X_{i_2}, X_{i_3}\}, \ldots, \{X_{i_L}, X_{i_1}\}$, ed L distinti sottoinsiemi di variabili $S_{k_1}, \ldots, S_{k_L}$ tali che $\{X_{i_l}, X_{i_{(l+1)}}\} \subseteq S_{k_l}$ per ogni $l \in \{1, \ldots, L\}$. Una fattorizzazione di una funzione è aciclica quando non contiene cicli di lunghezza $L \geqslant 2$.

Definizione 2.8. Una fattorizzazione $\prod_{k=1}^{K} f_k(S_k)$ di una funzione $f(x_1, x_2, ..., x_n)$ è detta disconnessa quando è possibile raggruppare i fattori $\prod_{k=1}^{K} f_k(S_k) = f_A(S_A) f_B(S_B)$ in modo che $S_A \cap S_B = \emptyset$. Se non è possibile tale raggruppamento, allora la fattorizzazione è connessa.

Definizione 2.9. Data una fattorizzazione, ad ogni fattore e ad ogni variabile è associata una *neighborhood*. Per un fattore f_k, essa è indicata con $\mathcal{N}(f_k)$ ed è l'insieme delle variabili che appaiono in f_k; per una variabile x_n è indicata con $\mathcal{N}(x_n)$ ed è l'insieme delle funzioni che hanno come variabile x_n.

$$x_n \in \mathcal{N}(f_k) \Leftrightarrow f_k \in \mathcal{N}(x_n)$$

2.1.2 Normal o Kschischang factor graphs

Definizione 2.10. Un *factor graph* è un grafo bipartito che rappresenta graficamente la struttura della fattorizzazione operata in 2.1. Per ogni variabile x_i è presente un vertice *nodo variabile* e per ogni f_j un vertice di tipo *factor node*, esiste un ramo che connette x_i a f_j se e solo se x_i è un argomento di f_j.

Esempio 2.1. Si consideri la seguente funzione:

$$\begin{aligned} f(x_1, x_2, x_3, x_4, x_5, x_6, x_7) =& f_1(x_1, x_2, x_3) f_2(x_1, x_4) f_3(x_1, x_6, x_7) \\ & \cdot f_4(x_4, x_5) f_5(x_4) f_6(x_2) \end{aligned} \tag{2.4}$$

la sua rappresentazione con FG risulta connessa ed aciclica ed è mostrata in figura.

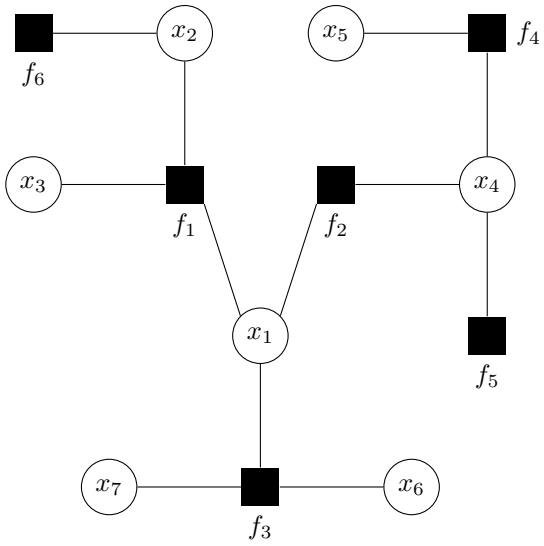


Figura 2.1: Rappresentazione della scomposizione 2.4 su factor graph

2.1.3 Calcolo dei marginali

Per procedere al calcolo dei marginali, il factor graph è particolarmente utile; il procedimento per il calcolo di $g_i(x_i)$ consiste di due step:

- riscrivere la fattorizzazione esplicitandola in funzione di x_i
- calcolare $g_i(x_i)$ applicando la tecnica ricorsiva descritta in Sezione 2.1.3.

Primo step - Riscrivere la funzione

Si consideri il fattore f_k e la variabile $x_n \in \mathcal{N}(f_k)$, alla coppia (x_n, f_k) si associa l'insieme di variabili $S_{f_k}^{(x_n)}$ definito ricorsivamente secondo il seguente algoritmo:

Algoritmo 1: Partizione variabili per determinare $S_{f_k}^{(x_n)}$

input : x_n e f_k
initialization: $S_{f_k}^{(x_n)} = \varnothing$;
for $x_m \in \mathcal{N}(f_k) \setminus \{x_n\}$ **do**
  add x_m to $S_{f_k}^{(x_n)}$;
  for $f_l \in \mathcal{N}(x_m) \setminus \{f_k\}$ **do**
    determine $S_{f_l}^{(x_m)}$ and add to the set $S_{f_k}^{(x_n)}$;
  end
end
output: $S_{f_l}^{(x_m)}$

La procedura descritta tramite l'algoritmo, è molto più intuitiva se eseguita sul factor graph. Si consideri la variabile x_n, essa sarà adiacente ad $|\mathcal{N}(x_n)|$ nodi funzione. Se si rimuove dal grafo di partenza la variabile x_n si ottengono $|\mathcal{N}(x_n)|$ factor graph. Sia $f_n \in \mathcal{N}(x_n)$, allora uno dei grafi ottenuti avrà f_k come nodo foglia, l'insieme delle variabili che compongono questo sotto grafo corrispondono esattamente a $S_{f_k}^{(x_n)}$.

Secondo step - Raggruppare i fattori

Quando la fattorizzazione in esame è connessa ed aciclica, è possibile esprimere la funzione nel modo seguente:

$$f(x_1, x_2, ..., x_n) = \prod_{f_k \in \mathcal{N}(x_n)} h_{f_k}^{(x_n)}(x_n, S_{f_k}^{(x_n)}) \tag{2.5}$$

dove $h_{f_k}^{(x_n)}(x_n, S_{f_k}^{(x_n)})$ sono definite ricorsivamente così:

$$h_{f_k}^{(x_n)}\left(x_n, S_{f_k}^{(x_n)}\right) = f_k\left(\{x_m\}_{x_m \in \mathcal{N}(f_k)}\right) \prod_{x_m \in \mathcal{N}(f_k) \setminus \{x_n\}} \left\{ \prod_{f_l \in \mathcal{N}(x_m) \setminus \{f_k\}} h_{f_l}^{(x_m)}\left(x_m, s_{f_l}^{(x_m)}\right) \right\}$$

dove $\{x_m\}_{x_m \in \mathcal{N}(f_k)}$ rappresenta l'insieme di variabili che compaiono in f_k.
La ricorsione termina quando è vuoto almeno uno tra gli insiemi $\mathcal{N}(x_m) \setminus \{f_k\}$ e $\mathcal{N}(f_k) \setminus \{x_n\}$. Come nello step precedente, anche in questo caso la rappresentazione con factor graph risulta particolarmente utile.
Si consideri la variabile x_n, essa sarà adiacente ad $|\mathcal{N}(x_n)|$ nodi funzione, ognuno dei quali individua un sottografo. Sia f_k uno dei nodi funzioni adiacenti a x_n, allora il sottografo individuato da f_k rappresenta la funzione $h_{f_k}^{(x_n)}(x_n, S_{f_k}^{(x_n)})$.

Esempio 2.2. Si consideri il factor graph di 2.1. Lo step di partizionamento delle variabili a partire dal nodo x_1 ottiene come risultato:

$$S_{f_1}^{(x_1)} = \{x_2, x_3\}$$
$$S_{f_2}^{(x_1)} = \{x_4, x_5\}$$
$$S_{f_3}^{(x_1)} = \{x_6, x_7\}$$

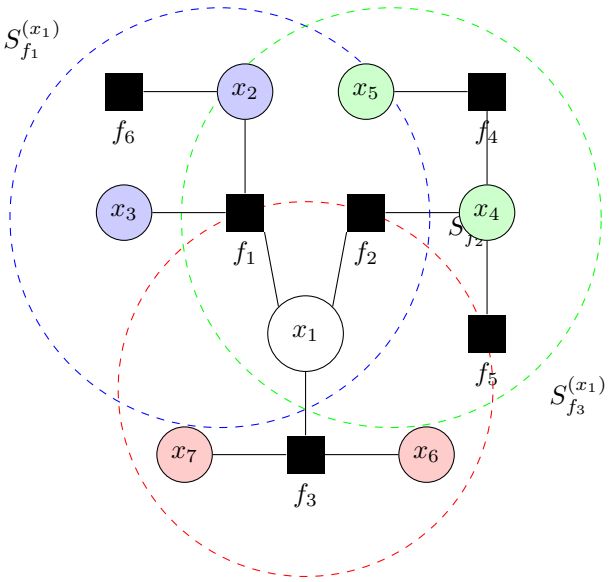


Figura 2.2: Partizionamento variabili di 2.4

Esempio 2.3. Si consideri ancora la funzione definita in 2.1, che può essere espressa come:

$$f(x_1, x_2, ..., x_n) = h_{f_1}^{(x_1)}(x_1, S_{f_1}^{(x_1)}) \cdot h_{f_2}^{(x_1)}(x_1, S_{f_2}^{(x_1)}) \cdot h_{f_3}^{(x_1)}(x_1, S_{f_3}^{(x_1)})$$

dove

$$h_{f_1}^{(x_1)}(x_1, S_{f_1}^{(x_1)}) = f_1(x_1, x_2, x_3) \underbrace{h_{f_6}^{(x_2)}(x_2, S_{f_6}^{(x_2)})}_{=f_6(x_2)}$$

$$h_{f_2}^{(x_1)}(x_1, S_{f_2}^{(x_1)}) = f_2(x_1, x_4) \underbrace{h_{f_4}^{(x_4)}(x_4, S_{f_4}^{(x_4)})}_{=f_4(x_4,x_5)} \underbrace{h_{f_5}^{(x_4)}(x_4, S_{f_5}^{(x_4)})}_{=f_5(x_4)}$$

$$h_{f_3}^{(x_1)}(x_1, S_{f_3}^{(x_1)}) = f_3(x_1, x_6, x_7)$$

Le tre sotto funzioni sono evidenziate con colori diversi in figura:

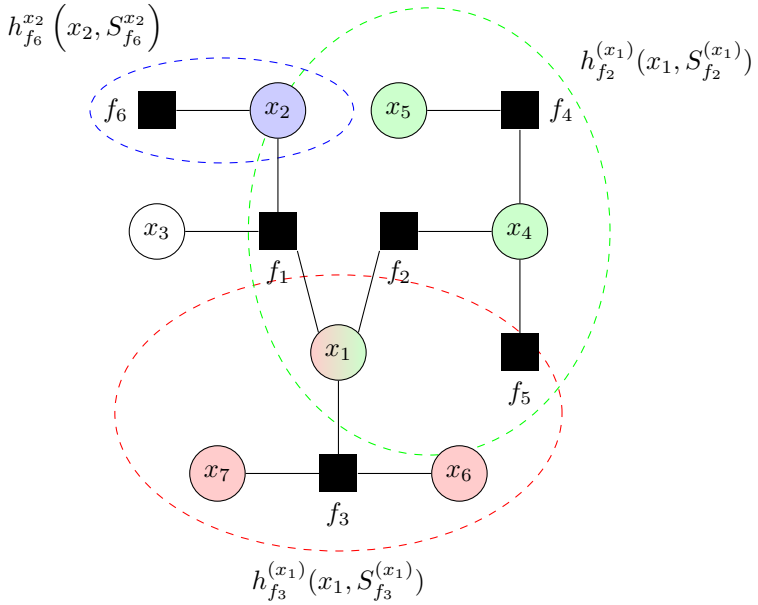


Figura 2.3: Sottofunzioni per il calcolo di $g_{x_1}(x_1)$.

Calcolo dei marginali

Utilizzando la scomposizione creata nello step 2, è possibile riscrivere la formula per estrarre i marginali di una funzione nel seguente modo:

$$\begin{aligned} g_{x_n}(x_n) &= \sum_{\sim\{x_n\}} f(x_1, x_2, ..., x_n) \\ &= \sum_{\sim\{x_n\}} \left\{ \prod_{f_k \in \mathcal{N}(x_n)} h_{f_k}^{(x_n)}(x_n, S_{f_k}^{(x_n)}) \right\} \\ &= \prod_{f_k \in \mathcal{N}(x_n)} \left\{ \sum_{s_{f_k}^{(x_n)}} h_{f_k}^{(x_n)}(x_n, S_{f_k}^{(x_n)}) \right\} \\ &= \prod_{f_k \in \mathcal{N}(x_n)} \left\{ \sum_{\sim\{x_n\}} h_{f_k}^{(x_n)}(x_n, S_{f_k}^{(x_n)}) \right\} \end{aligned} \tag{2.6}$$

E' ora possibile introdurre due funzioni che semplificheranno la trattazione seguente. Per ogni $f_k \in \mathcal{N}(x_n)$ si definiscono $\mu_{x_n \to f_k} : X_n \to \mathbb{R}$ e $\mu_{f_k \to x_n} : X_n \to \mathbb{R}$ nel modo seguente:

$$\begin{aligned} \mu_{f_k \to x_n}(x_n) &= \sum_{\backsim\{x_n\}} h_{f_k}^{(x_n)}(x_n, s_{f_k}^{(x_n)}) \\ &= \sum_{\backsim\{x_n\}} f_k\left(\{x_m\}_{x_m \in \mathcal{N}(f_k)}\right) \prod_{x_m \in \mathcal{N}(f_k)\backslash\{x_n\}} \mu_{x_m \to f_k}(x_m) \end{aligned} \tag{2.7}$$

$$\begin{aligned} \mu_{x_n \to f_k}(x_n) &= \prod_{f_l \in \mathcal{N}(x_n)\backslash\{f_k\}} \left\{ \sum_{\backsim\{x_n\}} h_{f_l}^{(x_n)}(x_n, s_{f_l}^{(x_n)}) \right\} \\ &= \prod_{f_l \in \mathcal{N}(x_n)\backslash\{f_k\}} \mu_{f_l \to x_n}(x_n) \end{aligned} \tag{2.8}$$

Grazie alle funzioni introdotte, si può esprimere il marginale di una funzione nel modo seguente:

$$\begin{aligned} g_n(x_n) &= \prod_{f_k \in \mathcal{N}(x_n)} \mu_{f_k \to x_n}(x_n) \\ &= \mu_{f_l \to x_n}(x_n) \mu_{x_n \to f_l}(x_n) \end{aligned} \tag{2.9}$$

con $f_l \in \mathcal{N}(x_n)$.
Nel contesto del factor graph le funzioni $\mu_{f_k \to x_n}(x_n)$ e $\mu_{x_n \to f_k}(x_n)$ possono essere interpretate come messaggi.
Un messaggio uscente da x_n verso f_k è calcolato sulla base dei messaggi entranti $\mu_{f_l \to x_n}(x_n), f_l \in \mathcal{N}(x_n)\backslash\{f_k\}$ secondo 2.8. In modo analogo un messaggio uscente da f_k verso x_n è calcolato sulla base della funzione f_k e dei messaggi entranti $\mu_{x_m \to f_k}(x_m), x_m \in \mathcal{N}(f_k)\backslash\{x_n\}$ secondo. 2.7.

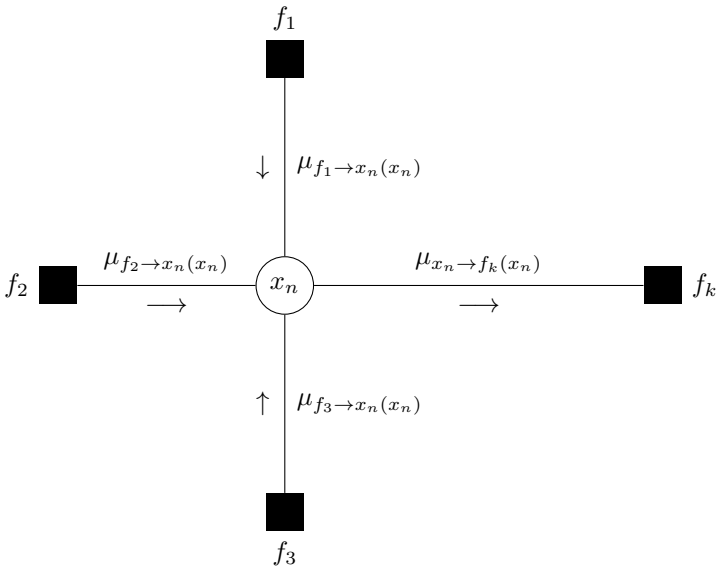


Figura 2.4: Rappresentazione di $\mu_{x_n \to f_k}(x_n)$

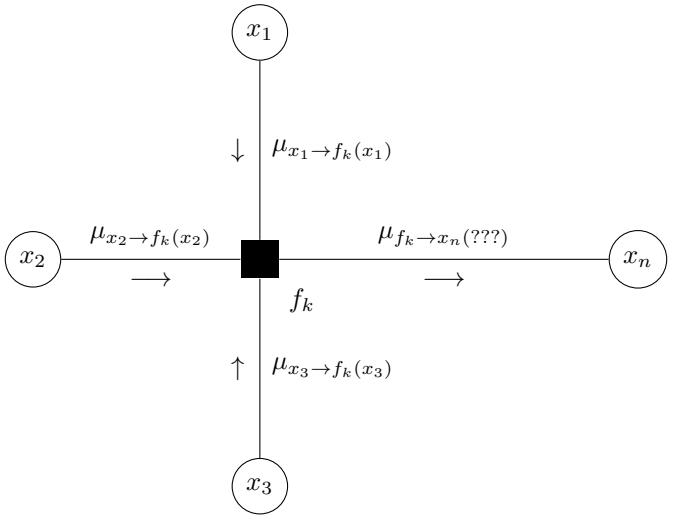


Figura 2.5: Rappresentazione di $\mu_{f_k \to x_n}(x_n)$

2.1.4 Algoritmo sum-product

Il calcolo dei marginali, effettuato attraverso le funzioni precedentemente descritte, prende il nome di *Algoritmo sum-product* abbreviato in SPA. L'algoritmo si articola in tre fasi:

1. **Inizializzazione**
   - Ogni nodo foglia funzione f_k con $\mathcal{N}(f_k) = x_m$, trasmette il messaggio $\mu_{f_k \to x_m}(x_m) = f_k(x_m)$
   - Ogni nodo foglia di tipo variabile x_n con $\mathcal{N}(x_n) = f_l$, trasmette il messaggio $\mu_{x_n \to f_l}(x_m) = 1$
2. **Regole per il calcolo dei messaggi**
   Da ripetere finchè tutti i messaggi $\mu_{f_k \to x_n}(x_n)$ e $\mu_{x_n \to f_k}(x_n)$ non sono stati calcolati. Ogni messaggio è calcolato un'unica volta.
   - Ogni nodo funzione f_k di grado D (l'insieme dei nodi variabile connessi a f_k è denotato con $\mathcal{N}(f_k)$), dopo aver ricevuto in ingresso $D-1$ messaggi distinti provenienti dai nodi variabili $x_i \in \mathcal{N}(f_k), x_i \in \mathcal{N}(f_k)$, trasmette in uscita il messaggio $\mu_{f_k \to x_m}(x_m)$ al nodo rimanente x_m secondo 2.7
   - Ogni nodo variabile x_n di grado D (l'insieme dei nodi funzione connessi a x_n è denotato con $\mathcal{N}(x_n)$), dopo aver ricevuto in ingresso $D-1$ messaggi distinti provenienti dai nodi funzione $f_i \in \mathcal{N}(x_n), f_i \in \mathcal{N}(x_n)$, trasmette in uscita il messaggio $\mu_{x_n \to f_l}(x_n)$ al nodo rimanente f_l secondo 2.8
3. **Fase finale**
   - Per procedere al calcolo del marginale di x_n, considerare una $f_k \in \mathcal{N}(x_n)$ e moltiplicarla per i messaggi trasmessi lungo l'arco (x_n, f_k) secondo 2.9.

Con la procedura appena descritta non è più necessario determinare in modo esplicito gli insiemi $\mathcal{S}_{f_l}^{(x_m)}$ e le funzioni $h_{f_l}^{(x_m)}(x_m, \mathcal{S}_{f_l}^{(x_m)})$.
L'esempio 2.5 propone un metodo alternativo per il calcolo delle funzioni marginali.

Esempio 2.4. In questo esempio si applica la procedura appena descritta alla funzione dell'Esempio 2.1 per calcolare il marginale $g_{x_1}(x_1)$.

$$\begin{aligned} f(x_1, x_2, x_3, x_4, x_5, x_6, x_7) =& f_1(x_1, x_2, x_3) f_2(x_1, x_4) f_3(x_1, x_6, x_7) \\ & \cdot f_4(x_4, x_5) f_5(x_4) f_6(x_2) \end{aligned}$$

Ripetendo il procedimento descritto è possibile calcolare qualsiasi altro marginale.

1. **Inizializzazione** I nodi foglia sono x_3,x_5,x_6,x_7,f_5 e f_6.

I messaggi iniziali sono quindi:

$$\begin{aligned}
\mu_{x_3 \to f_1}(x_3) &= 1 \\
\mu_{x_5 \to f_4}(x_5) &= 1 \\
\mu_{x_6 \to f_3}(x_6) &= 1 \\
\mu_{x_7 \to f_3}(x_3) &= 1 \\
\mu_{f_6 \to x_2}(x_2) &= f_6(x_2) \\
\mu_{f_5 \to x_4}(x_3) &= f_5(x_4)
\end{aligned}$$

I messaggi trasmessi sono evidenziati in Figura 2.6.

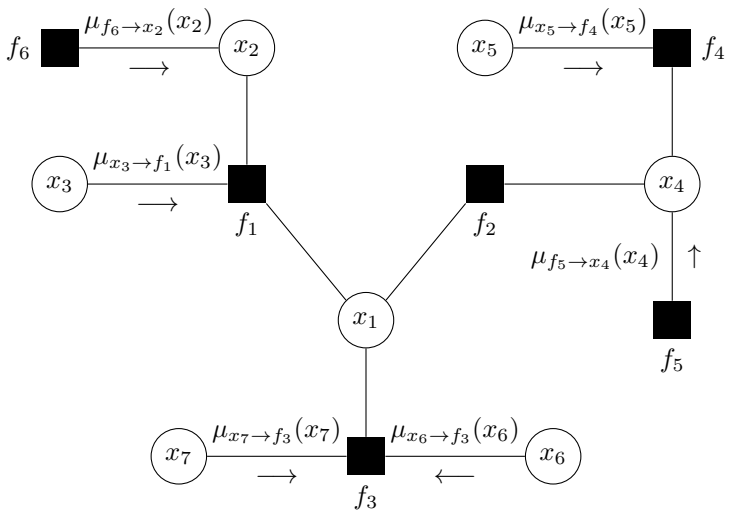


Figura 2.6: Inizializzazione

2. **Calcolo dei messaggi**

   - *Nodo* f_3: f_3 ha grado 3 e ha ricevuto 2 messaggi, può calcolare ed inviare $\mu_{f_3 \to x_1}(x_1)$:

   $$\mu_{f_3 \to x_1}(x_1) = \sum_{\sim\{x_1\}} f_3(x_1, x_6, x_7)\mu_{x_6 \to f_3}(x_6)\mu_{x_7 \to f_3}(x_7)$$

   - *Nodo* f_4: f_4 ha grado 2 e ha ricevuto 1 messaggio, può calcolare ed inviare $\mu_{f_4 \to x_4}(x_4)$:

   $$\mu_{f_4 \to x_4}(x_4) = \sum_{\sim\{x_4\}} f_4(x_4, x_5)\mu_{x_5 \to f_4}(x_5)$$

- *Nodo* x_2: x_2 ha grado 2 e ha ricevuto 1 messaggio, può calcolare ed inviare $\mu_{x_2 \to f_1}(x_2)$:

$$\mu_{x_2 \to f_1}(x_2) = \mu_{f_6 \to x_2}(x_2)$$

I messaggi trasmessi sono evidenziati in Figura 2.7.

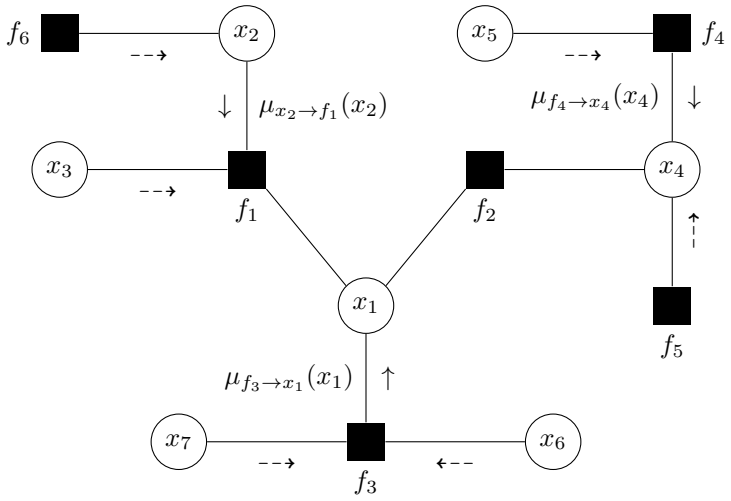


Figura 2.7: Calcolo dei messaggi

- *Nodo* f_1: f_1 ha grado 3 e ha ricevuto 2 messaggi, può calcolare ed inviare $\mu_{f_1 \to x_1}(x_1)$:

$$\mu_{f_1 \to x_1}(x_1) = \sum_{\sim\{x_1\}} f_1(x_1, x_2, x_3)\mu_{x_2 \to f_1}(x_2)\mu_{x_3 \to f_1}(x_3)$$

- *Nodo* x_4: x_4 ha grado 3 e ha ricevuto 2 messaggi, può calcolare ed inviare $\mu_{x_4 \to f_2}(x_4)$:

$$\mu_{x_4 \to f_2}(x_4) = \mu_{f_4 \to x_4}(x_4)\mu_{f_5 \to x_4}(x_4)$$

- *Nodo* f_2: f_2 ha grado 2 e ha ricevuto 1 messaggio, può calcolare ed inviare $\mu_{f_2 \to x_1}(x_1)$:

$$\mu_{f_2 \to x_1}(x_1) = \sum_{\sim\{x_1\}} f_2(x_1, x_4)\mu_{x_4 \to f_2}(x_4)$$

- *Nodo* x_1: x_1 ha grado 3 e ha ricevuto 2 messaggi, può calcolare ed inviare $\mu_{x_1 \to f_2}(x_1)$:

$$\mu_{x_1 \to f_2}(x_1) = \mu_{f_1 \to x_1}(x_1)\mu_{f_3 \to x_1}(x_1)$$

I messaggi trasmessi sono evidenziati in Figura 2.8.

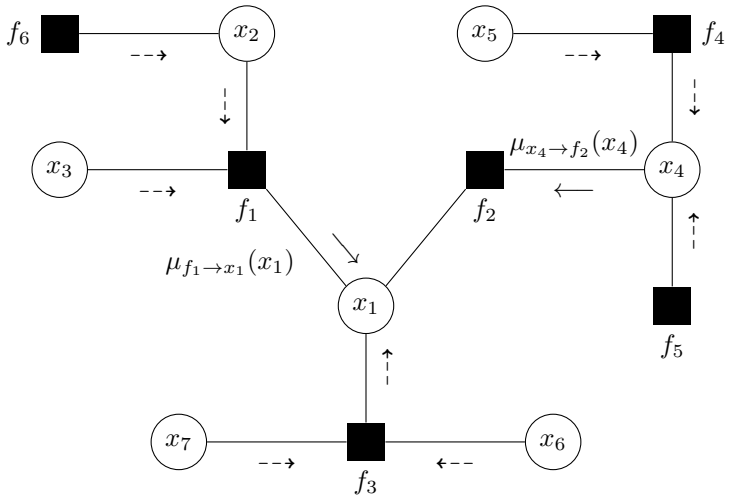


Figura 2.8: Calcolo dei messaggi - Terza parte

3. **Fase finale** Ora si procede al calcolo del marginale $g_{x_1}(x_1)$ attraverso il prodotto dei messaggi che passano sull'arco (x_1, f_2)

$$g_{x_1}(x_1) = \mu_{x_1 \to f_2}(x_1)\mu_{f_2 \to x_1}(x_1)$$

I messaggi trasmessi sono evidenziati in Figura 2.9.

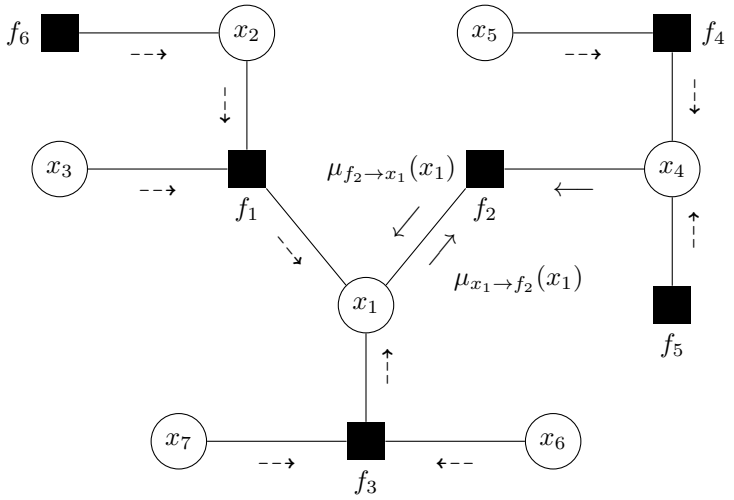


Figura 2.9: Calcolo del marginale $g_{x_1}(x_1)$

Esempio 2.5. Si consideri la funzione g esprimibile nel seguente modo:

$$\begin{aligned} g(x_1, x_2, x_3, x_4, x_5) =& f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) \\ & \cdot f_D(x_3, x_4) f_E(x_3, x_5) \end{aligned} \tag{2.10}$$

Riprendendo la notazione utilizzata in Sezione 2.1.1,
si ha $J = \{A, B, C, D, E\}$, $X_A = \{x_1\}$, $X_B = \{x_2\}$, $X_C = \{x_1, x_2, x_3\}$, $X_D = \{x_3, x_4\}$ ed $X_E = \{x_3, x_5\}$. La corrispondente rappresentazione con factor graph è rappresentata in Figura 2.10.

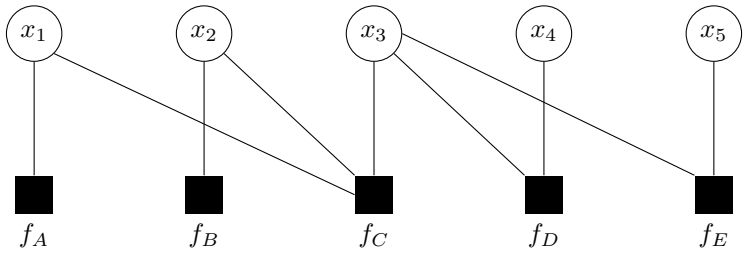


Figura 2.10: Factor graph di 2.10

Si supponga di voler conoscere la funzione marginale:

$$\begin{aligned} g_1(x_1) =& f_A(x_1) \left(\sum_{x_2} f_B(x_2) \left(\sum_{x_3} f_C(x_1, x_2, x_3) \right.\right. \\ & \left.\left. \cdot \left(\sum_{x_4} f_D(x_3, x_4) \right) \left(\sum_{x_5} f_E(x_3, x_5) \right) \right) \right) \end{aligned} \tag{2.11}$$

che, utilizzando la notazione precedentemente introdotta, può essere espressa anche come:

$$\begin{aligned} g_1(x_1) =& f_A(x_1) \times \sum_{\backsim\{x_1\}} \left(f_B(x_2) f_C(x_1, x_2, x_3) \right. \\ & \left. \times \left(\sum_{\backsim\{x_3\}} f_D(x_3, x_4) \right) \times \left(\sum_{\backsim\{x_3\}} f_E(x_3, x_5) \right) \right) \end{aligned} \tag{2.12}$$

Per procedere al calcolo della funzione cercata, si può utilizzare il seguente procedimento:

- Si costruisce il factor graph in modo che x_1 sia la radice del grafo (deve essere senza cicli)

- Ogni variabile foglia invia un messaggio fittizio con la funzione identità al proprio genitore, ogni foglia nodo funzione invia al proprio genitore una descrizione della funzione stessa
- Ogni vertice attende di ricevere messaggi da tutti i suoi figli prima di calcolare ed inviare il proprio al relativo genitore
- Il calcolo termina quando si raggiunge il nodo x_1 al quale, facendo il prodotto di tutti i messaggi ricevuti, si ottiene $g_1(x_1)$.

In particolare: ogni nodo variabile invia il prodotto dei messaggi ricevuti dai proprio figli mentre un nodo funzione, con genitore x, calcola il prodotto tra f stessa e i messaggi ricevuti dai figli, ovvero ottiene ciò che viene rappresentato da $\sum_{\sim\{x\}}$.

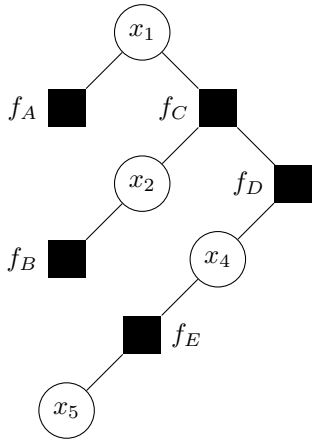


Figura 2.11: Rappresentazione ad albero del factor graph di 2.10

Il calcolo di $g_i(x_i)$, per tutti i valori di i contemporaneamente, può essere fatto in modo efficiente sovrapponendo tutti i risultati ottenuti variando tutti i possibili vertici radice. L'algoritmo terminerà non appena su ogni lato saranno passati due messaggi, uno per ogni direzione. Il valore di $g_i(x_i)$ si otterrà sommando tutti i messaggi entranti al nodo x_i, l'algoritmo descritto è detto *sum-product*.

2.1.5 Forney factor graphs

Si consideri una funzione f, anche detta *funzione globale*, fattorizzabile nel prodotto di sottofunzioni, dette anche *fattori*, ovvero nel caso di

$$f(u, w, x, y, z) = f_1(u, w, x) f_2(x, y, z) f_3(z) \tag{2.13}$$

f è la funzione globale mentre f_1,f_2 ed f_3 sono i fattori.
Secondo la definizione utilizzata da Forney in [32], un factor graph, sia abbreviato in FFG per distinguerlo dal precedente modello, è costituito da nodi, lati e mezzi lati ovvero connessi a un unico nodo) e la sua struttura obbedisce alle seguenti regole:

- Esiste un unico nodo per ogni fattore
- Esiste un unico lato o mezzo-lato per ogni variabile
- Il nodo che rappresenta un certo fattore g è connesso al lato (o mezzo-lato) che rappresenta la variabile x se e solo se g è una funzione di x

Per distinguere questa rappresentazione dalla precedente, i nodi funzione saranno rappresentati con dei quadrati indicanti il nome della funzione al loro interno.
Questa definizione assume implicitamente che ogni variabile non compaia in più di due fattori. La seguente definizione fornisce una procedura più generica utile per la costruzione di FFG di funzioni con variabili che compaiono in più di due fattori.

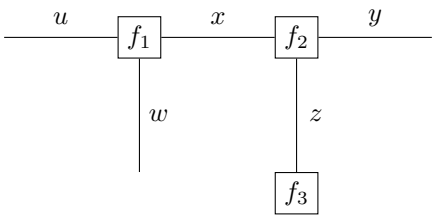


Figura 2.12: Forney factor graph di 2.13

Definizione 2.11. Data una fattorizzazione per la funzione f del tipo

$$f(x_1, x_2, \cdots, x_n) = \prod_{k=1}^{K} f_k(S_k)$$

il corrispondente FFG è costruito a partire dal factor graph normale attraverso la procedura seguente:

- Ogni nodo variabile x_i di grado 1 viene rimosso ed al suo posto viene creato un mezzo-lato etichettato con x_i
- Ogni nodo variabile x_j di grado 2 con $\mathcal{N}(x_j) = \{f_k, f_l\}$ viene rimosso insieme ai suoi due lati adiacenti e i nodi funzione f_k ed f_l vengono tra loro connessi dirrettamente da un lato etichettato x_j

- Ogni variabile x_m di grado $D > 2$ con $\mathcal{N}(x_m) = \{f_{k_1}, f_{k_2}, \cdots, f_{k_D}\}$ viene sostituita da un nodo identità

Uno dei vantaggi di questa rappresentazione è che la regola di aggiornamento dei messaggi dell'algoritmo sum-product è di più facile rappresentazione.

Esempio 2.6. Uno dei campi nei quali sono maggiormente sfruttati i FFG è quello dei modelli probabilistici. La funzione densità di probabilità congiunta per tre variabili è esprimibile e rappresentabile come segue:

$$p_{XYZ}(x, y, z) = p_X(x)p_{Y|X}(y|x)p_{Z|Y}(z|y)$$

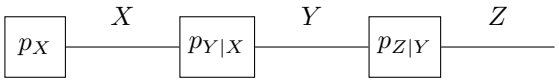


Figura 2.13: Funzione densità di probabilità rappresentata con FFG

Esempio 2.7. Rimanendo nell'ambito delle probabilità, si supponga di voler conoscere la probabilità marginale della seguente funzione

$$\begin{aligned} f(x_1, ..., x_8) =& f_1(x_1)f_2(x_2)f_3(x_1, x_2, x_3, x_4)f_4(x_4, x_5, x_6) \\ & \cdot f_5(x_5)f_6(x_6, x_7, x_8)f_7(x_7) \end{aligned} \tag{2.14}$$

ovvero

$$p_{x_4} = \sum_{x_1, x_2, x_3, x_5, x_6, x_7, x_8} f(x_1, ..., x_8) \tag{2.15}$$

oppure di definire la quantità cercata nel modo seguente:

$$p_{x_4} \stackrel{def}{=} \max_{x_1, x_2, x_3, x_5, x_6, x_7, x_8} f(x_1, ..., x_8). \tag{2.16}$$

Allora inserendo 2.14 in 2.15 si ottiene:

$$p_{x_4} = \underbrace{\left(\sum_{x_1} \sum_{x_2} \sum_{x_3} f_3(x_1, x_2, x_3, x_4) f_1(x_1) f_2(x_2) \right)}_{\mu_{f_3 \to x_4}} \cdot \underbrace{\left(\sum_{x_5} \sum_{x_6} f_4(x_4, x_5, x_6) f_5(x_5) \underbrace{\left(\sum_{x_7} \sum_{x_8} f_6(x_6, x_7, x_8) f_7(x_7) \right)}_{\mu_{f_6 \to x_6}} \right)}_{\mu_{f_4 \to x_4}} \tag{2.17}$$

in cui il raggruppamento secondo le parentesi corrisponde al tracciare le linee tratteggiare nel FFG sottostante:

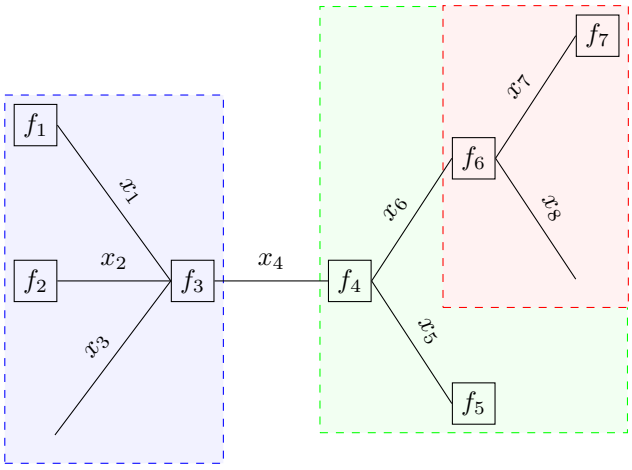


Figura 2.14: Nodi raggruppati nel calcolo del marginale $g_{x_1}(x_1)$

Se si considerano i termini tra perentesi come dei messaggi che escono dal corrispondente riquadro tratteggiato, si evidenziano le funzioni $\mu_{f_i \to x_j}$. I mezzi lati non inviano messaggi, contribuirebbero con un messaggio unitario che può essere ignorato nel prodotto.

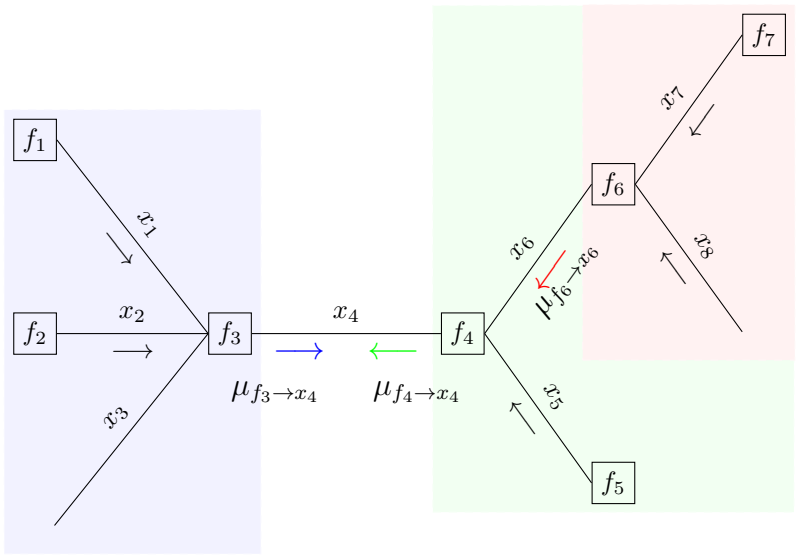


Figura 2.15: Visualizzazione dei messaggi in uscita per il calcolo di p_{x_4}

2.2 Generalized Distributive Law

Fino ad ora, sono state considerate funzioni con dominio $X_1 \times X_2 \times \cdots \times X_n \to \mathbb{R}$ e le operazioni di somma (indicate con $\sum, \int$ e $+$) e prodotto (indicate con $\prod, \times$) ma l'algoritmo sum-product può essere applicato anche a costrutti più generici i quali formano il framework conosciuto come GDL (Generalized Disributive Law) [36]

La procedura generica per lo scambio di messaggi su grafo definita dalla GDL, può essere applicata a insiemi F dotati delle operazioni di somma $\oplus$ e prodotto $\otimes$ tali che $(F, \oplus, \otimes)$ formino un semianello commutativo, ovvero un insieme in cui le operazioni soddisfino le seguenti:

- $\oplus$ è associativa e commutativa
- $\otimes$ è associativa e commutativa
- $\otimes$ è distributiva rispetto a $\oplus$: per ogni $a, b, c \in F$ vale $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$

Esempio 2.8. La tripla (ℝ,max,+) forma un semianello commutativo infatti:

- max è associativa

$$max(a, max(b, c)) = max(max(a, b), c)$$

- max è commutativa

$$max(a, b) = max(b, a)$$

- l'elemento neutro per max, $e_\oplus$, ovvero quello tale che $max(a, e_\oplus) = a$ è $e_\oplus = -\infty$
- $+$ è associativa e commutativa con elemento neutro $e_\otimes = 0$
- $+$ è distributiva rispetto a max infatti

$$a + max(b, c) = max(a + b, a + c)$$

Quando SPA si applica sul semianello (ℝ,max,+) prende il nome di *Algoritmo max-sum.*
Alcuni esempi di semianelli noti sono rappresentati in Tabella 2.1.

La proprietà distributiva fornisce un metodo alternativo *più veloce* per il calcolo di un'espressione: mentre da un lato le operazioni sono tre, dall'altro sono solamente due.
La GDL, che si avvale di questa proprietà, è quindi in grado di ridurre il numero di operazioni $\oplus$ ed $\otimes$ effettuate per certe classi di problemi computazionali.

Insieme	$(\oplus, el_{neutro})$	$(\otimes, el_{neutro})$	Nome
A	(+,0)	($\cdot$,1)	
A[x]	(+,0)	($\cdot$,1)	
A[x,y, $\cdots$]	(+,0)	($\cdot$,1)	
$[0, \infty)$	(+,0)	($\cdot$,1)	
$(0, \infty]$	(min, ∞)	($\cdot$, 1)	sum-product
$[0, \infty)$	(max,0)	($\cdot$,1)	min-product
$(-\infty, \infty]$	(min,∞)	(+,0)	max-product
$[-\infty, \infty)$	(max,$-\infty$)	(+,0)	max-sum
0,1	(OR,0)	(AND,1)	Boolean
2^S	($\cap$,$\emptyset$)	($\cup$,S)	
Λ	($\vee$,0)	($\wedge$,1)	
Λ	($\wedge$,1)	($\vee$,0)	

Tabella 2.1: Esempi di semianelli commutativi.

All'interno di questo framework ricadono diversi algoritmi e la sua applicazione copre i campi più vari (alcuni specifici esempi dell'applicazione di max-sum saranno proposti nel Capitolo 4). Tra i più noti: l'algortitmo di Baum–Welch, la Fast Fourier Transform (FFT) su un gruppo abeliano finito, l'algoritmo di decoding di Gallager–Tanner–Wiberg, di Viterbi, BCJR, il *belief propagation* di Pearl, quello di propagazione di probabilità di Shafer–Shenoy e quelli di turbo decoding [36].

2.3 Algoritmo max-sum

Data una funzione f definita in $X_1 \times X_2 \times \cdots \times X_n \to \mathbb{R}$, fattorizzabile con fattorizzazione aciclica:

$$f(X_1, X_2, \ldots, X_n) = \sum_{k=1}^{N} f_k(S_k) \tag{2.18}$$

L'algoritmo, analogamente a SPA, si articola in tre fasi:

1. **Inizializzazione**

   - Ogni nodo foglia funzione f_k incidente in x_m, trasmette il messaggio $\mu_{f_k \to x_m}(x_m) = f_k(x_m)$
   - Ogni nodo foglia di tipo variabile x_n incidente in f_l, trasmette il messaggio $\mu_{x_n \to f_l}(x_m) = 0$

2. **Regole per il calcolo dei messaggi**

   - Ogni nodo funzione f_k di grado D, dopo aver ricevuto in ingresso $D-1$ messaggi distinti provenienti dai nodi variabili $x_i \in \mathcal{N}(f_k)$, trasmette in uscita il messaggio $\mu_{f_k \to x_m}(x_m)$ al nodo rimanente x_m secondo

$$\mu_{f_k \to x_m}(x_m) = \max_{\backsim\{x_m\}} \left\{ f_k\left(\{x_m\}_{x_m \in \mathcal{N}(f_k)}\right) + \sum_{x_n \in \mathcal{N}(f_k)\{x_m\}} \mu_{x_n \to f_k}(x_n) \right\} \tag{2.19}$$

3. **Fase finale**

   - Per procedere al calcolo del marginale di x_n, considerare una $f_k \in \mathcal{N}(x_n)$ e sommare i messaggi trasmessi lungo l'arco (x_n, f_k) secondo

$$g_{x_n}(x_n) = \mu_{f_k \to x_n}(x_n) + \mu_{x_n \to f_k}(x_n) \tag{2.20}$$

Il significato del marginale così ottenuto è quello di determinare il massimo di una funzione, inteso come valore massimo che essa può assumere senza considerare la variabile x_i, e non come massimo dei valori delle singole variabili.

$$g_{x_i}(x_i) = \max_{\backsim\{x_i\}} f(x_1, x_2, \cdots, x_N) \tag{2.21}$$

Infatti il massimo della funzione complessiva può essere ottenuto per ogni i come:

$$\max_{x_1, x_2, \cdots, x_N} f(x_1, x_2, \cdots, x_N) = \max_{x_i} g_{x_i}(x_i) \tag{2.22}$$

Se, come accade per esempio nei problemi di ottimizzazione, si è interessati al valore di $\mathbf{x} = [x_1, \cdots, x_n]$ che permette ad f di raggiungere il massimo si può sfruttare l'algoritmo max-sum come segue.
Sia $\hat{\mathbf{x}} = [\hat{x_1}, \cdots, \hat{x_n}]$ il valore di $\mathbf{x}$ che massimizza f, e sia

$$\hat{x_i} = \operatorname*{argmax}_{x_i} g_{x_i}(x_i) \tag{2.23}$$

allora

$$g_{x_i}(\hat{x_i}) = \max_{x_1, \cdots, x_N} f(x_1, \cdots, x_N) \tag{2.24}$$

e se $\hat{\mathbf{x}}$ è unico, si ha

$$\hat{\mathbf{x}} = \operatorname*{argmax}_{x_1, \cdots, x_N} = f(x_1, \cdots, x_N) \tag{2.25}$$

Nel caso in cui la funzione abbia più massimi, i valori di $\hat{\mathbf{x}}$ che li producono possono essere trovati considerando tutte le combinazioni che massimizzano i marginali.

2.3.1 Da sum-product a max-sum

Per talune funzioni si preferisce l'utilizzo dell'algoritmo max-sum a quello di sum-product per evitare problemi di underflow quando si ha a che fare con valori particolarmente piccoli. Poichè gli operatori log e max soddisfano la seguente

$$\log\left(\max_x f(x)\right) = \max_x \log f(x)$$

e con il passaggio al logaritmo si mantiene valida la proprietà distributiva

$$\log\left(\max(a+b, a+c)\right) = \log\left(a + \max(b, c)\right)$$

una funzione globale che è esprimibile come prodotto di sottofunzioni, può essere espressa come somma del logaritmo delle medesime sottofunzioni. Ovvero, una funzione esprimibile come

$$f(x) = \prod_{j=0}^{k} f_j(X_j)$$

diventa

$$f(x) = \sum_{j=0}^{k} \log f_j(X_j)$$

Max-product è l'algoritmo utilizzato per trovare

$$\underset{\mathbf{x}}{\operatorname{argmax}} \prod_{j} f_j(X_j)$$

che, spostandosi nello spazio logaritmico diventa

$$\underset{\mathbf{x}}{\operatorname{argmax}} \sum_{j} \log(f_j(X_j)$$

ma, poichè la funzione logaritmo è monotona crescente, cercare l'argomento che massimizza la somma di logaritmi delle funzioni equivale a cercare l'argomento che massimizza la somma delle funzioni stesse, ovvero

$$\underset{\mathbf{x}}{\operatorname{argmax}} \sum_{j} f_j(X_j)$$

E quindi le regole per lo scambio dei messaggi diventano:

1. **Inizializzazione**
   I messaggi inviati dai nodi foglia sono

$$\begin{aligned} \mu_{f_k \to x_m}(x_m) &= \log f_k(x_m) \\ \mu_{x_n \to f_l}(x_m) &= 0 \end{aligned} \tag{2.26}$$

2. **Regole per il calcolo dei messaggi**
   Passata la fase iniziale i messaggi scambiati seguono la seguente

$$\begin{aligned} \mu_{f_k \to x_m}(x_m) &= \max_{\backsim\{x_m\}} \left\{ \log f_k\left(\{x_m\}_{x_m \in \mathcal{N}(f_k)}\right) + \sum_{x_n \in \mathcal{N}(f_k)\{x_m\}} \mu_{x_n \to f_k}(x_n) \right\} \\ \mu_{x_n \to f_l}(x_m) &= \sum_{x_n \in \mathcal{N}(f_k)\{x_m\}} \mu_{f_l \to x_n}(x_n) \end{aligned} \tag{2.27}$$

3. **Fase finale**
   Analoga a quella definita in 2.20.

2.4 Algoritmo max-sum per funzioni a variabili binarie

L'algoritmo max-sum risolve problemi di ottimizzazione in modo corretto o approssimato, rispettivamente a seconda che il factor graph associato sia o meno un albero. Negli ultimi anni è aumentato lo studio finalizzato alla ricerca di quali tipologie di problemi, e quindi quali funzioni, ne permettessero l'utilizzo ottenendo complessità polinomiali, invece che esponenziali nel numero di variabili del dominio della funzione, e questo ha portato al definire le funzioni di questi, **Tractable High Order Potentials** (THOPs) [45] [47].
Il beneficio, per ciò che concerne la complessità, è evidente nei grafi con elevato numero di nodi e, perciò, non particolarmente apprezzabile nello scenario cui fa riferimento questa tesi (è altamente improbabile che una challenge multirobot possa coinvolgere più di cinque o sei robot considerato il loro costo). Tuttavia, queste funzioni, ma in particolare la possibilità di ristrutturare il factor graph che esse apportano, renderà di comprensione più immediata i concetti collaborazione e *load*, trattati in Sezione 3.2.3.

Una categoria di problemi che trae particolare beneficio da questo è quella dei problemi relativi a sistemi multi agente, all'interno dei quali la rapidità del calcolo è elemento chiave [45] [46].

Nel contesto descritto quelle che erano le funzioni per il classico algoritmo max-sum, prendono anche il nome di *potenziali*, e l'algoritmo è detto anche *max-sum binario*.
La trasformazione di un'istanza di problema per il max-sum classico ad istanza per quello binario avviene come descritto:

- **Variabili**: una variabile con un dominio finito di cardinalità d diventa un insieme di d variabili booleane tutte connesse tra loro attraverso una funzione che assicura che solo una di esse sia attiva (se non diversamente specificato).

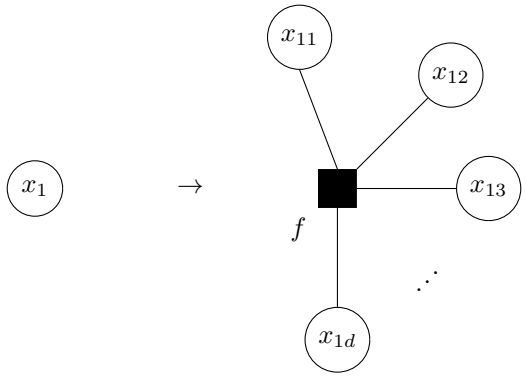


Figura 2.16: Trasformazione di nodo variabile per max-sum binario

- **Funzioni**: una funzione connessa a k variabili può avere nel caso peggiore d^k valori nel caso di algoritmo normale, passando al max-sum binario invece questo valore cresce fino a $2^{d \times k}$.

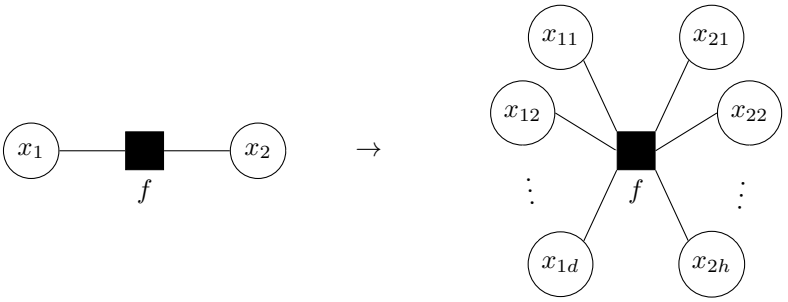


Figura 2.17: Trasformazione di nodo funzione per max-sum binario

- **Messaggi**: la procedura per il calcolo dei messaggi rimane la medesima

Come si evince da quanto appena esposto, il passaggio dalla prima alla seconda variante di max-sum comporta: un aumento nel numero di variabili; un incremento lineare in quello delle funzioni ed uno esponenziale nella taglia dei nuovi potenziali (e conseguentemente nella taglia della rappresentazione del problema) ed infine una semplificazione per quanto riguarda il calcolo dei messaggi.

In genere, il numero di variabili per funzione è elevato ma ci sono alcuni potenziali, i sopracitati THOPs che rendono polinomiale il calcolo dei messaggi.
Il tempo di calcolo dei messaggi funzione-variabile è esponenziale nel numero di fattori ma, attraverso la semplificazione che si ottiene coinvolgendo variabili binarie, diventa lineare.
Invece di inviare un messaggio per ognuno dei valori della variabile (1 e 0) è possibile inviarne un unico con la differenza tra i due, ovvero:

$$m_{i \to j} = \mu_{i \to j}(1) - \mu_{i \to j}(0) \tag{2.28}$$

dove i e j sono la coppia variabile/funzione o viceversa. Così facendo, ricevuto il messaggio, si può considerare $\mu_{i \to j}(0) = 0$ e il valore del messaggio pari a quello di $\mu_{i \to j}(1)$.

2.4.1 THOPs per max-sum binario

Vengono ora esposti alcuni tra i potenziali di maggior interesse per l'utilizzo di max-sum binario [45].

OneAndOnlyOne

$$OneAndOnlyOne(x) = \begin{cases} 0 & if \sum_{k=1}^{N} x_k = 1 \\ -\infty & altrimenti \end{cases}$$

Questa funzione richiede che solo una delle variabili binarie sia attiva, in questo caso il tempo per il calcolo dei potenziali è lineare e la funzione è detta *selettore*.

Potenziali di cardinalità

Sono funzioni il cui valore dipende solo dal numero di variabili attive e non da quali esse siano.

$$CardinalityPotential(X) = f\left(\sum_{x_i \in X} x_i\right)$$

Il calcolo dei messaggi in questo caso avviene in $O(N \log N)$.

Potenziali pesati

Sono funzioni che specificano un peso indipendente per ognuna delle variabili da cui dipende il loro valore. Dato l'insieme delle variabili $X = \{x_1, x_2, \cdots, x_k\}$ e un insieme di valori reali (pesi) $W = \{w_1, w_2, \cdots, w_k\}$, le funzioni sono definite come

$$Weighting(X, W) = \sum_{i=0}^{k} x_i w_i$$

Potenziali di uguaglianza

Sono definiti su una coppia di variabili x_i e x_j (ma si può estendere la definizione al numero desiderato di variabili) e richiedono che entrambe assumano lo steso valore. Il loro valore è calcolato in tempo costante.

$$Equality(x_i, x_j) = \begin{cases} 0 & if x_i = x_j \\ -\infty & altrimenti \end{cases}$$

Potenziali composti

Rappresentano la composizione condizionata di più THOPs: dato un potenziale il cui valore dipende dall'insieme di variabili X e un insieme di variabili condizionanti $C \subseteq X$, le rimanenti $Y = X - S$ formano un altro THOPs.

$$CompositePotential(C, X) = THOP_C(X)$$

AllOne e AllZero

Sono funzioni che richiedono che tutte le variabili siano attive o, rispettivamente, tutte non attive. Entrambi i potenziali sono calcolati in tempo polinomiale e sono ridondanti in quanto le corrispondenti variabili potrebbero essere sostituite da tutti valori 1 o, rispettivamente, 0.

$$AllOne(x) = \begin{cases} 0 & if \sum_{k=1}^{N} x_k = N \\ -\infty & altrimenti \end{cases}$$

$$AllZero(x) = \begin{cases} 0 & if \sum_{k=1}^{N} x_k = 0 \\ -\infty & altrimenti \end{cases}$$

AllEqual

Questi potenziali richiedono che tutte le variabili siano uguali tra loro, a 0 o 1.

$$AllEqual(x) = \begin{cases} 0 & if x_j = x_{j+1} 1 \leq j < N \\ -\infty & altrimenti \end{cases}$$

AtMostOne

Richiede che al più una variabile binaria dell'insieme si attiva.

$$AtMostOne(x) = \begin{cases} 0 & if \sum_{k=1}^{N} x_k \leq 1 \\ -\infty & altrimenti \end{cases}$$

AtLeastOne

Richiede che almeno una variabile binaria dell'insieme si attiva.

$$AtLeastOne(x) = \begin{cases} 0 & if \sum_{k=1}^{N} x_k \geq 1 \\ -\infty & altrimenti \end{cases}$$

Capitolo 3

Problema della distribuzione dei task

In occasione della presentazione della domanda di partecipazione a RoCKIn Camp 2014, ad ogni team è stato chiesto di descrivere ciò che esso auspicava di ottenere dalla partecipazione e di proporre eventuali idee per lo sviluppo di nuovi scenari per la challenge o il miglioramento di quelli esistenti.
In quella occasione il team con il quale ho partecipato ha deciso di proporre una possibile estensione della gara a sistemi multi robot ed è a questa idea che fa riferimento la tesi.

3.1 Max-sum ed i sistemi multiagente

Se i potenziali, o funzioni, f_j precedentemente utilizzati sono interpretati in termini di utilità rispetto alla configurazione di variabili da cui dipendono, il problema risolto da max-sum è un *constraint optimization problem* (COP) e l'assegnamento $\text{argmax}_{\mathbf{x}} = \sum_j f_j(\mathbf{x})$ è quello che permette al sistema di raggiungere la sua massima utilità.

É possibile applicare l'algoritmo max-sum su factor graph, per esempio, al problema del coordinamento in un sistema multi robot.
Questo può avvenire definendo opportunamente la funzione globale ed i fattori in modo che essi rappresentino l'utilità che il robot in esame ha nel seguire una particolare traiettoria o, più genericamente, una strategia. Si consideri un sistema multirobot con N agenti, ognuno dei quali deve dirigersi verso uno specifico goal (al robot r_i corrisponderà il goal g_i). Si ridefinisce quindi la funzione globale come:

$$U(x_1, x_2, ...x_m) = \sum_{i=1}^{N} u_i(X_i) \tag{3.1}$$

dove le u_i è la funzione utilità di r_i legata al goal g_i ed X_i è il dominio di u_i. Le funzioni utilità dei singoli robot possono essere definite, per esempio, in modo da considerare posizione e velocità del robot; in questo caso il dominio della funzione potrebbe tenere in considerazione sono gli elementi (ostacoli ed altri robot) che si trovano entro un certo raggio dal robot considerato.

In riferimento al contesto della challenge, si consideri un sistema multi robot ed un insieme di task che la challenge richiede di eseguire. Ad ognuno dei task sia assegnato, per regolamento, un punteggio che decresce all'aumentare del tempo necessario affinché il compito sia eseguito (e.g. punteggio inversamente proporzionale alla distanza del robot dal task, supponendo che tutti i robot possano muoversi con la medesima velocità) ed ogni robot si trovi nelle condizioni di eseguire i task, entro un certo raggio da lui, per i quali esso possiede i sensori e/o le dotazioni richieste. In quest'ottica, l'algoritmo max-sum può essere utilizzato al fine di ottenere la configurazione di assegnazioni robot-task che massimizza il punteggio ottenuto.
Questo approccio può essere confrontato con una strategia che preveda l'assegnazione del robot al task più vicino o quello con punteggio maggiore per lui tuttavia queste strategie non garantirebbero il punteggio maggiore né l'assenza di task non eseguiti nel caso in cui più robot si dirigano verso lo stesso obiettivo.

3.2 Modellazione del sistema

La modellazione del contesto appena descritto avviene in modo differente a seconda della versione dell'algoritmo max-sum utilizzato.

3.2.1 Istanza per max-sum originario

I task sono rappresentati nel factor graph come nodi funzione ed i robot come nodi variabile. Ogni robot è collegato ai task che può eseguire per dotazione sensoristica ed entro un determinato raggio.
Per rendere più immediato il reperimento delle informazioni sull'assegnazione dei robot ai task (visualizzato nell'output), i possibili valori che può assumere una variabile sono gli interi corrispondenti agli id dei task cui esso è collegato.
Se R_1 può eseguire i task t_2, t_3 e t_4 allora la variabile che rappresenta nel factor graph il robot, x_1, avrà dominio $D_1 = \{2, 3, 4\}$

Quando viene specificata l'istanza del problema, è necessario fornire tutte le combinazioni di valori possibili per ogni funzione. Il punteggio assegnato ad esse dipende dalle specifiche del task, il valore $-\infty$[1] assume un duplice significato: è presente in corrispondenza delle combinazioni di valori di variabili che non permettono l'esecuzione del task e in quelle in cui un robot compare impegnato in più di un task contemporaneamente.

Creazione dell'istanza a partire dalle specifiche del problema

L'istanza sulla quale viene eseguito l'algoritmo max-sum può essere fornita in due modi:

- Da file di testo con dichiarazione esplicita di funzioni, variabili e valori
- Da file di testo che fornisce le specifiche del problema secondo la formattazione descritta di seguito.

[1] Il valore $-\infty$, nell'implementazione dell'algoritmo, è approssimato con un valore negativo non eccessivamente grande per problemi di convergenza.

Attraverso la chiamata al risolutore specifico, `Aphrodite`, specificando gli operatori *max* e *sum*, l'algoritmo max-sum procede al calcolo delle assegnazioni robot-task che massimizzano il punteggio ottenibile.

Definizione della funzione utilità

La funzione utilità non è stata modificata rispetto all'algoritmo originale: in questo contesto la parola utilità equivale a punteggio, ogni funzione rappresenta il punteggio del task che essa rappresenta. Lo scopo di massimizzare l'utilità equivale quindi a quello di massimizzare la somma di punteggi parziali che corrisponde al punteggio finale del sistema.

Ad ogni configurazione di robot-task è assegnato un punteggio tanto maggiore quanto è minore la distanza del robot dal task. Sono evitate le assegnazioni di più robot al medesimo task ponendo ad un valore $-\infty$ le configurazioni di variabili che rappresentano questa scelta.
L'output visualizzato dal programma propone il punteggio massimo che si può ottenere data la configurazione iniziale del sistema e le assegnazioni robot-task che permettono di conseguirlo.

3.2.2 Istanza per max-sum binario

La formalizzazione del problema nel caso di max sum binario avviene seguendo le specifiche riportate in Sezione 2.4, ovvero:

- **robot**: ogni nodo variabile che prima rappresentava un robot, si suddivide in tante variabili binarie quanti sono i task cui il robot è collegato, e in un nodo selettore il quale permette che ogni robot non possa eseguire più di un task per volta (THOP *OneAndOnlyOne(**x**)* 2.4.1). In questo caso il valore 1 per x_{ij} indica che il robot i è stato assegnato al task j.

  In Figura 3.1 è possibile visualizzare un esempio di trasformazione di nodo variabile x_i connesso a d task diversi: il robot è ora rappresentato da un selettore s_i connesso a tutte le d variabili binarie x_{ij} con $j \in \{1, d\}$, ognuna delle quali è a sua volta connessa al task (rappresentato come indicato in Figura 3.2).

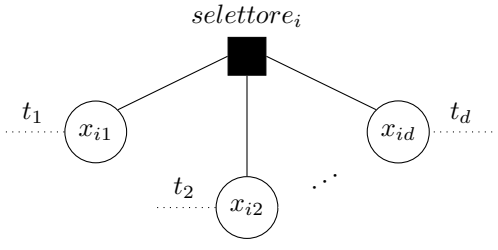


Figura 3.1: Trasformazione di nodo variabile per max-sum binario. Robot x_i connesso a d task.

- **task**: ogni nodo funzione che prima rappresentava un task viene suddiviso in due tipi di nodi funzione.
  - nodi punteggio: sono nodi funzione f_{ij} connessi solo alla variabile x_{ij} della quale indicano il punteggio in caso di assegnazione al task j
  - nodi collaborazione: sono i nodi connessi alle variabili binarie x_{ij} con j fissato, e rappresentano un eventuale incremento/decremento di punteggio per la collaborazione di più robot quando essa è permessa

  In Figura 3.2 è possibile visualizzare la trasformazione che subisce il nodo funzione che rappresenta il task t_k connesso ai robot r_i ed r_j: il task ora è rappresentato da un nodo collaborazione e da tanti nodi punteggio quanti sono i robot cui è connesso. La nuvola tratteggiata indica l'intero nodo variabile robot trasformato secondo le indicazioni appena fornite.

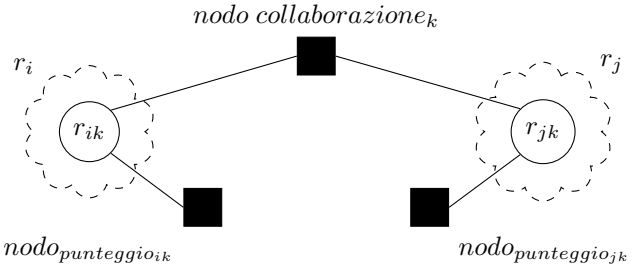


Figura 3.2: Trasformazione di nodo funzione per max-sum binario. Task t_k connesso ai robot r_i ed r_j.

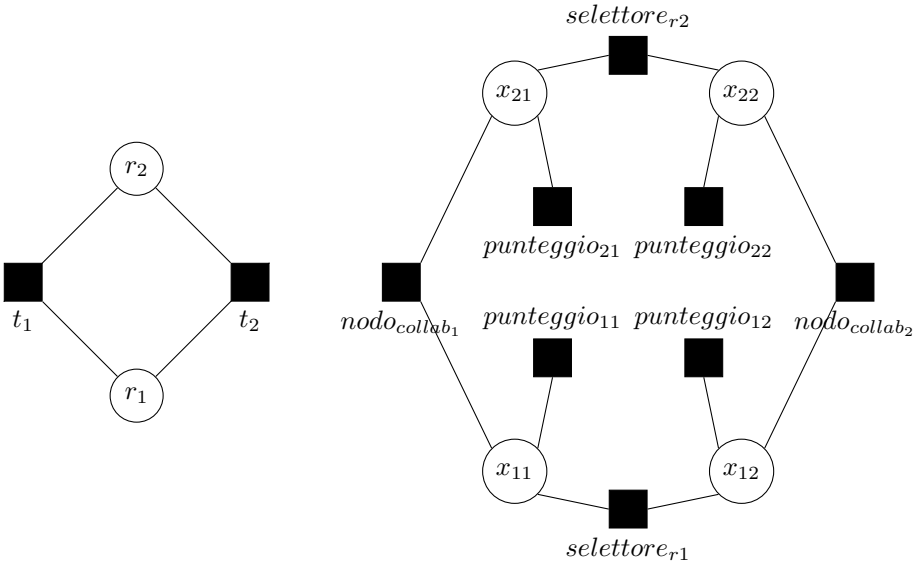


Figura 3.3: Trasformazione di una istanza per max-sum binario

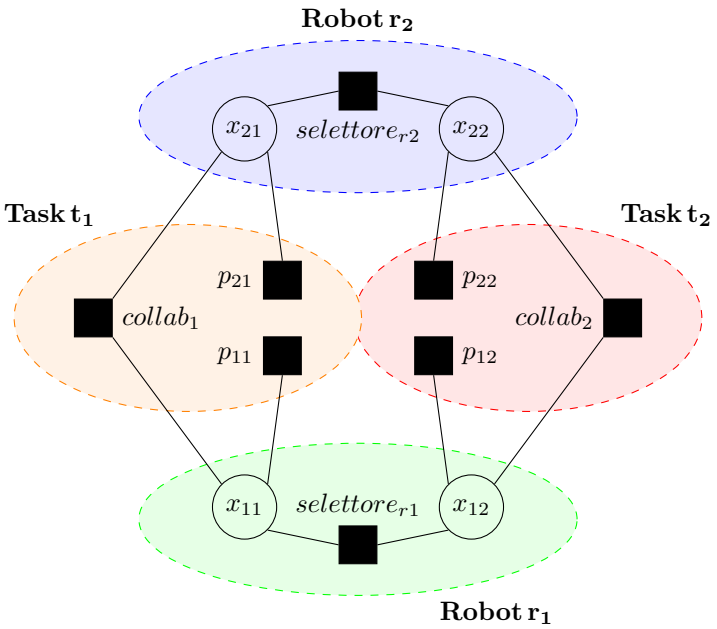


Figura 3.4: Istanza per max-sum binario con elementi evidenziati

3.2.3 Istanza con collaborazione e *load*

Un'istanza di problema di questa tipologia, è quella in cui è ammessa la collaborazione tra robot per eseguire un determinato task e/o l'esecuzione da parte di un robot di più di un task (esecuzione non contemporanea ma sequenziale).
La modellazione delle due possibilità avviene, come descritto di seguito, solo nel caso di istanza di problema binario poiché la loro gestione è più immediata ed efficace, nonché facilmente comprensibile, se applicata ad un factor graph del tipo appena esposto.

Task che permettono la collaborazione

L'entità che *decide* di permettere la collaborazione è il task. I motivi che determinano questa scelta sono vari: la descrizione specifica del task, vincoli di tempo, il numero di robot maggiore rispetto a quello di task e la conseguente volontà di non lasciare *inoccupato* alcun robot. Per questo motivo si è scelto di affidare la gestione di questa collaborazione all'oggetto `Collaboration_Node`.

La possibilità di collaborazione va opportunamente dichiarata nel file di input (secondo le specifiche riportate in Tabella 3.7) e consiste nello specificare, oltre alla possibilità di collaborazione, anche un punteggio aggiuntivo che il task fornisce nel caso di esecuzione in contemporanea da parte di più robot.
Questo punteggio viene riportato nella definizione della funzione del nodo collaborazione moltiplicato per il numero di robot successivi al primo che seguono il task.
Si consideri il task t_i collegato ai nodi robot r_j, r_k ed r_l che fornisce un incremento di n punti per la collaborazione nella propria esecuzione. Esso è rappresentato

dal nodo collaborazione f_{iii}[2] collegato ai nodi variabile x_{ji}, x_{ki} e x_{li} e potrà a assumere i valori indicati di seguito.

$\mathbf{x_{ji}}$	$\mathbf{x_{ki}}$	$\mathbf{x_{li}}$	$\mathbf{f_{iii}}$	**Significato**
0	0	0	$-\infty$	Nessun robot esegue il task
0	0	1	0	Solo r_l esegue il task, nessun incremento
0	1	0	0	Solo r_k esegue il task, nessun incremento
0	1	1	n	r_k ed r_l eseguono il task, incremento $+n$
1	0	0	0	Solo r_j esegue il task, nessun incremento
1	0	1	n	r_j ed r_l eseguono il task, incremento $+n$
1	1	0	n	r_j ed r_k eseguono il task, incremento $+n$
1	1	1	$2 * n$	Tutti i robot eseguono il task, incremento $+2n$

Tabella 3.1: Esempio di tabella di valori di un nodo collaborazione con *bonus*.

Esempio 3.1. Si consideri una situazione in cui il task t_4 possa essere eseguito dai robot r_1, r_2, r_3 e fornisce un incremento di punti per la collaborazione pari a 5. La sua tabella di valori sarà:

$\mathbf{x_{14}}$	$\mathbf{x_{24}}$	$\mathbf{x_{34}}$	$\mathbf{f_{444}}$
0	0	0	$-\infty$
0	0	1	0
0	1	0	0
0	1	1	5
1	0	0	0
1	0	1	5
1	1	0	5
1	1	1	10

Tabella 3.2: Valori del nodo collaborazione f_{444}.

Una ovvia conseguenza della possibilità di avere collaborazione di robot (ma analogo vale per la questione del *load* affrontata più avanti) è che un robot possa eseguire più di un task e che quindi l'assegnazione robot-task finale preveda più step per l'esecuzione.

Robot che impongono un *load* per l'esecuzione di più task

L'entità robot può imporre di accettare di eseguire più di un task per ogni sessione a condizione di una diminuzione di punteggio determinata dal carico di lavoro che gli viene assegnato, detto perciò *load*. Questo può avvenire per scoraggiare quelle situazioni in cui tutti i task scelgano di essere eseguiti dal medesimo robot sbilanciando il carico di lavoro complessivo.

[2]La convenzione adottata per gli id delle funzioni verrà esplicata in Sezione 3.4.3

La disponibilità ad eseguire più task con *load* va dichiarata nel file di input secondo le specifiche di Tabella 3.7 e viene implementata attraverso la struttura del `Selector_node`.
Il robot che dichiara l'intenzione di imporre un carico, fornisce anche un punteggio che va interpretato come unità di decremento da sottrarre in numero pari ai compiti a lui assegnati dopo il primo.
Se il robot r_i può eseguire i task t_j, t_k, t_l e impone un carico pari ad m punti, la tabella del nodo selettore f_i che lo rappresenta avrà i valori:

$\mathbf{x_{ij}}$	$\mathbf{x_{ik}}$	$\mathbf{x_{il}}$	$\mathbf{f_i}$	**Significato**
0	0	0	$-\infty$	Il robot non esegue nessun task
0	0	1	0	Il robot esegue solo t_l, nessun decremento
0	1	0	0	Il robot esegue solo t_k, nessun decremento
0	1	1	$-m$	Il robot esegue t_k e t_l, decremento $-m$
1	0	0	0	Il robot esegue solo t_j, nessun decremento
1	0	1	$-m$	Il robot esegue t_j e t_l, decremento $-m$
1	1	0	$-m$	Il robot esegue t_k e t_j, decremento $-m$
1	1	1	$-2 * m$	Il robot esegue tutti i task, decremento $-2m$

Tabella 3.3: Esempio di tabella di valori di un nodo selettore con *load*.

Esempio 3.2. Si consideri una situazione in cui il robot r_5 possa eseguire i task t_1, t_2, t_3 imponendo un decremento di punti per l'esecuzione di più task pari a 4. La sua tabella di valori sarà:

$\mathbf{x_{51}}$	$\mathbf{x_{52}}$	$\mathbf{x_{53}}$	$\mathbf{f_5}$
0	0	0	$-\infty$
0	0	1	0
0	1	0	0
0	1	1	-4
1	0	0	0
1	0	1	-4
1	1	0	-4
1	1	1	-8

Tabella 3.4: Valori del nodo selettore f_5.

Collaborazione e *load* in istanza non binaria

Nel caso di istanza non binaria di problema, con possibilità di inserire carico e/o collaborazione, l'implementazione su factor graph non binario, richiederebbe di modificare i valori di ogni singolo punteggio. Questi andrebbero modificati in corrispondenza di quei valori di variabili (assegnazioni di robot), che necessitano dell'aggiunta o sottrazione delle quantità specificate come bonus o *load*.

3.3 Implementazione dell'algoritmo

L'algoritmo sviluppato per risolvere il problema del massimo punteggio ottenibile è stato scritto in linguaggio `Java`, a partire da un pacchetto esistente per l'utilizzo dell'algoritmo max-sum su factor graph [3], ed è reperibile presso il seguente repository `https://github.com/elena89/challengemaxsum`.

Il codice è strutturato in modo modulare al fine di essere facilmente estendibile ad applicazioni diverse rispetto a quella sopra esposta, ed è organizzato in package, tra i quali, quelli di interesse per l'applicazione in oggetto sono:

- **olimpo**: fornisce classi e metodi per la gestione dell'input da file e terminale, i metodi risolutori che si interfacciano con max-sum ed il factorgraph
- **factorgraph**: fornisce le classi che gestiscono tutti gli elementi costitutivi del factor graph quali nodi varibile, nodi funzione, lati, agenti
- **operation**: comprende classi ed interfacce per la gestione dei messaggiQ e messaggiR, per la definizione delle operazioni oplus ed otimes generiche e loro specializzazioni
- **function**: ingloba tutte le funzioni utilità definite al fine di poter applicare l'algoritmo max-sum e la gestione della Tabular Function ovvero una tabella che memorizza i legami tra nodi
- **maxsum**: comprende le classi per la gestione dei diversi tipi di istanza di problema
- **challenge**: comprende le classi utili alla definizione dello specifico problema del calcolo del punteggio massimo nell'ottica della possibile applicazione ad una challenge
- **test**: raccoglie le classi di test e le istanze di prova sulle quali può essere testato il funzionamento dell'algoritmo

Il codice 3 era una versione testata unicamente sull'utilizzo di max-sum per un problema inerente l'emissione di CO_2 perciò, nello sviluppo del codice aggiuntivo per la risoluzione del problema del coordinamento, è stato necessario risolvere diversi bug riguardanti la costruzione dell'istanza sulla quale applicare il problema ed il formato dei dati forniti in ingresso.

3.3.1 Formato dei dati in ingresso per i solver di *olimpo*

I dati in ingresso devono essere forniti utilizzando la sintassi in Tabella 3.5:

- Agente: va fornito l'id che lo identifica univocamente
- Variabili: necessitano del loro id, l'id dell'agente cui si riferiscono, il numero di valori che possono assumere e tali valori all'interno di una coppia di parentesi tonde separati da virgole

[3] `https://code.google.com/p/jmaxsum/`

- Funzioni: la definizione di una funzione va fatta in due step. Va prima dichiarato l'id della funzione (voce `CONSTRAINT` della tabella), quello dell'agente di riferimento e quelli delle variabili dal cui valore dipende la funzione; vanno poi definiti i valori della funzione in corrispondenza di tutte le possibili configurazioni di valori delle variabili (voce `F`). É importante che l'ordine col quale vengono dati gli id dei parametri sia lo stesso con cui vengono forniti i valori dei parametri stessi

Oggetto	**Formato**
Agente	`AGENT id_agente`
Variabili	`VARIABLE id_var id_agente num_valori (val`$_1$`, val`$_2$`, ⋯, val`$_n$`)`
Funzioni	`CONSTRAINT id_funzione id_agente id_param`$_1$ ⋯ `id_param`$_m$
	`F val_param`$_{11}$ ⋯ `val_param`$_{m1}$ `val_funzione`$_1$
	`F val_param`$_{12}$ ⋯ `val_param`$_{m2}$ `val_funzione`$_2$
	⋯

Tabella 3.5: Formattazione dei dati in ingresso

In Tabella 3.6 è possibile visualizzare un esempio di file di ingresso corretto.

Istanza
AGENT 1
VARIABLE 1 1 2 (1,3)
VARIABLE 2 1 3 (2,3,4)
VARIABLE 3 1 2 (0,1)
VARIABLE 4 1 1 (1)
CONSTRAINT 1 1 1 4
F 1 1 -inf
F 3 1 10
CONSTRAINT 2 1 3 2
F 0 2 -inf
F 0 3 0
F 0 4 0
F 1 2 -inf
F 1 3 0
F 1 4 0
CONSTRAINT 3 1 1 4 3
F 1 1 0 4
F 1 1 1 13
F 3 1 0 -inf
F 3 1 1 5

Tabella 3.6: Esempio di file di ingresso correttamente formattato.

3.4 Descrizione del package *challenge*

Il package *challenge* riunisce tutti gli oggetti e metodi utili a rappresentare lo scenario descritto in precedenza, cui viene applicato l'algoritmo max-sum. Segue ora una breve descrizione delle caratteristiche di ciascun oggetto/classe seguendo la suddivisione per entità.
In ognuno dei casi a seguire sono stati creati oggetti diversi per la versione ordinaria e binaria dell'algoritmo.

3.4.1 Rappresentazione dei robot

Istanza ordinaria - Robot.java

Nel caso di *max-sum ordinario*, un robot è rappresentato attraverso un oggetto omonimo, dotato dei campi:

- `id`: id del robot;
- `id_zona`: id della zona nella quale si trova il robot;
- `dotazioni`: tipo di dotazioni di cui è fornito espresse come oggetto `String`[4];
- `id_task`: array che riunisce gli id dei task cui è collegato il robot;
- `spec`: array di oggetti `Specification` ovvero coppie <task nel raggio del robot, relativa distanza dal robot> dove la distanza è espressa in generiche unità;
- `points`: array di oggetti `Points` ovvero coppie <task per cui il robot ha le dotazioni corrette, punteggio relativo al task>. Questi oggetti sono creati dal metodo `generateAssignment()` all'interno della classe `Challenge`.

Ogni campo è dotato dei relativi metodi di accesso e modifica.

Istanza binaria

- Robot_BinaryV.java: nella versione binaria dell'algoritmo il robot è un rappresentato da tanti nodi variabile binarie quanti sono i task cui è collegato, ognuno dotato dei campi indicati i quali possiedono metodi di accesso e modifica:

  - `id`: id del robot costruito secondo la convenzione per cui esso è un intero di due cifre di cui la prima è l'id del robot nella versione ordinaria e la seconda è l'id del task cui è connesso;
  - `id_zona`: id della zona nella quale si trova il robot;
  - `id_old_robot`: prima cifra dell'`id`;
  - `id_old_task`: seconda cifra dell'`id`.

- Selector_Node.java: questo oggetto rappresenta il nodo selettore e permette che solo una delle variabili binarie che rappresentano il robot sia attiva. Esso è dotato dei campi:

  - `id`: id del nodo, ovvero secondo la convenzione adottata, esso è un intero con valore pari all'`id` del robot che rappresenta;
  - `id_robot`: array che raccoglie gli id di tutti i nodi variabile in cui è stato suddiviso il robot;
  - `id_zona`: id della zona di riferimento.

[4]Negli esempi che seguiranno sono state utilizzate solo tre tipi di stringe: `P` per robot dotato di piattaforma, `1B` per robot con un braccio e `2B` per robot con due bracci.

3.4.2 Rappresentazione dei task

Istanza ordinaria - Task.java

Nel caso base, un task è rappresentato attraverso un oggetto omonimo, dotato dei campi riportati, ognuno con possibilità di accesso e modifica:

- `id`: id del task;
- `id_zona`: id della zona nella quale si trova il task;
- `richieste`: tipo di dotazioni che esso richiede ad un robot al fine di essere eseguito, espresse come oggetto `String`;
- `id_robot`: array che riunisce gli `id` dei robot che possono eseguire il task perché sono in grado di vederlo e dispongono delle dotazioni richieste. Questo array è aggiornato dal metodo `generateAssignment()`;
- `punti`: intero che indica il punteggio massimo messo a disposizione dal task.

Istanza binaria

- Points_Node.java: rappresenta il nodo punteggio, ovvero quel nodo funzione connesso solo al robot con medesimo id. É quindi una funzione ad un'unica variabile, è dotato dei campi:

  - `id`: id del nodo punteggio che, secondo convenzione, è il medesimo del nodo variabile cui è collegato;
  - `r`: identifica l'oggetto `Robot_BinaryV` cui è connesso il nodo punteggio;
  - `punteggio`: valore intero che rappresenta il punteggio che il robot otterrebbe eseguendo il task;
  - `id_zona`: id della zona di riferimento;

- Collaboration_Node.java: rappresenta il nodo collaborazione, è dotato di:

  - `id`: id del nodo che, secondo convenzione, è un intero di tre cifre tra loro uguali e pari all'`id` che il task aveva nella versione ordinaria dell'algoritmo.
  - `id_robot`: array che riunisce gli `id` dei robot ai quali è collegato il nodo collaborazione. Secondo convenzione, tutti questi nodi avranno come seconda cifra quella dell'`id` che il task aveva nella versione ordinaria dell'algoritmo;
  - `id_zona`: id della zona di riferimento;

3.4.3 Rappresentazione della challenge

Formato dati in ingresso per Challenge.java **e** Challenge_BinaryV.java

In Tabella 3.7 è possibile visualizzare la formattazione richiesta al file di input per la creazione dell'istanza. Il formato è lo stesso per entrambe le versioni dell'algoritmo poiché, come si vedrà a breve, `Challenge_BinaryV` e `Challenge_Collaboration` agiscono su un'istanza di tipo `Challenge` e la specializzano implementando in modo differente la sezione di creazione del file ingresso per il solver.

In dettaglio, nell'input vanno specificate:

- **Zone**: l'id della zona di riferimento

- **Task**: il numero totale di task e, per ognuno di essi i relativi dettagli: id, id della zona in cui si trova, dotazioni richieste ad un robot per eseguirlo e punteggio massimo
- **Robot**: il numero totale di robot e, per ognuno di essi, i relativi dettagli ovvero id, id della zona, dotazioni di cui il robot è fornito e tante coppie (task, distanza dal task) quanti sono i task che si trovano nel raggio del robot

Oggetto	**Formato**
Zona	`ZONE id_zona`
Task	`TASKS num_task`
	`T id_task id_zona dotazione_richiesta punteggio_massimo [C p_aggiuntivo]`[5]
	...
Robot	`ROBOTS num_robot`
	`R id_robot id_zona dotazione_robot (task_m , dist_m)` ⋯ `[L p_sottratto]`[6]
	...

Tabella 3.7: Formattazione dei dati per la creazione di un'istanza di problema Challenge

Istanza ordinaria - Challenge.java

La creazione dell'istanza sulla quale applicare *max-sum* avviene attraverso un oggetto `Challenge` sul quale vengono applicati, nell'ordine, i metodi seguenti:

- `getInstanceFromFile(`*filepath*`)`: a partire da un file di testo formattato secondo le specifiche appena esposte, aggiorna l'oggetto di tipo `Challenge` sul quale viene invocato modificandone i campi `zone, tasks, robots` inserendovi tutti i dati letti in ingresso così come appaiono cioè senza procedere a filtrare le assegnazioni robot-task secondo le dotazioni;

- `generateAssignment()`: questo metodo procede in due step. Per prima cosa filtra le assegnazioni robot-task in modo che i robot siano assegnati solo a quei task di cui soddisfano le richieste e poi, ripartisce il punteggio che andrà a definire l'utilità del task nello scegliere un robot piuttosto che un altro. Dato il punteggio massimo del task, esso viene attribuito così com'è al robot che si trova più vicino al task, viene quindi diminuito in modo inversamente proporzionale alla distanza (Esempio 3.3);

- `createInstanceForFile()`: poichè ora l'oggetto `Challenge` è stato adeguatamente manipolato per soddisfare tutte le specifiche del sistema, attraverso l'analisi dei suoi campi è possibile restituire in output i dati così come vengono processati dal risolutore, ovvero sotto forma di istanza di `MS_COP_Instance`.

[5] Parametro opzionale, considerato solo nel caso di collaborazione tra i robot permessa.
[6] Parametro opzionale, considerato solo nel caso sia permesse ad un robot l'esecuzione di più task.

Esempio 3.3. Si consideri la seguente istanza in ingresso, il punteggio viene ripartito tra i robot che possono soddisfare i rispettivi task.

Istanza
ZONE 1
TASKS 5
T 1 P 100
T 2 P 80
T 3 2B 150
T 4 2B 200
T 5 1B 90
ROBOTS 5
R 1 P (1,3) (2,4)
R 2 2B (2,5) (4,2) (3,8)
R 3 P (3,6) (1,5)
R 4 2B (3,5) (4,1)
R 5 1B (1,3) (5,6)

In seguito all'intersezione tra dotazioni dei robot e richieste dei task, le possibili coppie sono:

- $task_1$ (100 punti): R_1 ed R_3
- $task_2$ (80 punti): R_1
- $task_3$ (150 punti): R_2 ed R_4
- $task_4$ (200 punti): R_2 ed R_4
- $task_5$ (90 punti): R_5

La ripartizione del punteggio per i task con più di un robot avviene così:

- $task_1$ (100 punti): (R_1, 100 punti) ed (R_3, 60 punti)
- $task_3$ (150 punti): (R_2, 94 punti) ed (R_4, 150 punti)
- $task_4$ (200 punti): (R_2, 100 punti) ed (R_4, 200 punti)

dove è stata utilizzata la formula:

$$punteggio = \left\lceil \frac{punteggio\_massimo \times distanza\_minima}{distanza\_attuale} \right\rceil$$

Istanza binaria - Challenge_BinaryV.java

Questa classe estende la precedente specificandone il comportamento per istanza binaria. L'oggetto `Challenge_BinaryV` sul quale vengono invocati i metodi sottostanti possiede, oltre ai campi `zone, tasks, robots` di `Challenge`, anche i seguenti:

- `nodi_collab`: array di `Collaboration_Node`;
- `nodi_punteggio`: array di `Points_Node`;
- `nodi_selettore`: array di `Selector_Node`;
- `robots_binari`: array di `Robots_BinaryV`;

ognuno di essi viene aggiornato inserendovi i nodi del tipo corrispondente creati dal metodo `getInstance(`*filepath*`)`.

I metodi, da invocare nell'ordine in cui sono riportati, agiscono come descritto:

- `getInstance(`*filepath*`)`: dopo aver invocato `getInstanceFromFile(`*filepath*`)` della superclasse, agisce sui campi `tasks` e `robots` per operare la trasformazione descritta in Sezione 3.2.2 e creare, quindi, le rispettive versioni binarie di ogni entità.
- `createInstance()`: attraverso l'ispezione dei campi appena aggiornati, ovvero `nodi_collab, nodi_punteggio, nodi_selettore` e `robots_binari`, crea l'istanza adatta ad essere trasmessa al risolutore.

Per un vincolo imposto dal pacchetto originario `jmaxsum`, il quale richiede che l'istanza in ingresso riporti funzioni il cui id sia solo un intero, è stata usata la seguente convenzione (rappresentata in Figura 3.5):

- id ad una cifra: indica i selettori
- id a due cifre: indica i nodi punteggio
- id a tre cifre: indica i nodi collaborazione

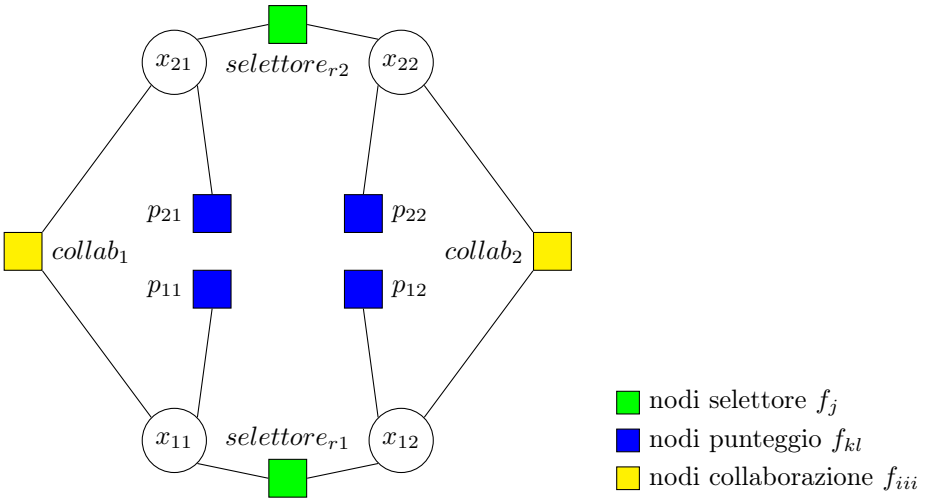


Figura 3.5: Tipologie di nodi funzione differenti per `Challenge_BinaryV`.

Istanza binaria - Challenge_Collaboration.java

Questa classe è specifica per quelle istanze di problema che ammettono che tra i robot vi sia collaborazione e/o che i robot stessi indichino una penalizzazione per il punteggio nel caso che più di un task assegni il proprio compito allo stesso robot. I dettagli sull'implementazione di questa versione sono esposti in 3.2.3.
I concetti di collaborazione e *load* sono indipendenti tra loro e possono essere utilizzati sia singolarmente che combinati. Il loro utilizzo singolo comporta delle conseguenze e implica delle assunzioni,indicate in seguito nell'opzione specifica.

I campi dell'oggetto `Challenge_Collaboration` sono i medesimi di quelli di `Challenge_BinaryV` e i metodi gli stessi di `Challenge`.
Il formato richiesto al file di input è simile a quello esposto per le altre classi, con la differenza che vanno specificate le estensioni di interesse:

- `L punteggio`: questa specifica va aggiunta in corrispondenza della riga con le specifiche del robot cui si riferisce. `L` indica il *load* o carico insieme al decremento di punteggio ad esso assegnato. Il `punteggio` indica l'unità di decremento del punteggio per ogni assegnazione di compito aggiuntiva rispetto alla prima.
  Una istanza che imponga *load* ma non indichi collaborazione sarà risolta in modo diverso a seconda del rapporto tra numero di task e robot. In dettaglio:

  - $n°robot = n°task$: l'esecuzione ha lo stesso esito che avrebbe se fosse un'istanza di `Challenge_BinaryV` poiché, non essendo permessa la collaborazione e non essendoci task in sovrannumero, non verrà utilizzato il concetto di *load*;
  - $n°robot > n°task$: in questo caso non verrà utilizzato il concetto di *load* e l'algoritmo non restituirà un risultato realizzabile poiché, essendoci più task che robot e non essendo permesso che i robot collaborino, quelli in sovrannumero non avranno la possibilità di eseguire alcun compito;
  - $n°robot < n°task$: in questa circostanza si potrà avere l'effettivo utilizzo del carico poiché uno o più robot potrebbero trovarsi ad eseguire più di un task al fine di soddisfare tutti i compiti assegnati.

- `C punteggio`: questa specifica va aggiunta in corrispondenza della riga del task cui si riferisce e fornisce un bonus di punteggio nel caso di esecuzione combinata da parte di più robot.
  La lettera `C` sta ad indicare *collaboration* e la voce `punteggio` è l'unità di incremento del punteggio che va aggiunta ad ogni assegnazione di robot al task successiva alla prima.
  Una istanza che ammette collaborazione ma non *load*, avrà comportamento diverso a seconda del rapporto tra numero di task e robot:

  - $n°robot = n°task$: come per il concetto di carico, l'esecuzione avrà lo stesso esito che avrebbe se fosse un'istanza di `Challenge_BinaryV` poiché, non essendoci robot che accettano di eseguire più di un task, non sarà utilizzata la possibilità di collaborazione;

- *n°robot* > *n°task*: la condizione che ci siano robot in sovrannumero da la possibilità di utilizzare la collaborazione al fine di occupare tutti i robot disponibili;
- *n°robot* < *n°task*: in questo caso, oltre a non avvalersi del concetto di collaborazione, l'istanza non permetterà di giungere a risultato realizzabile poichè non tutti i task sarebbero soddisfacibili.

Alcuni esempi delle differenze tra i casi appena esposti sono visibili nelle Sezioni 3.5.4 e 3.5.3.

Ognuna delle versioni che rappresentano la Challenge, implementa un controllo in seguito all'invocazione del metodo `generateAssignment()`: esso prevede di interpellare l'utente nel caso ci siano task o robot isolati, cioè, task non connessi ad alcun robot (e quindi non eseguibili) e robot non connessi ad alcun task (e quindi non impiegabili per l'esecuzione di nessun compito) al fine di procedere escludendoli dal calcolo successivo. Un esempio di questa interazione è mostrato in Figura 3.7 e 3.6.

```
Il task t_5 non è connesso ad nessun robot.
Vuoi procedere senza considerarlo? Sì [Y] o No [N]
Default: Y

Procedo ignorando il task t_5
```

Figura 3.6: Output nel caso di task isolato rilevato.

```
Il robot r_4 non è connesso ad alcun task.
Vuoi procedere senza considerarlo? Sì [Y] o No [N]
Default: Y

Procedo ignorando il robot r_4.
```

Figura 3.7: Output nel caso di robot isolato rilevato.

3.4.4 Interpretazione del risultato

All'interno del package *olimpo* in cui, tra gli altri, sono presenti anche i solver che si occupano dell'applicazione di *max-sum* sull'istanza che ricevono in ingresso, sono stati inseriti due solver che si distinguono dagli altri solo per la modalità con la quale restituiscono il risultato dell'algoritmo.

Istanza ordinaria - Aphrodite.java

Grazie alla modalità con la quale è stata costruita l'istanza, il metodo `solve()` restituisce, oltre al punteggio massimo che il sistema può raggiungere, anche la

configurazione di assegnazioni robot-task che permette questo risultato andando ad esaminare i valori che assumono le variabili: tale valore corrisponde infatti all'id del task presso il quale è stato allocato il robot.

Istanza binaria e con collaborazione/*load* - Era.java

Analogamente, il risultato restituito da questo solver permette di conoscere il punteggio massimo e la configurazione di assegnazioni robot-task che lo permette. In questo, considerato il formato adottato per i dati in ingresso, per conoscere a quale task sia stato assegnato ogni robot è sufficiente esaminare gli id delle variabili il cui valore è posto ad 1, da essi si può ricavare l'id del robot come prima cifra e l'id del task cui è stato assegnato come seconda cifra.

Entrambi i solver suddividono l'output per zone, qualora ve ne sia più d'una nella challenge e restituiscono l'assegnazione solo se essa è effettivamente realizzabile. In caso contrario elencano il punteggio che non è possibile superare (negativo) ed i vincoli, ovvero le funzioni, non soddisfatte. Un esempio di quanto appena detto è mostrato in Figura 3.8.

La scelta circa l'istanza corretta da creare ed il solver da invocare, dipendono dalle specifiche del problema ed eventualmente il formato di input del quale si dispone, in Tabella 3.8 è possibile vedere le principali differenze.

```
L'esecuzione dell'algoritmo non ha restituito la configurazione cercata.
Alcuni vincoli non sono soddisfacibili.
------------------------------------------------------------------------
Vincoli non soddisfatti: f_4, f_1,
Il punteggio finale non supera : -1836.0punti.
```

```
L'esecuzione dell'algoritmo non ha restituito la configurazione cercata.
Alcuni vincoli non sono soddisfacibili.
-----------------------------------------------------
Vincoli non soddisfatti: f_2, f_222, f_1, f_333, f_111,
Il punteggio finale non supera : -5000.0punti.
```

Figura 3.8: Esempi di output nel caso istanza di problema senza soluzione.

`Challenge`	`Challenge_BinaryV`	`Challenge_Collaboration`
No collaborazione/*load*	No collaborazione/*load*	Si Collaborazione e/o *load*
Input standard	Input binario	Input-binario
max-sum ordinario	max-sum binario	max-sum binario
`Aphrodite`	`Era`	`Era`

Tabella 3.8: Riepilogo delle differenze tra gli oggetti che rappresentano la *challenge*.

3.5 Esempi

Gli esempi di base di applicazione dell'algoritmo prevedono una situazione in cui il numero di robot ed il numero di task da eseguire siano uguali, successivamente questo vincolo non è applicato.
I robot rappresentano le variabili ed i task le funzioni, l'esecuzione dell'algoritmo produce la configurazione di assegnazioni (robot, task) che permette il raggiungimento del punteggio massimo date le condizioni iniziali.

3.5.1 Esempio di base

Si consideri il seguente scenario con tre robot e tre task.
I task $task_1, task_2, task_3$ siano rispettivamente rappresentati nelle funzioni f_1, f_2, f_3 ed i robot R_1, R_2, R_3 dalle variabili x_1, x_2, x_3.

Si supponga che una visione dall'alto dell'arena sia quella di Figura 3.9.
Ogni robot ha dotazione diversa, nel dettaglio:

- $\mathbf{R_1}$: robot con un braccio
- $\mathbf{R_2}$: robot con due bracci privo di piattaforma
- $\mathbf{R_3}$: robot con sola piattaforma omnidirezionale

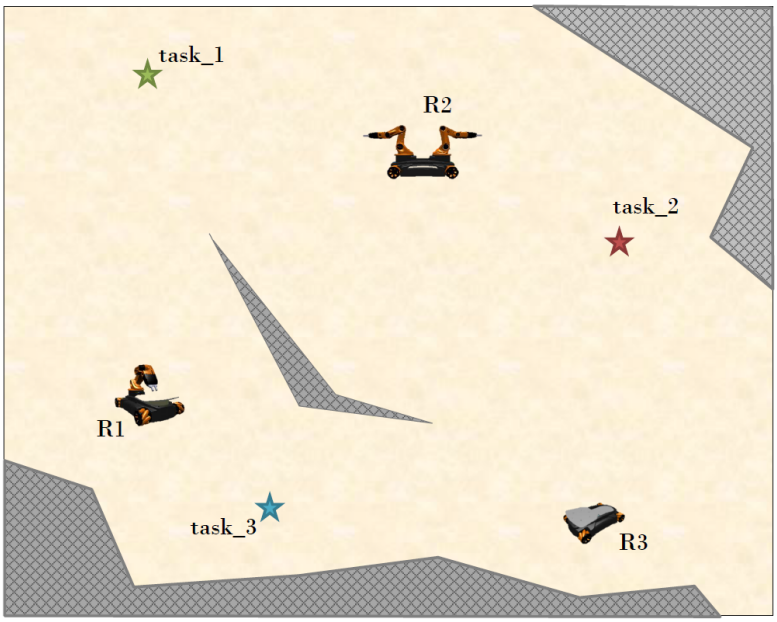


Figura 3.9: Visione dall'alto dell'arena per Esempio 3.5.1

Le specifiche dei task sono le seguenti:

- $\mathbf{task_1}$: richiede un robot con almeno un braccio perché consiste nella manipolazione di un oggetto

- **task$_2$**: richiede un robot dotato di piattaforma o braccio perché consiste nel trasportare un oggetto
- **task$_3$**: il suo punteggio varia in base al robot che lo esegue poiché ha più opzioni. Richiede di spostare un oggetto che viene posto su piattaforma, se il robot è dotato anche di braccio allora può provvedere autonomamente a movimentare l'oggetto al termine del trasporto.

In Figura 3.10 e Tabella 3.9 sono rappresentate le connessioni task-robot realizzabili.

Task	Descrizione	R_1	R_2	R_3
$Task_1$	Manipolazione oggetto	√	√	
$Task_2$	Trasporto oggetto	√	√	√
$Task_3$	Trasporto su piattaforma (e manipolazione)	√		√

Tabella 3.9: Possibili assegnazioni task-robot.

Il dominio delle variabili, che può essere ricavato leggendo gli id dei task in corrispondenza dei quali c'è il segno di spunta nella colonna del robot, è il seguente:

- $D_1 = \{1, 2, 3\}$
- $D_2 = \{1, 2\}$
- $D_3 = \{2, 3\}$

e i valori delle configurazioni di variabili e punteggio sono:

x_1	x_2	punteggio f_1
1	1	$-\infty$
1	2	10
2	1	2
2	2	$-\infty$
3	1	12
3	2	$-\infty$

x_1	x_3	punteggio f_3
1	2	$-\infty$
1	3	2
2	2	$-\infty$
2	3	1
3	2	9
3	3	$-\infty$

x_1	x_2	x_3	punteggio f_2
1	1	2	$-\infty$
1	1	3	$-\infty$
1	2	2	$-\infty$
1	2	3	11
2	1	2	$-\infty$
2	1	3	16
2	2	2	$-\infty$
2	2	3	$-\infty$
3	1	2	25
3	1	3	$-\infty$
3	2	2	$-\infty$
3	2	3	$-\infty$

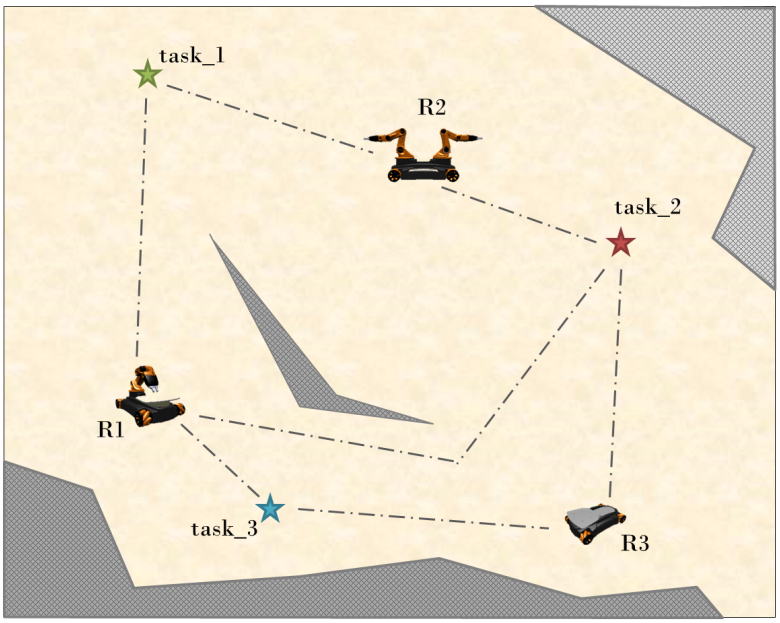


Figura 3.10: Possibili coppie robot-task per Esempio 3.5.1

L'output prodotto dall'esecuzione del programma in Figura 3.11 riporta il punteggio totale di 51 punti così ottenuto:

- $\mathbf{task_1}$ eseguito da $\mathbf{R_2}$ per un punteggio parziale di 15 punti
- $\mathbf{task_2}$ eseguito da $\mathbf{R_3}$ per un punteggio parziale di 25 punti
- $\mathbf{task_3}$ eseguito da $\mathbf{R_1}$ per un punteggio parziale di 11 punti

Ovvero, sulla mappa i robot seguono le traiettorie rappresentate in Figura 3.12.

```
Esito dell'applicazione dell'algoritmo
---------------------------------------
Punteggio massimo ottenibile:51.0
La configurazione di robot finale è la seguente:
Robot R1 esegue task t_3
Robot R2 esegue task t_1
Robot R3 esegue task t_2
```

Figura 3.11: Output del programma per Esempio 3.5.1

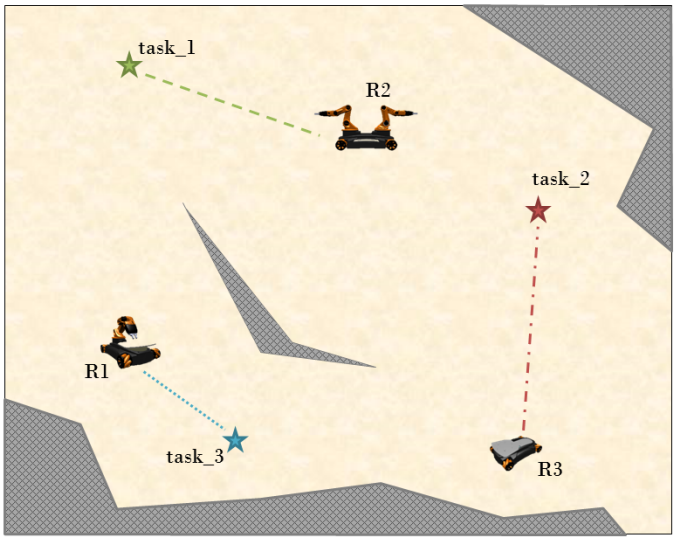


Figura 3.12: Risultato prodotto dall'algoritmo per Esempio 3.5.1

3.5.2 Esempio binario

Si consideri l'istanza di problema indicata in tabella. Su di essa l'esito dell'applicazione di max-sum con il risolutore `Athena`[7] restituisce il risultato indicato in seguito.

L'istanza rappresenta un sistema con due task e due robot in cui ogni robot è connesso ad entrambi i task.
Ogni variabile ha id corrispondente alla coppia `(robot, task)`. Le funzioni hanno id che rispetta le convenzioni precedentemente esposte.

Istanza
AGENT 1
VARIABLE 12 1 2 (0,1)
VARIABLE 11 1 2 (0,1)
VARIABLE 22 1 2 (0,1)
VARIABLE 21 1 2 (0,1)
CONSTRAINT 1 1 12 11
F 0 0 -inf
F 0 1 0
F 1 0 0
F 1 1 -inf

[7] `Athena` è il risolutore di base, non specifico per istanze del pacchetto `challenge`. Viene qui utilizzato solo a titolo di esempio.

```
CONSTRAINT 2 1 22 21
F 0 0 -inf
F 0 1 0
F 1 0 0
F 1 1 -inf
CONSTRAINT 12 1 12
F 0 0
F 1 100
CONSTRAINT 11 1 11
F 0 0
F 1 150
CONSTRAINT 22 1 22
F 0 0
F 1 250
CONSTRAINT 21 1 21
F 0 0
F 1 200
CONSTRAINT 222 1 22 12
F 0 0 -inf
F 0 1 0
F 1 0 0
F 1 1 -inf
CONSTRAINT 111 1 21 11
F 0 0 -inf
F 0 1 0
F 1 0 0
F 1 1 -inf
```

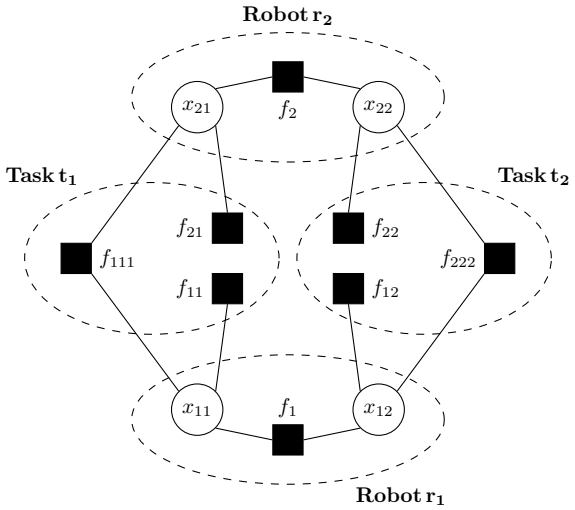


Figura 3.13: Istanza per max-sum binario con elementi evidenziati

L'esito riportato è $x_{11} = x_{22} = 1, x_{12} = x_{21} = 0$, ovvero il robot r_2 è assegnato al task t_2 ed il robot r_1 al task t_1. Il punteggio raggiunto è di 400 punti.

3.5.3 Esempio con solo *load*

In questa sezione, ed analogamente nella successiva, viene presentato il caso in cui un oggetto di tipo `Challenge_Collaboration` ammetta l'utilizzo di solo uno dei concetti tra carico e collaborazione e si evidenziano gli esiti diversi dati dall'applicazione dell'algoritmo max-sum a seconda del rapporto tra numero di task e robot.

Numero di task e robot uguali

Si consideri l'istanza di Tabella 3.11, l'esito ottenuto dall'applicazione dell'algoritmo (Figura 3.14) prevede:

- **Punteggio massimo**: 450 punti
- **Assegnazioni**: R_2 esegue t_1, R_1 esegue t_3 e R_3 esegue t_2

Input File
ZONE 1
TASKS 3
T 1 1 P 100
T 2 1 P 200
T 3 1 P 150
ROBOTS 3
R 1 1 P (1,3) (2,4) (3,5) L 20
R 2 1 P (2,5) (1,2) (3,8) L 10
R 3 1 P (2,3) L 5

Tabella 3.11: Istanza per Esempio con solo *load*.

```
Esito applicazione algoritmo
-------------------------------------
Punteggio massimo ottenibile:450.0
La configurazione di robot finale è la seguente:

Zona 1
Robot_2 assegnato al Task_1
Robot_1 assegnato al Task_3
Robot_3 assegnato al Task_2

Dettagli punteggi parziali (non nulli):
Punteggio parziale per f_21: 100.0punti.
Punteggio parziale per f_32: 200.0punti.
Punteggio parziale per f_13: 150.0punti.
```

Figura 3.14: Risultato prodotto dall'algoritmo per l'istanza di Tabella 3.11

Robot in sovrannumero rispetto ai task

Modificando l'istanza precedente al fine di aggiungere due robot, come mostrato in Tabella 3.12, non viene restituito il risultato cercato (Figura 3.18) e l'output ne indica la causa.

Input File
ZONE 1
TASKS 3
T 1 1 P 100
T 2 1 P 200
T 3 1 P 150
ROBOTS 5
R 1 1 P (1,3) (2,4) (3,5) L 20
R 2 1 P (2,5) (1,2) (3,8) L 10
R 3 1 P (2,3) L 5
R 4 1 P (3,5) (1,2) L 4
R 5 1 P (2,6) L 12

Tabella 3.12: Istanza per Esempio con solo *load* e robot in sovrannumero.

```
WARNING: Il numero di robot è MAGGIORE rispetto a quello dei task.
L'algoritmo potrebbe non restituire una assegnazione valida
nel caso di collaborazione non permessa nei task interessati.
```

```
L'esecuzione dell'algoritmo non ha restituito la configurazione cercata.
Alcuni vincoli non sono soddisfacibili.
```

Figura 3.15: Risultato prodotto dall'algoritmo per l'istanza di Tabella 3.12

Task in sovrannumero rispetto ai robot

Se, invece, la modifica dell'istanza iniziale è fatta aggiungendo i task t_4 e t_5 (Tabella 3.13), l'algoritmo restituisce un'assegnazione valida:

- **Punteggio massimo**: 685 punti
- **Assegnazioni**: R_2 esegue t_1, R_1 esegue t_3 e R_3 esegue t_2, t_5 e t_4

Input File
ZONE 1
TASKS 5
T 1 1 P 100
T 2 1 P 200
T 3 1 P 150
T 4 1 P 175
T 5 1 P 75
ROBOTS 3
R 1 1 P (1,3) (2,4) (3,5) (5,3) L 20
R 2 1 P (2,5) (1,2) (3,8) (4,7) L 10
R 3 1 P (2,3) (4,2) (5,2) L 5

Tabella 3.13: Istanza per Esempio con solo *load* e task in sovrannumero.

In questo caso è stato effettivamente utilizzato il *load* per il robot R_3 (Figura 3.16).

```
WARNING: Il numero di robot è MINORE rispetto ai task,
L'algoritmo potrebbe non restituire una assegnazione valida
nel caso i robot interessati non permettano l'esecuzione di più di un task.
```

```
Esito applicazione algoritmo
-------------------------------------
Punteggio massimo ottenibile:685.0
La configurazione di robot finale è la seguente:

Zona 1
Robot_2 assegnato al Task_1
Robot_1 assegnato al Task_3
Robot_3 assegnato al Task_2
Robot_3 assegnato al Task_5
Robot_3 assegnato al Task_4

Dettagli punteggi parziali (non nulli):
Punteggio parziale per f_35: 75.0punti.
Punteggio parziale per f_34: 175.0punti.
Punteggio parziale per f_21: 100.0punti.
Punteggio parziale per f_32: 200.0punti.
Punteggio parziale per f_3: -15.0punti.
Punteggio parziale per f_13: 150.0punti.
```

Figura 3.16: Risultato prodotto dall'algoritmo per l'istanza di Tabella 3.13.

3.5.4 Esempio con sola collaborazione

Questo esempio mostra l'esito prodotto dall'esecuzione di max-sum nel caso in cui in ingresso sia fornita una istanza di problema che permetta la collaborazione per uno o più task, al variare del rapporto tra numero di task e robot.

Numero di task e robot uguali

Si consideri l'istanza di Tabella 3.14, l'esito ottenuto dall'applicazione dell'algoritmo (Figura 3.17) prevede:

- **Punteggio massimo**: 276 punti
- **Assegnazioni**: R_2 esegue t_1, R_1 esegue t_2 e R_3 esegue t_3

Esso è il medesimo che si otterrebbe se l'oggetto fosse di tipo `Challenge_BinaryV` e non avesse specifiche sulla collaborazione.

Input File
ZONE 1
TASKS 3
T 1 1 P 100 C 20
T 2 1 P 75 C 10
T 3 1 P 120 C 15
ROBOTS 3
R 1 1 P (1,3) (2,4)
R 2 1 P (2,5) (1,2) (3,8)
R 3 1 P (3,6) (2,3)

Tabella 3.14: Istanza per Esempio con solo collaborazione.

```
Esito applicazione algoritmo
-------------------------------------
Punteggio massimo ottenibile:276.0
La configurazione di robot finale è la seguente:

Zona 1
Robot_2 assegnato al Task_1
Robot_1 assegnato al Task_2
Robot_3 assegnato al Task_3

Dettagli punteggi parziali (non nulli):
Punteggio parziale per f_21: 100.0punti.
Punteggio parziale per f_33: 120.0punti.
Punteggio parziale per f_12: 56.0punti.
```

Figura 3.17: Risultato prodotto dall'algoritmo per l'istanza di Tabella 3.14

Robot in sovrannumero rispetto ai task

Aggiungendo all'istanza precedente i robot R_4 ed R_5 (Tabella 3.15), l'esito prodotto vedrà l'utilizzo del concetto di collaborazione per i task t_1 e t_3:

Input File
ZONE 1
TASKS 3
T 1 1 P 100 C 20
T 2 1 P 75 C 10
T 3 1 P 120 C 15
ROBOTS 5
R 1 1 P (1,3) (2,4)
R 2 1 P (2,5) (1,2) (3,8)
R 3 1 P (3,6) (2,3)
R 4 1 P (1,5) (3,4)
R 5 1 P (1,7) (2,3) (3,3)

Tabella 3.15: Istanza per Esempio con sola collaborazione e robot in sovrannumero.

- **Punteggio massimo**: 487 punti
- **Assegnazioni**: R_1 esegue t_1, R_2 esegue t_1, R_3 esegue t_2, R_4 esegue t_3 e R_5 esegue t_3

```
WARNING: Il numero di robot è MAGGIORE rispetto a quello dei task.
L'algoritmo potrebbe non restituire una assegnazione valida
nel caso di collaborazione non permessa nei task interessati.
```

```
Esito applicazione algoritmo
------------------------------------
Punteggio massimo ottenibile:487.0
La configurazione di robot finale è la seguente:

Zona 1
Robot_5 assegnato al Task_3
Robot_1 assegnato al Task_1
Robot_2 assegnato al Task_1
Robot_3 assegnato al Task_2
Robot_4 assegnato al Task_3

Dettagli punteggi parziali (non nulli):
Punteggio parziale per f_333: 15.0punti.
Punteggio parziale per f_53: 120.0punti.
Punteggio parziale per f_21: 100.0punti.
Punteggio parziale per f_32: 75.0punti.
Punteggio parziale per f_43: 90.0punti.
Punteggio parziale per f_111: 20.0punti.
Punteggio parziale per f_11: 67.0punti.
```

Figura 3.18: Risultato prodotto dall'algoritmo per l'istanza di Tabella 3.15

Task in sovrannumero rispetto ai robot

Se, all'istanza di partenza, si aggiunge il task t_3 (Tabella 3.16),

Input File
ZONE 1
TASKS 5
T 1 1 P 100 C 20
T 2 1 P 75 C 10
T 3 1 P 120 C 15
T 4 1 P 115 C 12
ROBOTS 3
R 1 1 P (1,3) (2,4) (4,5)
R 2 1 P (2,5) (1,2) (3,8) (5,2)
R 3 1 P (3,6) (2,3) (5,6) (4,3)

Tabella 3.16: Istanza per Esempio con sola collaborazione e task in sovrannumero.

non si ottiene un risultato realizzabile (Figura 3.19), uno dei task non può essere eseguito.

```
WARNING: Il numero di robot è MINORE rispetto ai task,
L'algoritmo potrebbe non restituire una assegnazione valida
nel caso i robot interessati non permettano l'esecuzione di più di un task.
```

```
L'esecuzione dell'algoritmo non ha restituito la configurazione cercata.
Alcuni vincoli non sono soddisfacibili.
```

Figura 3.19: Risultato prodotto dall'algoritmo per l'istanza di Tabella 3.16.

3.5.5 Esempio finale

Questo esempio racchiude l'utilizzo dei concetti di collaborazione, *load* e challenge multi-zona.

Si supponga di considerare un'arena per la challenge suddivisa in due zone (`ZONE 1` e `ZONE 2`), di avere un numero totale di task pari a cinque, di cui tre nella prima zona e due nella seconda, e di avere un insieme di sei robot con la medesima dotazione distribuiti equamente tra le zone.
Tutti i task richiedono la stessa dotazione al/ai robot, tre di essi permettono la collaborazione e forniscono un punteggio bonus:

- $\mathbf{task_1}$ ($zona_1$) fornisce 100 punti massimi, permette la collaborazione con un bonus di 10 punti. Richiede di eseguire l'assemblaggio di un oggetto dati i suoi componenti.
- $\mathbf{task_2}$ ($zona_1$) fornisce 80 punti massimi, non permette la collaborazione. Consiste nell'identificare la forma di un oggetto ed inserirlo in apposita scatola.
- $\mathbf{task_3}$ ($zona_1$) fornisce 150 punti massimi, permette la collaborazione con un bonus di 15 punti. Consiste nel preparare il piano di lavoro disponendo un insieme di oggetti, già presenti sul posto, secondo l'ordine indicato dal task.
- $\mathbf{task_4}$ ($zona_2$) fornisce 75 punti massimi, non permette la collaborazione. Richiede di raccogliere un oggetto dal nastro trasportatore in movimento e posizionarlo sulla piattaforma.
- $\mathbf{task_5}$ ($zona_2$) fornisce 100 punti massimi, permette la collaborazione con un bonus di 8 punti. Consiste nella ricerca e prelievo di un insieme di oggetti specifici disposti in modo casuale all'interno di una scatola.

Definita una generica unità di distanza **u**, la situazione dei robot sia la seguente:

- $\mathbf{robot_1}$ ($zona_1$) vede t_1 a $3\mathbf{u}$ e t_2 a $4\mathbf{u}$, accetta di eseguire più di un task con *load* di 15 punti,
- $\mathbf{robot_2}$ ($zona_1$) vede t_2 a $5\mathbf{u}$, t_1 a $2\mathbf{u}$ e t_3 a $8\mathbf{u}$,
- $\mathbf{robot_3}$ ($zona_1$) vede t_3 a $6\mathbf{u}$ e t_2 a $3\mathbf{u}$, accetta di eseguire più di un task con *load* di 10 punti;
- $\mathbf{robot_4}$ ($zona_2$) vede t_4 a $3\mathbf{u}$ e t_5 a $6\mathbf{u}$;
- $\mathbf{robot_5}$ ($zona_2$) vede t_5 a $2\mathbf{u}$ e t_4 a $1\mathbf{u}$;
- $\mathbf{robot_6}$ ($zona_2$) vede t_5 a $3\mathbf{u}$;

Come negli altri esempi, si cerca l'assegnazione robot-task che, rispettando tutte le condizioni, massimizza il punteggio che può essere ottenuto.

Lo scenario appena descritto si traduce, rispettando le specifiche per la costruzione del file in ingresso, in quello visualizzato in Tabella 3.17.
Il file in ingresso è letto dal metodo `getInstanceFromFile(`*`filepath`*`)` della classe `Challenge_Collaboration`, su di esso l'invocazione di `createInstanceForFile()` produce l'istanza di tipo `MS_COP_Instance` rappresentata in Tabella 3.18. Su questa istanza viene invocato il solver `Era` che restituisce il risultato in Figura 3.20.

La rappresentazione dello scenario su factor graph ordinario e binario è riportata nelle Figure 3.21, 3.22 e 3.23.

Input File
ZONE 1
ZONE 2
TASKS 5
T 1 1 1B 100 C 10
T 2 1 1B 80
T 3 1 1B 150 C 15
T 4 2 1B 75
T 5 2 1B 100 C 8
ROBOTS 6
R 1 1 1B (1,3) (2,4) L 15
R 2 1 1B (2,5) (1,2) (3,8)
R 3 1 1B (3,6) (2,3) L 10
R 4 2 1B (4,3) (5,6)
R 5 2 1B (5,2) (4,1)
R 6 2 1B (5,3)

Tabella 3.17: File di ingresso per lo scenario di Esempio 3.5.5.

```
Esito applicazione algoritmo
-------------------------------------
Punteggio massimo ottenibile:605.0
La configurazione di robot finale è la seguente:

Zona 2
Robot_5 assegnato al Task_5
Robot_6 assegnato al Task_5
Robot_4 assegnato al Task_4

Zona 1
Robot_2 assegnato al Task_3
Robot_1 assegnato al Task_1
Robot_3 assegnato al Task_3
Robot_3 assegnato al Task_2
```

Figura 3.20: Risultato prodotto dall'algoritmo per l'istanza di Tabella 3.17

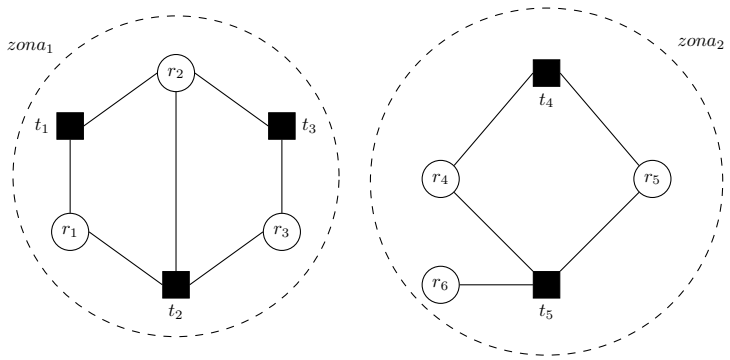


Figura 3.21: Factor graph di zone 1 e 2.

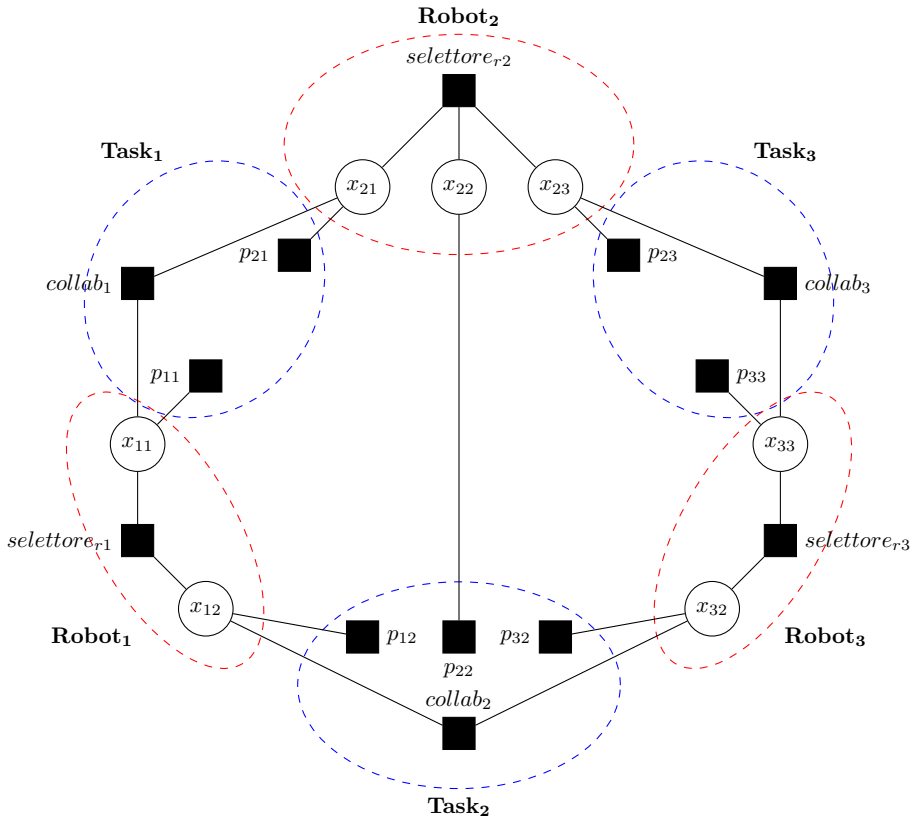


Figura 3.22: Factor graph binario di zona 1.

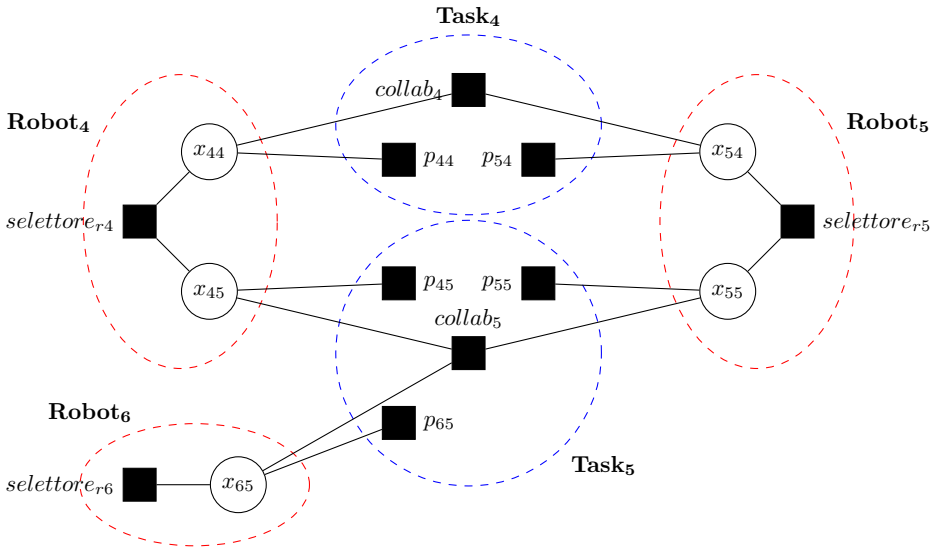


Figura 3.23: Factor graph binario di zona 2.

Istanza
AGENT 1 AGENT 2 VARIABLE 11 1 2 (0,1) VARIABLE 12 1 2 (0,1) VARIABLE 21 1 2 (0,1) VARIABLE 22 1 2 (0,1) VARIABLE 23 1 2 (0,1) VARIABLE 32 1 2 (0,1) VARIABLE 33 1 2 (0,1) VARIABLE 44 2 2 (0,1) VARIABLE 45 2 2 (0,1) VARIABLE 54 2 2 (0,1) VARIABLE 55 2 2 (0,1) VARIABLE 65 2 2 (0,1) CONSTRAINT 1 1 11 12 F 0 0 -1000 F 0 1 0 F 1 0 0 F 1 1 -30 CONSTRAINT 2 1 21 22 23 F 0 0 0 -1000 F 0 0 1 0 F 0 1 0 0 F 0 1 1 -1000 F 1 0 0 0 F 1 0 1 -1000 F 1 1 0 -1000 F 1 1 1 -1000 CONSTRAINT 3 1 32 33 F 0 0 -1000 F 0 1 0 F 1 0 0 F 1 1 -20 CONSTRAINT 4 2 44 45 F 0 0 -1000 F 0 1 0 F 1 0 0 F 1 1 -1000 CONSTRAINT 5 2 54 55 F 0 0 -1000 F 0 1 0 F 1 0 0 F 1 1 -1000 CONSTRAINT 6 2 65 F 0 -1000 F 1 0 CONSTRAINT 11 1 11 F 0 0 F 1 67 CONSTRAINT 12 1 12 F 0 0 F 1 60 CONSTRAINT 21 1 21 F 0 0 F 1 100 CONSTRAINT 22 1 22 F 0 0 F 1 48 CONSTRAINT 23 1 23 F 0 0 F 1 113 CONSTRAINT 32 1 32 F 0 0 F 1 80 CONSTRAINT 33 1 33 F 0 0 F 1 150 CONSTRAINT 44 2 44 F 0 0 F 1 25 CONSTRAINT 45 2 45 F 0 0 F 1 33 CONSTRAINT 54 2 54 F 0 0 F 1 75 CONSTRAINT 55 2 55 F 0 0 F 1 100 CONSTRAINT 65 2 65 F 0 0 F 1 67 CONSTRAINT 111 1 11 21 F 0 0 -1000 F 0 1 0 F 1 0 0 F 1 1 10 CONSTRAINT 222 1 12 22 32 F 0 0 0 -1000 F 0 0 1 0 F 0 1 0 0 F 0 1 1 -1000 F 1 0 0 0 F 1 0 1 -1000 F 1 1 0 -1000 F 1 1 1 -1000 CONSTRAINT 333 1 23 33 F 0 0 -1000 F 0 1 0 F 1 0 0 F 1 1 15 CONSTRAINT 444 2 44 54 F 0 0 -1000 F 0 1 0 F 1 0 0 F 1 1 -1000 CONSTRAINT 555 2 45 55 65 F 0 0 0 -1000 F 0 0 1 0 F 0 1 0 0 F 0 1 1 8 F 1 0 0 0 F 1 0 1 8 F 1 1 0 8 F 1 1 1 16

Tabella 3.18: Istanza di problema max-sum ottenuta dal file in ingresso di Tabella 3.17.

Capitolo 4

Altre applicazioni di max-sum

Questo capitolo riunisce la presentazione di alcuni casi di applicazione di max-sum a problemi reali facenti parte della letteratura più recente.

4.1 Max-sum per il problema del commesso viaggiatore

In questa sezione sarà presentato il risultato esposto in [49] inerente la possibilità di risolvere un problema TSP rappresentando l'istanza di problema su factor graph ed applicando l'algoritmo max-sum.

Il problema del commesso viaggiatore consiste, dato un insieme di città, nel determinare il cammino più breve che visiti ogni città un'unica volta facendo ritorno a quella di partenza.
Il problema è spesso rappresentato su un grafo pesato non diretto in cui ogni nodo rappresenta una città e su di esso si cerca il circuito hamiltoniano di peso minimo.

4.1.1 Formalizzazione su factor graph

Sia data una matrice $[s(i,j)]_{(N+1)\times(N+1)}$ di (N+1) città tale che $s(i,j)$ indichi la *similarity* tra le città i e j ovvero la differenza tra massima distanza Euclidea e la distanza Euclidea tra le due. Lo scopo è quello di trovare un ciclo tra le città che massimizzi la somma di *similarities*.
Poiché la somma è indipendente dalla città in cui si conclude il cammino, si fissi la (N+1)-esima città. Sia $c = [c_1, \cdots, c_N]$ un vettore il cui i-esimo elemento rappresenti l'id della città visitata al passo t, allora il risultato cercato è il vettore delle visite che soddisfa

$$\max_{c_t \in \{1,\cdots,N\}/c_t \neq c_{t'} \forall t \neq t'} \sum_{t=1}^{N-1} s(c_t, c_{t+1} + s(c_N, N+1) + s(N+1, c_1)$$

Al fine di formalizzare la funzione sopra, si introducono $B = [b_{it}]_{N\times N}$, matrice a valori binari che rappresenta l'ordine con cui vengono visitate le prime N città,

ovvero

$$b_{it} = \begin{cases} 1 & \text{la città } i \text{ è visitata al passo } t \\ 0 & \text{altrimenti} \end{cases} \tag{4.1}$$

e il vincolo c_t sia trasformato in un *city constraint* ed uno *step contraint*.

- *city constraint*: la città i deve essere visitata esattamente una volta

$$I_i(b_{i1}, \cdots, b_{iN}) = \begin{cases} 0 & \text{se } \sum_{t=1}^{N} b_{it} = 1 \\ -\infty & \text{altrimenti} \end{cases} \tag{4.2}$$

- *step constraint*: al passo t deve essere vistata esattamente una città

$$E_t(b_{i1}, \cdots, b_{iN}, c_t) = \begin{cases} 0 & \text{se } b_{c_t t} = 1 \text{ e } b_{it} = 0 \; \forall i \neq c_t \\ -\infty & \text{altrimenti} \end{cases} \tag{4.3}$$

La funzione $S_t(c_{t+1}|c_t)$ indica la *similarity* tra c_t e c_{t+1}

$$S_t(c_{t+1}|c_t) = \begin{cases} s_t(c_t, c_{t+1}) & c_t \neq c_{t+1} \\ -\infty & c_t = c_{t+1} \end{cases} \tag{4.4}$$

La funzione da massimizzare si traduce in

$$\begin{aligned} G(B, c) = &\sum_{t=1}^{N+1} S_t(c_{t+1}|c_t) + S_N(N+1|c_N) + S_{N+1}(c_1|N+1) \\ &+ \sum_{i=1}^{N} I_i(b_{i1}, \cdots, b_{iN}) + \sum_{t=1}^{N} E_t(b_{i1}, \cdots, b_{iN}, c_t) \end{aligned} \tag{4.5}$$

Tipologie di messaggi

I messaggi in questo factor graph sono di otto tipi differenti, a seconda dei nodi coinvolti:

- η_{it} da I_i a b_{it}

$$\begin{aligned} \eta_{it}(m) &= \max_{b_{it'}: t \neq t'} \left[I_i(b_{i1}, \cdots, b_{iN}) + \sum_{t': t \neq t'} w_{it'}(b_{it'}) \right] \\ &= \begin{cases} \sum_{t': t \neq t'} w_{it'}(0) & m = 1 \\ \max_{t': t \neq t'} \left[w_{it'}(1) + \sum_{t'': t'' \notin \{t, t'\}} w_{it''}(0) \right] & m = 0 \end{cases} \end{aligned} \tag{4.6}$$

- ω_{it} da b_{it} a I_i

$$\omega_{it}(m) = \phi_{it}(m) \tag{4.7}$$

- ϕ_{it} da E_t a b_{it}

$$\begin{aligned} \phi_{it}(m) &= \max_{b_{i't}: i \neq i', c_t} \left[E_t(b_{it}, \cdots, b_{Nt}, c_t) + \sum_{i': i \neq i'} \gamma_{i't}(b_{i't}) + \zeta_t(c_t) \right] \\ &= \begin{cases} \sum_{i': i \neq i'} \gamma_{i't}(0) + \zeta_t(i) & m = 1 \\ \max_{i': i \neq i'} \left[\gamma_{i't}(1) + \zeta_t(i') + \sum_{i'': i'' \notin \{i, i'\}} \gamma_{i''t}(0) \right] & m = 0 \end{cases} \end{aligned} \tag{4.8}$$

- γ_{it} da b_{it} a E_t

$$\gamma_{it}(m) = \eta_{it}(m) \tag{4.9}$$

- β_t con $t = 1, \cdots, N-1$ da c_t a S_t

$$\beta_t(m) = \begin{cases} \lambda_t(m) + s(N+1, m) & t = 1 \\ \lambda_t(m) + \delta_{t-1}(m) & t = 2, \cdots, N-1 \end{cases} \tag{4.10}$$

- δ_t con $t = 1, \cdots, N-1$ da S_t a c_{t+1}

$$\delta_t(m) = max_{c_t}[S_t(c_t, m) + \beta_t(c_t)] = max_{c_t \neq m}[s(c_t, m) + \beta_t(c_t)] \tag{4.11}$$

- λ_t da E_t a c_t

$$\begin{aligned} \lambda_t(m) =& \max_{b_{it}: i=1,\cdots,N} \left[E_t(b_{1t}, \cdots, b_{Nt}, m) + \sum_{i=1}^{N} \gamma_{it}(b_i t) \right] \\ =& \gamma_{mt}(1) + \sum_{i: i \neq m} \gamma_{it}(0) \end{aligned} \tag{4.12}$$

- ζ_t da c_t a E_t

$$\zeta_m = \begin{cases} s(N+1, m) & t = 1 \\ \delta_{t-1}(m) & t = 2, \cdots, N-1 \\ \delta_{N-1}(m) + s(m, N+1) & t = N \end{cases} \tag{4.13}$$

Su questi messaggi si può operare una semplificazione: per ognuno di quelli associati ad una variabile binaria si introduce la differenza tra variabile attiva e non attiva, mentre i rimanenti possono essere scomposti in somma di un valore costante ed una variabile.

$$\begin{aligned} \tilde{\eta_{it}} =& \eta_{it}(1) - \eta_{it}(0) \\ \tilde{\omega_{it}} =& \omega_{it}(1) - \omega_{it}(0) \\ \tilde{\phi_{it}} =& \phi_{it}(1) - \phi_{it}(0) \\ \tilde{\gamma_{it}} =& \gamma_{it}(1) - \gamma_{it}(0) \\ \beta_t(m) =& \tilde{\beta_t}(m) + \overline{\beta_t}(m) \\ \delta_t(m) =& \tilde{\delta_t}(m) + \overline{\delta_t}(m) \\ \lambda_t(m) =& \tilde{\lambda_t}(m) + \overline{\lambda_t}(m) \\ \zeta_t(m) =& \tilde{\zeta_t}(m) + \overline{\zeta_t}(m) \end{aligned}$$

Concludendo, per stimare il valore di un elemento di c_t, vanno sommati tutti i messaggi in ingresso ed estratto il $\hat{c}_t$ che massimizza la somma:

$$\begin{aligned} \hat{c}_t =& \underset{c_t=1,\cdots,N}{\text{argmax}} [\tilde{\lambda_t}(c_t) + \tilde{\delta_t}(c_t)], \ \forall t = 2, \cdots, N-1 \\ \hat{c}_1 =& \underset{c_1=1,\cdots,N}{\text{argmax}} [\tilde{\lambda_1}(c_1) + s(N+1, c_1)] \\ \hat{c}_N =& \underset{c_N=1,\cdots,N}{\text{argmax}} [\tilde{\delta}_{N-1}(c_N) + \tilde{\lambda}_N(c_N) + s(c_N, N+1)] \end{aligned}$$

Date s e il fattore di *dampening* θ, si inizializzano a 0 i messaggi $\tilde{\phi}_{it}$, $\tilde{\gamma}_{it}$, $\beta_t(m)$, $\delta_t(m)$, $\lambda_t(m)$. Ognuno di essi viene quindi aggiornato dall'esecuzione di max-sum che termina dopo un numero fissato di iterazioni o dopo aver rilevato un risultato costante per un certo numero di iterazioni.
In Figura 4.1 è possibile visualizzare un esempio di istanza per il problema.

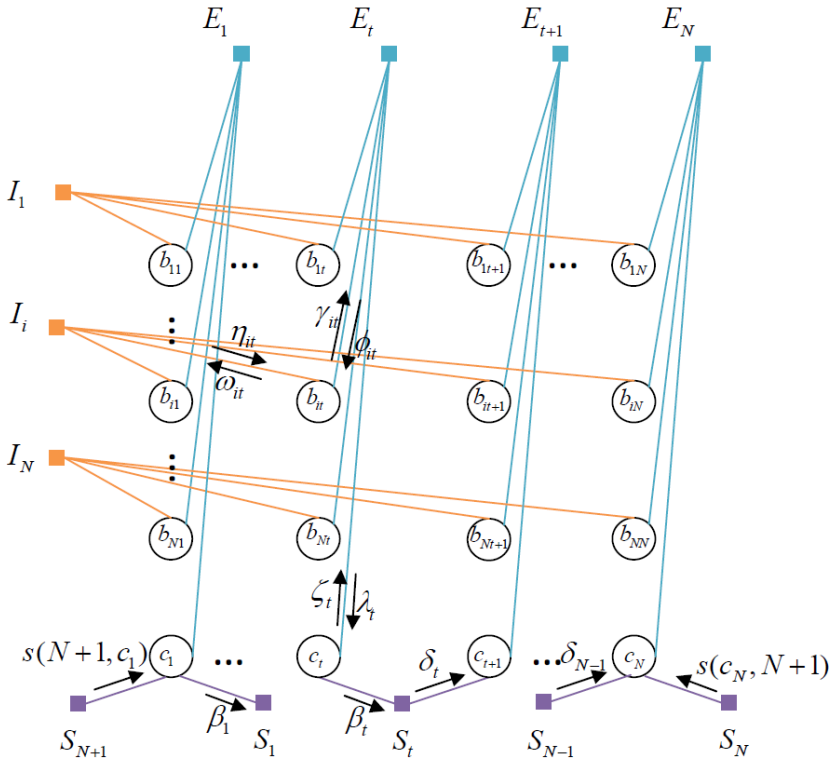


Figura 4.1: Formalizzazione di istanza per max-sum di TSP

4.2 Coordinamento di UAV

In questa sezione sarà presentata un'applicazione dell'algoritmo max-sum al controllo di UAVs pubblicata in [48].

Gli UAVs (Unmanned Aerial Vehicles)sono una tecnologia particolarmente utile nell'ambito della sorveglianza su vasto raggio poichè sono dispositivi relativamente economici, dotati di sensoristica di vario genere, durevoli e capaci di comunicare tra loro via radio.
Il problema che si propone di affrontare l'articolo citato consiste nel coordinare un gruppo di UAV al fine di gestire un insieme di richieste, inviate da un operatore umano, inerenti la sorveglianza di una zona più ampia rispetto al raggio di comunicazione degli UAV stessi.
Rispetto ad altri problemi riguardanti sistemi multi-robot, questo caso assume

che gli UAV possano comunicare tra loro (entro un raggio fissato) e che non ci sia uno scheduler centralizzato che assegna le zone da sorvegliare al singolo robot. Lo scenario reale che motiva queste assunzioni è il seguente: si consideri un gruppo di UAV che monitorano una porzione di terreno molto ampia al fine di segnalare casi di incendio, in questo caso un coordinamento centralizzato non sarebbe realizzabile poichè il raggio di comunicazione dei robot, notevolmente inferiore rispetto all'estensione dell'area interessata, non lo permetterebbe.
Le soluzioni proposte sono due: la prima considera il costo associato al servire una richiesta indipendente dalle altre mentre la seconda considera anche un fattore di carico dipendente dal numero di richieste assegnate al robot.

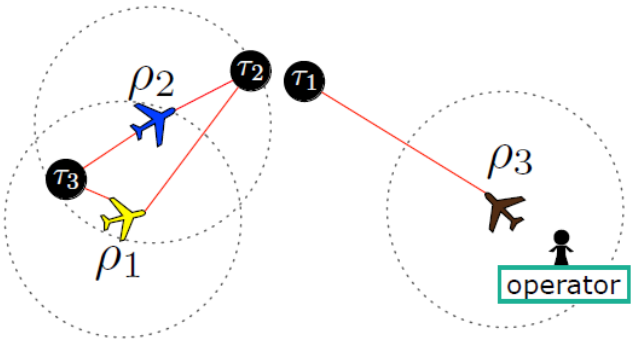


Figura 4.2: Scenario del problema di coordinamento di UAV

4.2.1 Formalizzazione del problema

Modello con MRF binario

Una formalizzazione dello scenario può essere effettuata attraverso MRF (Markov Random Field) binario. Siano $R = \tau_1, \cdots, \tau_m$ un insieme di richieste e $P = \rho_1, \cdots, \rho_n$ un insieme di UAV, r e p gli indici che identificano un generico elemento dei due insiemi rispettivamente. Siano $R_p \subseteq R$ l'insieme delle richieste che l'UAV p può servire e $P_r \subseteq P$ l'insieme degli UAV che possono servire la richiesta r.

La codifica su MRF è la seguente:

- ogni richiesta τ_r viene rappresentata nella variabile x_r il cui dominio è l'insieme P_r degli UAV che può servire la richiesta. Se $x_r = \rho_p$ allora τ_r sarà servita da ρ_p.
- ogni UAV ρ_p possiede un vincolo n-ario c_p che valuta il costo per il robot di servire la richiesta a lui assegnata. Data l'assunzione di indipendenza delle richieste, il costo totale per un insieme di richieste è dato dalla somma dei singoli costi

Siano X_p l'insieme delle variabili che nel loro dominio contengono ρ_p, $\mathbf{X_p}$ un assegnamento di valori su questo insieme; risolvere il problema consiste nel trovare

la combinazione di assegnazioni UAV-richieste $\mathbf{X}^*$ che soddisfa

$$\mathbf{X}^* = \underset{\mathbf{X}}{\operatorname{argmin}} \sum_{p \in P} c_p(\mathbf{X_p})$$

Un'esempio di istanza di problema e relative formalizzazioni sono rappresentate in Figura 4.2 e 4.3. Le formalizzazioni proposte sono due: la prima presenta il problema che i valori del vincolo c_p, dipendendo dal prodotto del numero di valori che possono assumere le variabili, crescono esponenzialmente rispetto alle richieste dello UAV ρ_p e la seconda lo risolve sfruttando l'indipendenza dei costi delle richieste scomponendo il vincolo in fattori, ognuno dei quali relativo ad una singola richiesta.

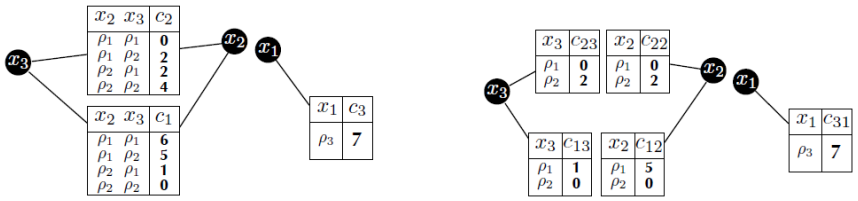


Figura 4.3: Formalizzazione iniziale del problema di coordinamento di UAV

Questa tipologia di modello non possiede una buona scalabilità e non esplicita agevolmente l'assunzione di indipendenza sui costi per servire una richiesta, per questo si procede a suddividere ogni variabile, con dominio di dimensione d, in d variabili binarie legate da un nodo selettore (Sezione 2.4.1). L'istanza per lo scenario di Figura 4.2 è rappresentata in Figura 4.4.

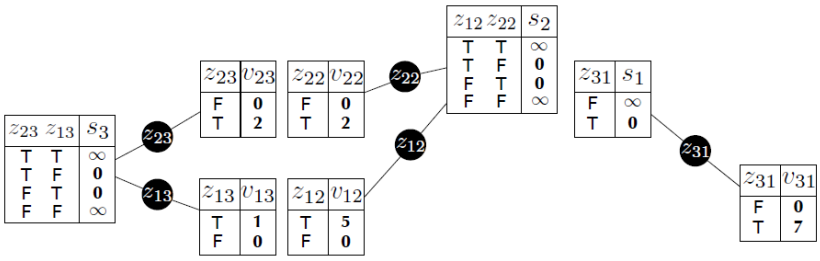


Figura 4.4: Formalizzazione finale del problema di coordinamento di UAV

Soluzione con max-sum

Sull'istanza così generata è possibile applicare max-sum. Poiché ogni nodo variabile binaria è connesso solo al relativo costo e al selettore, il messaggio che

esso invia corrisponde a quello ricevuto: z_{pr} invia a v_{pr} il messaggio ricevuto da s_r e analogamente invia a s_r il messaggio ricevuto da v_{pr}. Questo permette che max-sum consideri solo messaggi diretti da una funzione ad un'altra funzione.
Il generico messaggio che due nodi funzione f e g si scambiano, considerato che $\mathbf{Z_{f\cap g}}$ è l'assegnamento di valori a variabili nel dominio di entrambe le funzioni e $\mathbf{Z_{f-g}}$ quello a variabili presenti solo nel dominio di f, è:

$$\mu_{f\leftarrow g}(\mathbf{Z_{f\cap g}}) = \min_{\mathbf{Z_{f-g}}} \left[f(\mathbf{Z_{f\cap g}}, \mathbf{Z_{f-g}}) + \sum_{h\in\mathcal{N}(f)\setminus g} \mu_{h\leftarrow f}(\mathbf{Z_{h\cap g}}) \right]$$

I messaggi scambiati riguardano il valore che assume la variabile ma, poichè essa è binaria con valori $\{0, 1\}$, si può semplificare il calcolo inviando la differenza tra il valore della variabile attiva e non attiva.
Nello specifico:

- messaggio da nodo costo a selettore: esprime la differenza per lo UAV tra il servire e non servire la richiesta

$$m_{v_{pr}\leftarrow s_r} = v_{pr}(1) - v_{pr}(0) = \delta_{pr}$$

- messaggio da selettore a costo: esprime il costo del servire la richiesta, se il robot è selezionato vale 0 altrimenti è pari al minimo costo che si avrebbe scegliendo un robot diverso

$$\begin{aligned} m_{s_r\leftarrow v_{pr}}(1) &= 0 \\ m_{s_r\leftarrow v_{pr}}(0) &= \min_{\rho_{p'}\in P_r-\rho_p} \delta_{p'r} \end{aligned}$$

4.2.2 Utilizzo del fattore di carico

Nella seconda soluzione, quella che considera il numero di richieste assegnate allo UAV e sulla base di questo valore, modifica il risultato, è più realistica rispetto alla precedente perché permette di non considerare l'assunzione di indipendenza dei messaggi che è troppo restrittiva.
Per ogni UAV si introduce un carico che, aggiunto al costo, permette di conoscere il costo totale per il robot nel servire le richieste. Il *workload* per lo UAV ρ_p è

$$w_p(\mathbf{Z_p}) = f(\eta_p) = k \cdot (\eta_p)^\alpha$$

dove il numero di richieste assegnate a ρ_p è $\eta_p = \sum_{r\in R_p} \mathbf{z_{pr}}$, e k ed a sono parametri da settare per regolare il peso del carico tali che $k \geq 0$ e $\alpha \geq 1$. Quindi il costo totale per ρ_p per servire l'insieme di richieste a lui assegnate è

$$w_p(\mathbf{Z_p}) + \sum_{\tau_r\in R_p} c_{pr}(z_{pr})$$

4.3 Cooperative Control per una rete di sensori

In questa sezione è proposta l'applicazione di max-sum alla gestione di una rete distribuita di sensori proposta in [40].

I recenti sviluppi nell'ambito dei dispositivi per l'elaborazione di dati dotati di sensori multifunzione, hanno portato alla nascita e diffusione di reti di sensori wireless operanti su larga scala (WSN ovvero Wireless Sensor Networks). In una rete di questo tipo ogni nodo è equipaggiato con una serie di sensori in grado di osservare l'ambiente circostante e con un canale di comunicazione; lo scopo della ricerca citata è quello di costituire il *cooperative control* ovvero gestire il controllo e la riconfigurazione dei sensori al fine di massimizzare le performance per l'applicazione richiesta nell'intera rete.
Lo scenario cui si applica la procedura descritta in seguito prevede N sensori fissi impegnati a tracciare M target mobili.

La cooperazione tra sensori può avvenire su due livelli:

- **cooperazione implicita**: si raggiunge attraverso lo scambio diretto di stime o misure (sezione tratteggiata in basso in Figura 4.5). Un esempio di questo è la *Distributed picture compilation*, ovvero il processo di costruzione di un'immagine unica a partire da immagini fornite dai singoli sensori. Un modo per risolvere questo problema è quello di avvalersi di un algoritmo *DDF* (Decentralized Data Fusion).
  Questo algoritmo mantiene informazioni sullo stato dei target attraverso la matrice di informazione di Fisher $\mathbf{Y}(\mathbf{k}|\mathbf{k})$[1].
  L'algoritmo si articola in quattro fasi: ogni sensore stima lo stato del target ($\mathbf{Y}(\mathbf{k}|\mathbf{k}-\mathbf{1})$, vengono selezionati i parametri per il nodo in oggetto al fine di determinare quale target esso debba osservare, vengono riconfigurati i sensori, ognuno di essi osserva il target e trasmette l'informazione raccolta alla rete al fine di calcolare le performance globali della rete.

- **cooperazione esplicita**: è raggiunta con un processo di pianificazione e calcolo congiunto (sezione tratteggiata in alto in Figura 4.5). É il tipo considerato per l'applicazione in oggetto, in particolare il problema affrontato è quello del determinare i parametri di controllo dei sensori al fine di massimizzare l'informazione raccolta.

L'approccio usuale prevede che i parametri di controllo siano scelti localmente, ovvero che ogni sensore massimizzi l'informazione che è in grado di ricavare; in questo caso invece essi verranno scelti in modo da massimizzare l'informazione globale per il sistema.
Il sensore i che osserva il target j ricava l'informazione $\mathbf{I}_{i,j}(k)$ che viene propagata all'intera rete, esso assimila la propria informazione $\mathbf{Y}_{i,j}(k|k-1)$ con quella ricevuta dagli altri sensori:

$$\mathbf{Y}_{i,j}(k|k) = \mathbf{Y}_{i,j}(k|k-1) + \sum_{i=1}^{N} \mathbf{I}_{i,j}(k) \tag{4.14}$$

[1] La notazione $(k|l)$ indica una stima al tempo k condizionata dalle osservazioni effettuate fino al tempo l compreso.

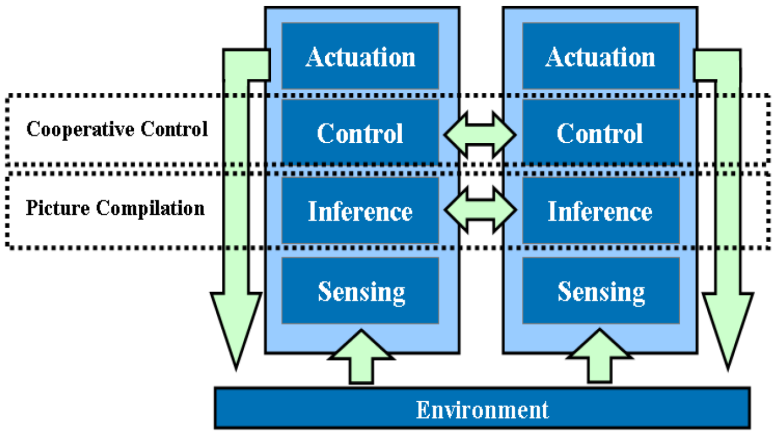


Figura 4.5: Livelli di cooperazione in una WSN

Il rendimento complessivo GI della rete può essere quindi ottenuto misurando l'incertezza delle stime delle posizioni:

$$GI = \sum_{j=0}^{M} \left(\frac{1}{2} \log(2\pi e)^9 (|Y_{i,j}(k|k)|) \right) \tag{4.15}$$

4.3.1 Definizione della funzione utilità

Date le osservazioni precedenti, la funzione di utilità globale può essere definita come

$$U = \sum_{j=0}^{M} \left(\frac{1}{2} \log(2\pi e)^9 (|Y_{i,j}(k|k)|) + \sum_{i=0}^{N} I_{\theta_i,j}(k) \right) \tag{4.16}$$

dove M è il numero di target, N il numero di sensori, $I_{\theta_i,j}$ è l'informazione stimata per il target j dal sensore i che è controllato attraverso il parametro θ_i.
La funzione così definita è fattorizzabile nella somma di sottofunzioni massimizzabili singolarmente che permettono di ridurre la complessità di calcolo.

Il factor graph è costruito nel modo seguente:

- nodo funzione: ogni target j definisce un nodo funzione U_j
- nodo variabile: per ogni sensore i, viene creato un nodo variabile che rappresenta il parametro θ_i

Il grafo è continuamente aggiornato in modo da contenere quei lati che rappresentano solo le effettive e possibili connessioni tra target e sensori, ovvero, ogni θ_i è collegato solo a quei target che si trovano nel raggio in cui esso opera, questa informazione è ottenuta grazie all'ausilio dell'algoritmo DDF.
L'algoritmo viene eseguito in modo standard con l'unica eccezione che durante

l'inizializzazione, e ad ogni spostamento di target che comporta quindi aggiornamento del factor graph, i messaggi inviati per diffondere nella rete la situazione di rottura dell'equilibrio, sono numeri random molto piccoli.

Nello scenario di Figura 4.6 che rappresenta 3 target mobili e 5 sensori fissi e relativo raggio di copertura, la funzione utilità del sistema è:

$$U(\theta_1, \theta_2, \cdots, \theta_N) = U_A(\theta_1, \theta_2, \theta_3) + U_B(\theta_2, \theta_4) + U_C(\theta_5)$$

rappresentata come factor graph in Figura 4.7.

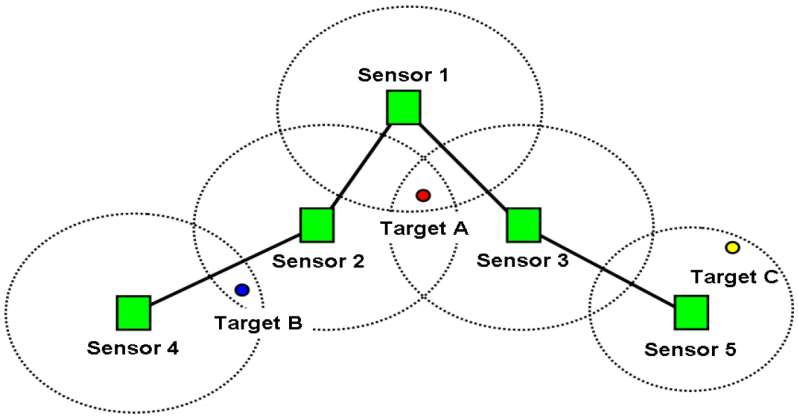


Figura 4.6: Esempio di scenario WSN

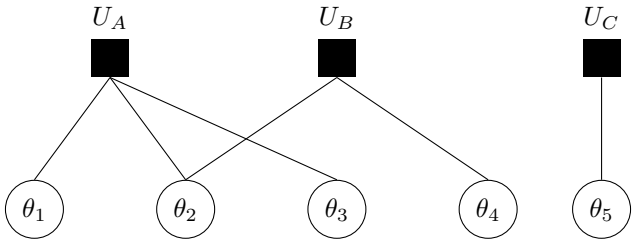


Figura 4.7: Factor graph di per la rete di Figura 4.6

4.4 Coordinazione di radar

In questa sezione è presentata l'applicazione di max-sum al caso del coordinamento di radar di tipo NetRad utilizzati per rilevare e monitorare fenomeni atmosferici sviluppato in [37]. In questo caso, infatti, l'applicazione di max-sum è particolarmente conveniente poiché tale algoritmo permette di generare una policy real-time per lo scheduling grazie alla rapidità con la quale esso restituisce una

soluzione vicina all'ottimo e, poiché in questo contesto ogni evento meteorologico è rappresentato attraverso un task dotato di utilità, il singolo radar beneficia della possibilità di collaborazione con i vicini dato il suo limitato raggio d'azione.

I sistemi NetRad, così come descritti in [50], sono sistemi di radar concepiti per il rilevamento rapido di fenomeni meteorologici di basso livello come i tornado; ogni radar ha corto raggio ed è capace di effettuare la scansione di un determinato volume.
Ogni radar del sistema NetRad è un sistema a catena chiusa MCC (Meterorological command and control) che controlla un insieme di radar. L'utente finale può fornire informazioni sul fenomeno che sta cercando e l'MCC si adatta di conseguenza per stabilire un rank di preferenze su ciò che osserva sulla base delle richieste degli utenti al fine di determinare la miglior strategia di scanning.
Quest'ultima è la strategia che massimizza la somma delle utilità associate ai task scelti basata sulle priorità degli utenti. In Figura 4.8 è possibile visualizzare un sistema di radar e relativo raggio d'azione (settori blu) e un insieme di fenomeni da osservare rappresentati da punti rossi. A destra la strategia di scanning migliore per due radar che necessitano di osservare due fenomeni.

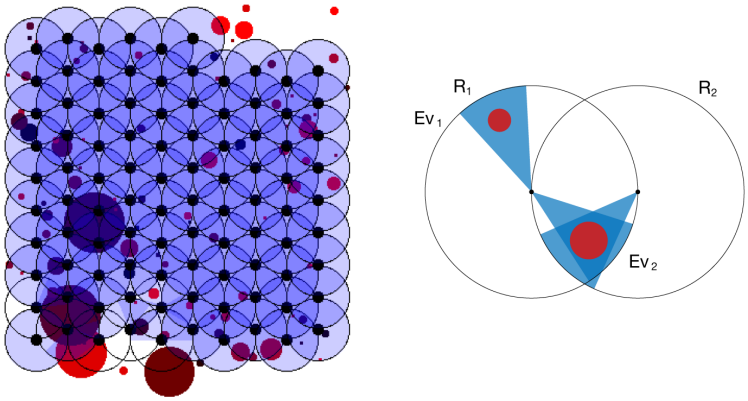


Figura 4.8: Esempio di sistema NetRad.

4.4.1 Rappresentazione della funzione utilità

L'obiettivo è quello di massimizzare l'utilità globale ottenuta come somma delle utilità singole ovvero quelle dei singoli task scansionati.
Ogni radar può scegliere dove effettuare le scansioni selezionando un sottoinsieme di fenomeni nel suo raggio (le possibilità sono un insieme finito), ognuno dei task ha una scansione preferita cioè quella che copre la regione del task in modo più esaustivo.

Per ogni fenomeno t_i, la funzione utilità è u_i è determinata dalla priorità dell'utente o del fenomeno:

$$u_i = t_i \rightarrow r \in \mathbb{R} \; con \; 0 \leq r \leq 1 \tag{4.17}$$

Insieme ad u_i si definisce una funzione per la qualità dello scan, q_i, date le scanning policy r_k dei radar che si trovano nella condizione di poter osservare il fenomeno:

$$q_i : t_i \times (r_1, \cdots, r_n) \rightarrow \in \mathbb{R} \tag{4.18}$$

L'utilità globale del sistema U, la cui massimizzazione permette di trovare la configurazione di $r_1, \cdots, r_n$ cercata, è

$$U = \sum_i u_i \times q_i \tag{4.19}$$

4.4.2 Costruzione del modello

Le formalizzazioni proposte per il problema sono due:

- **coarse-grained**: ad ogni MCC è associato un nodo funzione ed un nodo variabile. Ogni nodo variabile rappresenta una strategia congiunta di scansione per tutti i radar controllati dal MCC, ogni nodo funzione rappresenta l'utilità per i task che possono essere eseguiti dal MCC ed è connesso alle variabili con stesso MCC o MCC vicino diretto. Questa formalizzazione ha il vantaggio di essere indipendente dal numero di task ma la complessità è troppo elevata a causa delle dimensioni dei domini dei nodi variabile.
- **fine-grained**: ogni radar è rappresentato da una variabile il cui dominio è l'insieme delle scansioni che esso è in grado di eseguire; ogni task è identificato da un nodo funzione il cui valore è quello espresso in 4.18 ed è connesso a quei nodi variabili-radar che possono effettuare lo scan del fenomeno. Questa è la formalizzazione utilizzata per lo svolgimento dei test di performance.

4.4.3 Max-sum a due fasi

La versione di max-sum utilizzata con la rappresentazione *fine-grained* è modificata rispetto a quella nota. L'algoritmo procede in due fasi:

- **prima fase - algoritmo di ottimizzazione**: ogni messaggio in uscita da un nodo racchiude informazioni sui messaggi da esso ricevuti all'iterazione precedente e, via via che l'algoritmo esegue, l'informazione raccolta aumenta. Per sfruttare in modo opportuno questa informazione, invece di inviare messaggi di inizializzazione nulli, si utilizza un algoritmo di ottimizzazione a scelta che restituisce un assegnamento di valori per i nodi variabile $\hat{x}$ che viene utilizzato come configurazione di partenza. Il max-sum è modificato in modo che i primi ad inviare messaggi siano sempre i nodi funzione.
  I messaggi scambiati considerano i valori di $\hat{x}$. Se j è l'indice del nodo funzione che crea il messaggio e i quello del nodo variabile che lo riceve, esso è così definito
  $$r_{i \rightarrow j}(x_i) = F_j((\hat{x}_j \backslash i) \cup x_i) \tag{4.20}$$
  e rappresenta il valore di F_j quando i è nello stato x_{ji} mentre tutte le altre variabili si trovano nello stato indicato in $\hat{x}$. Dopo che il nodo variabile ha ricevuto i messaggi, valuta se cambiare il proprio valore che, nel caso, sarà
  $$\tilde{x}_i = \underset{x_i}{\operatorname{argmax}} \sum_{j \in N_i} F_j((\hat{x}_j \backslash i) \cup x_i) \tag{4.21}$$

Se $\hat{x}$ è tale che nessuna variabile possa cambiare il proprio valore per aumentare la funzione utilità globale allora $\hat{x}$ è un assegnamento che soddisfa 4.21. Se cambiando un valore qualunque delle variabili l'utilità globale diminuisce allora $\hat{x}$ è la soluzione unica di 4.21. Questa proprietà sta alla base della scelta di inserire una prima fase nell'algoritmo.

- **seconda fase - algoritmo max-sum**: dopo che i nodi variabili hanno ricevuto i messaggi, sono liberi di scambiarsi altri messaggi prima di procedere con l'esecuzione dell'algoritmo usuale, questo per consentire che l'assegnamento iniziale sia diverso ad ogni esecuzione.

I risultati sperimentali hanno mostrato che le performance di questa versione sono migliori in termini di tempo, questo a causa dell'elevata mole di calcolo richiesta nella versione **coarse-grained**, poiché la convergenza verso la soluzione ottima è molto rapida.

Capitolo 5

Conclusioni

La tecnica adottata in questa tesi per la risoluzione del problema dell'allocazione dei task, può essere confrontata con algoritmi differenti al fine di valutarne punti di forza e punti deboli.

L'algoritmo per la distribuzione di compiti adottato è motivato dalle assunzioni seguenti:

- **scenario generico**: lo scenario in riferimento al quale è stato pensato l'algoritmo è la lega @work ma, modificando opportunamente la definizione dei task, esso è facilmente adattabile e riutilizzabile nella lega @home. Nel file che viene fornito in ingresso al metodo `getInstanceFromfile(`*filepath*`)` (o analogo) non viene richiesto di inserire alcun dettaglio sull'esecuzione del task, bensì se ne indicano solo il punteggio e la dotazione richiesta al robot.

- **assenza di leader**: l'algoritmo non prevede una distinzione di ruoli nel team di robot e quindi non richiede l'individuazione di uno o più leader cui delegare la scelta delle allocazioni

- **robustezza**: l'essere un algoritmo distribuito lo rende più robusto alla perdita di messaggi/errori rispetto ad uno centralizzato

- **utilità globale**: le specifiche dell'algoritmo max-sum fanno si che ogni robot non persegua il fine migliore per se stesso ma quello migliore per il sistema. Questo proibisce il verificarsi di situazioni in cui, se permessa la collaborazione e più robot scelgono di eseguire il medesimo task, uno o più compiti non vengono soddisfatti.

- **tempo per il pre-processing**: a volte lo scenario di una challenge prevede una valutazione delle performance, oltre che in termini di punteggio raggiunto, anche in termini di tempo. In questo caso, l'assenza di uno scheduler centrale e la distribuzione del calcolo, permettono di ridurre il tempo richiesto per la fase preliminare all'esecuzione del task.
  Questa valutazione in riferimento al tempo avviene solo per alcune tipologie di task in cui è definita una durata massima di permanenza del/dei robot nell'arena o è misurato il tempo necessario all'esecuzione della totalità dei task proposti: può essere loro richiesto di eseguire, nel minor tempo possibile, un determinato task fornendo un bonus per il tempo risparmiato

rispetto a quello offerto, oppure, può essere chiesto loro di ripetere, con successo, il maggior numero di volte possibile un'operazione ben definita quale ad esempio il riconoscimento di un oggetto e la spostamento in un container specifico secondo la sua forma e dimensione.

A titolo di esempio si osservino le figure seguenti. In Figura 5.1 sono riportati oggetti e superficie sulla quale svolgere il Precision Placement test (PPT) di RoboCup@work che prevede il riconoscimento della forma di un oggetto e la collocazione nella cavità specifica allineandone la sezione. Questo task trae vantaggio dall'esecuzione in parallelo coordinata da parte di più robot se sono imposti vincoli temporale.
Analogamente, in Figura 5.2 sono rappresentati i singoli componenti e l'oggetto finito ottenuto da un task di assemblaggio pensato da RoCKIn@work. In questo caso la suddivisione di compiti tra più robot, la loro gestione distribuita e continua, può portare vantaggio al sistema apportando un risparmio di tempo rispetto ad un coordinamento centralizzato iniziale.

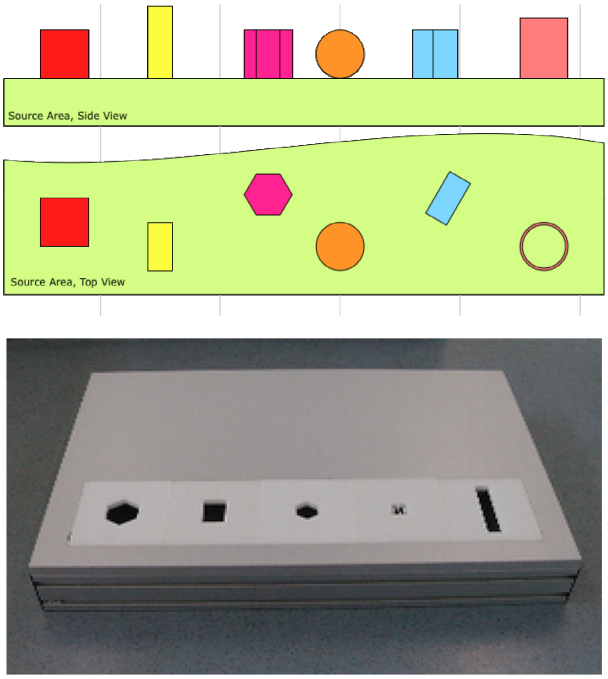
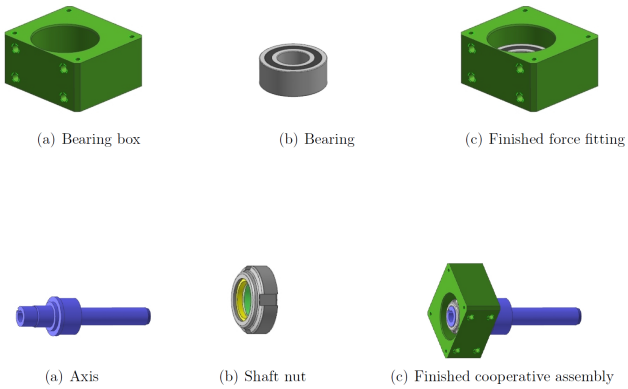


Figura 5.1: Illustrazione del Precision Placement Test di RoboCup@work.



(a) Bearing box (b) Bearing (c) Finished force fitting

(a) Axis (b) Shaft nut (c) Finished cooperative assembly

Figura 5.2: Illustrazione di diversi task di assemblaggio per RoCKIn@work.

5.1 Valutazione delle performance

La complessità, in termini di numero di operazioni effettuate, di un algoritmo del framework GDL, è stata dimostrata in [36] ed è pari a

$$4 \sum_{v} d(v) q_v$$

dove v identifica il generico vertice del grafo cui è applicato, $d(v)$ il suo grado e q_v è la cardinalità dell'insieme ottenuto dal prodotto cartesiano dei domini delle variabili di competenza di v.
Il tempo richiesto per l'esecuzione di questa tipologia di algoritmi varia in relazione al grafo sul quale vengono applicati, alla presenza di cicli e alla modalità con la quale è operata la fattorizzazione della funzione globale. Se questa fattorizzazione non è univoca, ne esisterà almeno una che permetterà di ottenere un tempo di esecuzione inferiore rispetto alle altre e, al fine di ottenere le prestazioni migliori, sarà opportuno che essa sia quella rappresentata nel grafo [36].

Analisi temporale

Il caso di applicazione di max-sum sviluppato in questa tesi non fornisce libertà di scelta sulla fattorizzazione del factor graph poiché essa è strettamente legata allo scenario, ovvero alla disposizione di task e robot all'interno dell'arena.
Per ciò che riguarda la versione binaria dell'algoritmo, come accennato in precedenza, l'effettivo miglioramento delle performance si ha in presenza di grafi di dimensioni molto elevate [46] e quindi, nei casi presi in esame, esso non è sempre apprezzabile.
Si osservino in proposito i dati di Tabella 5.1[1]che evidenziano l'impatto dell'aumento del numero di nodi sul tempo di esecuzione dell'algoritmo, in particolare per quello ordinario.

[1]Tutti i tempi di esecuzione riportati in questa sezione sono stati ottenuti come media matematica su cinque prove per istanza. Le prove sono state effettuate su un pc con processore Intel® Core™ i7-3612QM CPU 2.10GHz sul quale era in esecuzione il solo algoritmo.

Scenario	Istanza			Tempo
	Tipo	# nodi variabile	# nodi funzione	
2 robot,2 task	ord.	2	2	47ms
	bin.	4	8	124ms
3 robot,3 task	ord.	3	3	78ms
	bin.	7	13	125ms
4 robot,4 task	ord.	4	4	219ms
	bin.	12	20	218ms
5 robot,5 task	ord.	5	5	1154ms
	bin.	18	28	359ms
6 robot,6 task	ord.	6	6	811ms
	bin.	20	32	281ms
7 robot,7 task	ord.	7	7	889ms
	bin.	25	39	327ms
8 robot,8 task	ord.	8	8	1887ms
	bin.	27	43	359ms
9 robot,9 task	ord.	9	9	2652ms
	bin.	30	48	406ms

Tabella 5.1: Tempi di esecuzione dell'algoritmo per esecuzione dell'algoritmo con istanza ordinaria e binaria.

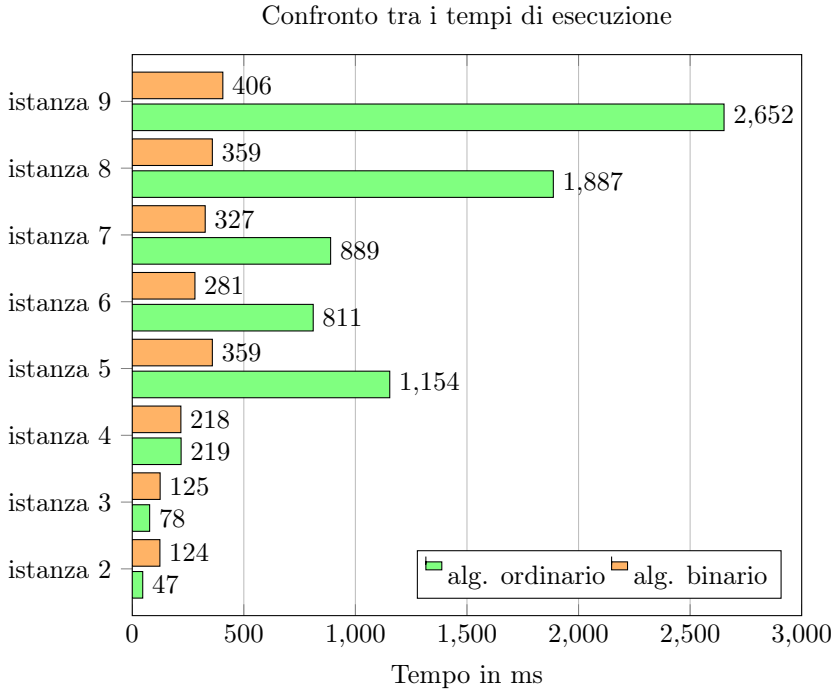


Figura 5.3: Ortogramma dei risultati di Tabella 5.1.

Le istanze esaminate comprendono scenari con robot tutti della stessa tipologia; se ve ne fossero di tipi diversi, infatti, la taglia effettiva del problema sarebbe inferiore, equivarrebbe cioè al risolvere un problema su più zone di taglia minore. Esse sono ottenute a partire da quella di taglia immediatamente inferiore aggiungendo un robot, un task ed un numero di task pari a quello specificato di seguito:

- *istanza 3*: ottenuta da *istanza 2* aggiungendo 3 collegamenti
- *istanza 4*: ottenuta da *istanza 3* aggiungendo 5 collegamenti
- *istanza 5*: ottenuta da *istanza 4* aggiungendo 6 collegamenti
- *istanza 6*: ottenuta da *istanza 5* aggiungendo 2 collegamenti
- *istanza 7*: ottenuta da *istanza 6* aggiungendo 5 collegamenti
- *istanza 8*: ottenuta da *istanza 7* aggiungendo 3 collegamenti
- *istanza 9*: ottenuta da *istanza 8* aggiungendo 3 collegamenti

Le valutazioni sono state ottenute applicando sulla medesima istanza il risolutore `Aphrodite` su un oggetto `Challenge` per la versione ordinaria della rappresentazione ed `Era` su un oggetto `Challenge_BinaryV` per quella binaria.
I dati sono stati raccolti misurando il tempo di esecuzione dell'algoritmo max-sum (non è incluso il tempo per la costruzione dell'istanza) avviando i due test nelle medesime condizioni e imponendo sempre il numero di iterazioni massimo pari a 100.
Come evidenziato in Figura 5.3, le performance dell'algoritmo ordinario sono migliori per taglie di istanza inferiori a 5 robot - 5 task, da questa taglia in poi prevalgono nettamente le performance della versione binaria nonostante il numero di nodi sia sempre maggiore, questo a causa del peso della fase di calcolo e scambio dei messaggi.

Le migliori prestazioni della versione binaria sono particolarmente evidenti nei casi in Tabella 5.2, in cui le istanze utilizzate in precedenza sono state modificate incrementando il numero di connessioni tra robot e task di poche unità, nello specifico:

- *istanza 7 modificata*: ottenuta da *istanza 7* aggiungendo 4 connessioni
- *istanza 8 modificata*: ottenuta da *istanza 8* aggiungendo 3 connessioni
- *istanza 9 modificata*: ottenuta da *istanza 9* aggiungendo 4 connessioni

Questo equivale ad aggiungere connessioni robot-task (cioè nella nuova situazione aumentano i task che il/i robot può/possono eseguire), come evidenziato dal confronto tra il numero di nodi variabile e nodi funzione delle corrispondenti istanze binarie.
In questo caso è evidente il miglioramento di prestazioni apportato dal passaggio a funzioni a variabili binarie, nonostante il consistente aumento del numero di nodi del grafo.

Scenario	**Istanza**			**Tempo**
	Tipo	# nodi variabile	# nodi funzione	
7 robot,7 task	ord.	7	7	11'263ms
	bin.	29	43	343ms
8 robot,8 task	ord.	8	8	19'890ms
	bin.	30	46	452ms
9 robot,9 task	ord.	9	9	20'623ms
	bin.	34	52	437ms

Tabella 5.2: Tempi di esecuzione per istanze con elevato numero di connessioni.

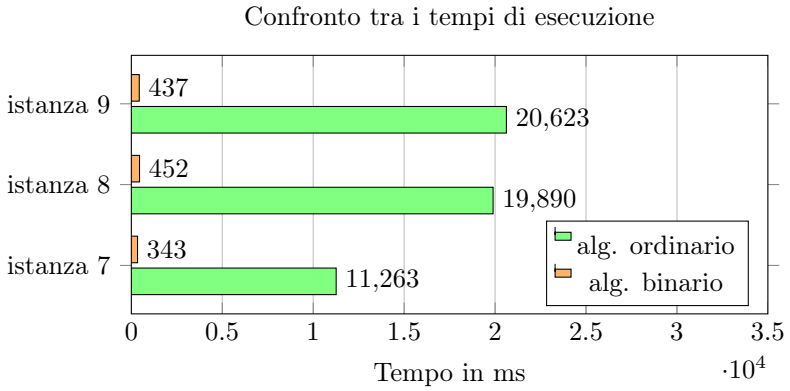


Figura 5.4: Ortogramma dei risultati di Tabella 5.2.

Nel caso di istanza con elevato numero di robot-task suddivisi in più zone, il tempo di esecuzione risulta inferiore rispetto ad istanza mono zona.
In questo caso, infatti, i factor graph costruiti nella versione a più zone sono di taglia inferiore rispetto all'unico factor graph di un problema a zona unica.
Un esempio di dati raccolti in queste circostanze è mostrato in Tabella 5.3.

# zone	**Istanza (8 robot)**			**Tempo**
	Tipo	# nodi variabile	# nodi funzione	
1 zona	ord.	8	8	702ms
	bin.	25	41	328ms
2 zone	ord.	8	8	156ms
	bin.	16	32	234ms
3 zone	ord.	8	8	93ms
	bin.	15	31	188ms

Tabella 5.3: Tempi di esecuzione al variare del numero di zone.

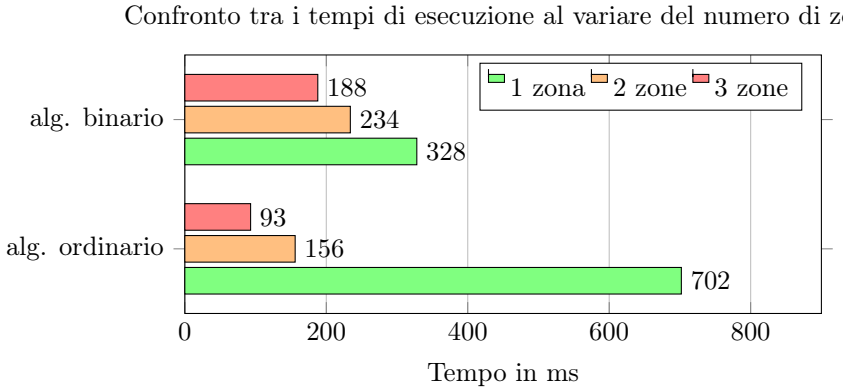


Figura 5.5: Ortogramma dei risultati di Tabella 5.3.

All'aumentare del numero di zone, la differenza tra i tempi di esecuzione dei due algoritmi diminuisce. Tutti i test sono stati effettuati sulla medesima istanza con 8 robot ed 8 task (istanza diversa rispetto alle precedenti) suddivisa prima in due zone della stessa dimensione e poi in tre zone di dimensione 3robot-3robot-2robot.

5.1.1 Confronto tra strategie per l'allocazione di task

In questa sezione verrà proposto un confronto di performance in termini di punteggio raggiunto da differenti algoritmi per alcuni scenari elementari.

Negli esempi che seguiranno verranno confrontati, ed identificati come indicato, le procedure di allocazione di task:

- $\mathbf{A_1}$: algoritmo max-sum trattato in questa tesi (originario o binario a seconda dell'istanza di problema);
- $\mathbf{A_2}$: algoritmo distribuito, ogni robot persegue la propria utilità in termini di punteggio. Comunica con gli altri membri del team solo per ciò che concerne l'evitare conflitti nel *path planning*. Se più task offrono lo stesso punteggio sceglierà di eseguire il più vicino.
- $\mathbf{A_3}$: algoritmo distribuito, ogni robot persegue la propria utilità in termini di vicinanza la task. Comunicazione analoga al caso di A_2. Se più task sono alla stessa distanza, scegliere di eseguire quello che offre maggior punteggio.

Esempio di comparazione 5.1. Si consideri l'istanza di Tabella 5.4 (e i suoi dettagli in Tabella 5.6) l'applicazione dei tre algoritmi restituisce i risultati riportati sotto.

Lo scenario prevede 5 robot e 5 task, tre robot di tipo `P` e due di tipo `2B`. Analogamente i task richiedono: tre robot di tipo `P` e due di tipo `2B`.
Non è permessa collaborazione e nemmeno *load*.

Istanza
ZONE 1
TASKS 5
T 1 1 P 100
T 2 1 P 100
T 3 1 P 100
T 4 1 2B 200
T 5 1 2B 100
ROBOTS 5
R 1 1 P (1,3) (2,4)
R 2 1 P (1,2) (2,3) (3,8)
R 3 1 P (2,6) (3,5)
R 4 1 2B (4,5) (5,1)
R 5 1 2B (4,3) (5,6)

Tabella 5.4: Istanza per Esempio di comparazione 5.1.

Robot	**(task, distanza, punteggio)**	$\mathbf{A_1}$	$\mathbf{A_2}$	$\mathbf{A_3}$
R_1	(1,3,67)			√
	(2,4,75)	√	√	
R_2	(1,2,100)	√	√	√
	(2,3,100)		$(\surd)^a$	
	(3,8,63)			
R_3	(2,6,50)			
	(3,5,100)	√	√	√
R_4	(4,5,120)		√	
	(5,1,100)	√		√
R_5	(4,3,200)	√	√	√
	(5,6,17)			

Tabella 5.6: Dettagli sulla ripartizione di punti e assegnazioni per l'istanza di Tabella 5.4.

- $\mathbf{A_1}$: punti 575, tutti i task sono soddisfatti
- $\mathbf{A_2}$: punti al più 475 (595 ma con task 4 eseguito da due robot, decremento di 120 per l'esecuzione di t_4 che comporta minor punteggio)
- $\mathbf{A_3}$: punti al più 500 (567 ma con task 2 eseguito da due robot, decremento di 67 per l'esecuzione di t_2 che comporta minor punteggio)

L'algoritmo che restituisce l'allocazione a maggior punteggio soddisfacendo tutti i vincoli è $\mathbf{A_1}$, gli altri restituiscono coppie (robot-task) che non soddisfano uno dei task.

[a]Tra parentesi sono indicati i task selezionati perché di punteggio più elevato ma scartati poichè non sono quelli a distanza più breve.

Esempio di comparazione 5.2. Si consideri l'istanza di Tabella 5.7 ed i dettagli in Tabella 5.12).
Lo scenario prevede 5 robot e 5 task, tutti richiedono ed offrono lo stesso tipo di dotazioni. Ogni task offre lo stesso punteggio, non sono permessi collaborazione e *load*. I robot si trovano nelle condizioni di poter eseguire almeno quattro dei cinque task.

- $\mathbf{A_1}$: punti 475, tutti i task sono soddisfatti
- $\mathbf{A_2}$: punti 475 (la fase di scelta nel caso di possibilità di scelta tra task restituisce lo stesso assegnamento di max-sum)
- $\mathbf{A_3}$: punti al più 400 (467 ma con t_1 eseguito da due robot, decremento minimo di 67 punti per R_1)

In questo caso $\mathbf{A_1}$ e $\mathbf{A_2}$ restituiscono lo stesso assegnamento ottimale.

Istanza
ZONE 1
TASKS 5
T 1 1 P 100
T 2 1 P 100
T 3 1 P 100
T 4 1 P 100
T 5 1 P 100
ROBOTS 5
R 1 1 P (1,3) (2,4) (4,5) (5,8)
R 2 1 P (1,2) (2,3) (3,8) (5,4)
R 3 1 P (1,4) (2,6) (3,5) (4,2)
R 4 1 P (1,2) (2,3) (3,7) (4,5) (5,1)
R 5 1 P (1,3) (3,1) (4,3) (5,6)

Tabella 5.7: Istanza per Esempio di comparazione 5.2.

Robot	**(task, distanza, punteggio)**	$\mathbf{A_1}$	$\mathbf{A_2}$	$\mathbf{A_3}$
R_1	(1,3,67)			√
	(2,4,75)	√	√	
	(4,5,40)			
	(5,8,13)			
R_2	(1,2,100)	√	√	√
	(2,3,100)		(√)	
	(3,8,13)			
	(5,4,25)			
R_3	(1,4,25)			
	(2,6,17)			
	(3,5,20)			
	(4,2,100)	√	√	√
R_4	(1,2,100)		(√)	
	(2,3,100)		(√)	
	(3,7,15)			
	(4,5,40)			
	(5,1,100)	√	√	√
R_5	(1,3,67)			
	(3,1,100)	√	√	√
	(4,3,67)			
	(5,6,17)			

Tabella 5.9: Dettagli sulla ripartizione di punti e assegnazioni per l'istanza di Tabella 5.7.

Esempio di comparazione 5.3. Si consideri l'istanza di Tabella 5.10 dettagliata in Tabella 5.9.

Lo scenario comprende 4 task e 4 robot con le medesime richieste e dotazioni. Due robot impongono *load* e tre task consentono la collaborazione fornendo un bonus.
Esito dell'esecuzione:

- $\mathbf{A_1}$: punti 1093, tutti i vincoli sono soddisfatti
- $\mathbf{A_2}$: punti 928 ma non viene eseguito t_1
- $\mathbf{A_3}$: punti 864, tutti i vincoli sono rispettati

In questo caso $\mathbf{A_1}$ restituisce il punteggio migliore ma anche $\mathbf{A_3}$ restituisce un assegnamento valido.

Istanza
ZONE 1
TASKS 5
T 1 1 1B 100 C 20
T 2 1 1B 200
T 4 1 1B 250 C 15
T 5 1 1B 125 C 10
ROBOTS 5
R 1 1 1B (1,3) (4,6) (5,3) L 5
R 2 1 1B (1,7) (2,4) (5,7)
R 3 1 1B (1,4) (4,8) (5,4) L 5
R 4 1 1B (1,4) (2,6) (4,3)

Tabella 5.10: Istanza per Esempio di comparazione 5.3.

Robot	**(task, distanza, punteggio)**	$\mathbf{A_1}$	$\mathbf{A_2}$	$\mathbf{A_3}$
R_1	(1,3,100)	√		√
	(4,6,100)	√	√	
	(5,3,125)	√	√	√
R_2	(1,7,43)			
	(2,4,200)	√	√	√
	(5,7,54)			
R_3	(1,4,75)	√		√
	(4,8,188)	√	√	
	(5,4,94)	√	√	√
R_4	(1,4,75)			
	(2,6,134)			
	(4,3,250)	√	√	√

Tabella 5.12: Dettagli sulla ripartizione di punti e assegnazioni per l'istanza di Tabella 5.10.

Dallo studio dei casi appena esposti e loro generalizzazione, si possono formulare le seguenti osservazioni:

- max-sum è l'unico algoritmo che restituisce sempre un'assegnazione valida che rispetta tutti i vincoli
- in alcuni scenari l'assegnamento prodotto da max-sum può coincidere con quello di uno (o entrambi) gli algoritmi
- nel caso di robot equidistanti da task dello stesso punteggio, max-sum si troverebbe in una situazione di stallo mentre un algoritmo a priorità permetterebbe di eseguire parte, o eventualmente la totalità, dei compiti

5.2 Sviluppi futuri

Integrazione con ROS

La soluzione proposta è stata sviluppata in linguaggio `Java` al fine di poter incorporare il pacchetto `jamaxsum` citato in precedenza. Poiché, per ciò che concerne la lega RoCKIn@work, gli organizzatori hanno posto particolare enfasi nell'utilizzo di un framework comune al fine di poter utilizzare i dati prodotti per il *benchmarking*, è stato indicato l'utilizzo di `ROS` (Robot Operating Sytem). Esso è in grado di integrare in modo ottimale strutture sviluppate in linguaggio `C/C++` e in modo ancora sperimentale pacchetti scritti in linguaggio `Java`; al fine di rendere la soluzione integrabile con esso, è perciò necessario provvedere ad un'eventuale traduzione.

Estensione a challenge diverse rispetto alla lega @work

Come accennato in precedenza, il lavoro è stato sviluppato in riferimento alla challenge sulla robotica industriale poiché nei Camp RoCKIn cui ho preso parte, il mio interesse è stato rivolto ad essa. Tuttavia, ridefinendo opportunamente la struttura del Task per renderlo aderente alle specifiche di challenge multirobot di leghe diverse (robotica di servizio, rescue o soccer), l'allocazione di compiti può essere eseguita con le stesse modalità.

Specializzazione della struttura Task

La definizione di task utilizzata in questo lavoro è elementare ovvero, un compito è sempre stato considerato come entità indivisibile.
Per alcuni task che possono essere eseguiti, in alcune loro sottoparti, in modo parallelo da più robot, potrebbe essere utile suddividere il task globale in sotto compiti al fine di verificare se, date le condizioni iniziali del sistema, sia possibile raggiungere un punteggio maggiore o un tempo di esecuzione inferiore attraverso la divisione e ripartizione.

Creatore di istanza da interfaccia grafica/sensori

La modalità prevista per il passaggio dell'istanza prevede che vengano indicati manualmente in un file di testo, per ogni robot, i task che esso vede e la distanza. In presenza di robot reale, o eventualmente simulato, dotato di sensori, si potrebbe automatizzare la raccolta di queste informazioni e conseguente creazione di istanza.
Analogamente per i task da eseguire, ammesso che ne venga fornito un database con le specifiche tecniche, si potrebbe automatizzare la creazione della sezione di istanza ad essi relativo selezionando, di volta in volta da un elenco, il nome con il quale vengono identificati nella challenge.

Elenco delle figure

Elenco delle tabelle

Bibliografia

[1] Jenhui Chen and Li-ren Li, *Path Planning Protocol for Collaborative Multi-Robot System*, IEEE International Symposium on Computational Intelligence in Robotics and Automation, 2005

[2] Steven M. LaValle, *Planning Algorithms*, Cambridge University Press, 2006

[3] Roland Siegwart and Illah R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, The MIT Press, Cambridge, 2011

[4] Gerhard Weiss, *Multiagents Systems*, The MIT Press, 2013

[5] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006

[6] Henk Wymeersch, *Iterative Receiver Design*, Cambridge University Press, 2007

[7] Meritxell Vinyals, Juan A. Rodriguez-Aguilar and Jesús Cerquides, *Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law*, Journal Autonomous Agents and Multi-Agent Systems Volume 22, Issue 3 , pp 439-464, 2011

[8] Jesus Cerquides, Alessandro Farinelli, Pedro Meseguer and Sarvapali D. Ramchurn, *A tutorial on optimisation for multi-agent systems*, The Computer Journal, 2013

[9] A.Ahmad, I.Awaad, F.Amigoni, J.Berghofer, R.Bischoff, A.Bonarini, R.Dwiputra,G. Fontana, F. Hegger, N.Hochgeschwender, L.Iocchi, G.Kraetzschmar, P.Lima, M.Matteucci, D.Nardi, V.Schiaffionati and S.Schneider *Specification of General Features of Scenarios for benchmarking Through Competitions*, 2013

[10] A.Ahmad, F.Amigoni, I.Awaad, J.Berghofer, R.Bischoff, A.Bonarini, R.Dwiputra, F. Hegger, N.Hochgeschwender, L.Iocchi, G.Kraetzschmar, P.Lima, M.Matteucci, D.Nardi and S.Schneider *RoCKIn@Work - Innovation in Industrial Mobile Manipulation - Competition Design, Rule Book ans Scenario Constructions*, 2013

[11] A.Ahmad, F.Amigoni, I.Awaad, J.Berghofer, R.Bischoff, A.Bonarini, R.Dwiputra, F. Hegger, N.Hochgeschwender, L.Iocchi, G.Kraetzschmar, P.Lima, M.Matteucci, D.Nardi and S.Schneider *RoCKIn@Home - A Competition for Domestic Service Robots - Competition Design, Rule Book ans Scenario Constructions*, 2013

[12] Jakob Berghofer, Rhama Dwiputra, Gerhard Kraetzschmar et alii, *RoCKIn@Work in a Nutshell*, `http://rockinrobotchallenge.eu/rockin_work_nutshell.pdf`

[13] Sven Schneider, Gerhard Kraetzschmar et alii, *RoCKIn@Home in a Nutshell*, `http://rockinrobotchallenge.eu/rockin_home_nutshell.pdf`

[14] L. Iocchi, G.Kraetzschmar, P.Lima, D. Nardi and G. Randelli, *RoCKIn Camp 2013 Report*, 2012

[15] Gerhard Kraetzschmar, Walter Nowak, Nico Hochgeschwender, Rainer Bischoff, Daniel Kaczor and Frederik Hegger, *RoboCup@Work Rule Book* version 2013

[16] Locomotec, *KUKA youBot User Manual*, 2013

[17] Douwe Dresscher, *Robust autonomy for the youBot*, Master Thesis, University of Twente (NL), 2010

[18] S.Alers, D.Bloembergen, D. Claes, J. Fossel, D. Hennes, K. Tuyls and G.Weiss, *Swarmlab@Work Team description paper 2013*, 2013

[19] Jasper Van Enk, *Navigating YouBot through a rose field with A**, Project Report, Universiteit van Amsterdam, 2013

[20] Shashank Sharma and Gerhard K. Kraetzschmar, *Unified Closed Form Inverse Kinematicf for the KUKA youBot*, 7th German Conference onRobotics, Proceedings of ROBOTIK, 2012

[21] Timothy Jenkel, Richard Kelly and Paul Shepanski, *Mobile Manipulation for the KUKA youBot Platform*, Project Report, Worchester Polytechnic Institute, 2013

[22] Rainer Bischoff, Ulrich Huggebìnberger and Erwing Prassler, *KUKA youBot - a mobile manipulator for research and education*, IEEE International Conference on Robotics and Automation (ICRA), 2011

[23] Dourwe Dresscher, Yury Brodskiy, Peter Breedveld, Jan Broenink and Stefano Stramigioli, *Modeling of the youBot in a serial link structure using twists and wrenches in a bound graph*, 2nd International Conference on Simulation, Modeling, and Programming for Autonomous Robots, SIMPAR 2010 Workshops, Darmstadt, Germany, 2010

[24] Mads Hvilshøj, Simon Bøgh, Ole Madsen and Morten Kristiansen, *The Mobile Robot "Little Helper": Concepts, ideas and working principles*, IEEE Conference on Emerging Technologies & Factory Automation, Mallorca (ES), 2009

[25] Mads Hvilshoj and Simon Bogh, *"Little Helper" - An Autonomous Industrial Mobile Manipulator Concept*, International Journal of Advanced Robotic Systems, 2011

[26] Angel P. del Pôbil et alii, *Benchmarks in Robotics Research*, Lecture Notes for IROS 2006 Workshop II (WS-2), Tuesday, October 10, 2006

[27] Kaz Kawamura et alii, *Benchmarks in Robotics Research*, Lecture Notes for IROS 2007 Workshop III Monday, October 29

[28] F. Amigoni, A. Bonarini, G. Fontana, M. Matteucci and V. Schiaffonati, *Benchmarking Through Competitions*, 2nd Workshop on Robot Competitions: Benchmarking, Technology Transfer and Education, European Robotics Forum 2013, Lyon, France, 2013.

[29] Fabio Maffioletti, Riccardo Reffato, Alessandro Farinelli, Alexander Kleiner, Sarvapali Ramchurn and Bing Shi *RMASBench: a Benchmarking System for Multi-Agent Coordination in Urban Search and Rescue*, International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2013), Saint Paul, US, 06 - 10 May 2013.

[30] Nam Ma, Yinglong Xia and Victor K. Prasanna, *Task Parallel Implementation of Belief Propagation in Factor Graphs*, IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012

[31] Hans-Andrea Loeliger, *An Introduction to Factor Graphs*, IEEE Signal Processing Magazine, Volume 21 Issue 1, 2004

[32] G.D. Forney Jr., *Codes on graph: Normal realizations*, IEEE Transaction of Information Theory, vol 47, no. 2, pp. 520-548, 2001

[33] Xiaojin Zhu, *Inference in Graphical Models*, Spring 2010 Advanced Natural Language Processing, 2010

[34] B.J. Frey, *Graphical Models for Machine Learning and Digital Communication*, The MIT Press, 1998

[35] Frank R.Kschischang, Brendan J.Frey and Hans-Andrea Loeliger, *Factor Graphs and the Sum-Product Algorithm*, Information Theory, IEEE Transactions on (Volume:47, Issue:2), 2001

[36] Srinivas M. Aji and Robert J. McEliece, *The Generalized Distributive Law*,, IEEE Transactions on Information Theory (Volume:46 , Issue: 2), 2000

[37] Yoonheui Kim, Michael Krainin and Victor Lesser *Application of Max-Sum Algorithm to Radar Coordination and Scheduling*, Twelfth International Workshop on Distributed Constraint Reasoning, pp. 5-19. 2010. Held in conjunction with AAMAS 2010.

[38] Yoonheui Kim, Michael Krainin and Victor Lesser *Effective Variants of the Max-Sum Algorithm for Radar Coordination and Scheduling*, Proceedings of 2011 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp. 357-364, 201 1

[39] Daniel Tarlow, Immar E. Givoni, Richard S.Zemel and Brendan J.Frey, *Graph Cuts is a Max-Product Algorithm*, Proceedings of Uncertainty in Artificial Intelligence, Barcelona (ES), 2011

[40] A. Waldock, D.Nicholson and A.Rogers *Cooperative Control using the Max-Sum Algorithm*, Second International Workshop on Agent Technology for Sensor Networks, Estoril (P), 2008

[41] Nam Ma,Yinglong Xia and Victor K. Prasanna, *Data Parallelism for Belief Propagation in Factor Graphs*, 23rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2011

[42] Yongyi Mao and Frank R.Kschischang, *On Factor Graphs and the Fourier Transform*, IEEE Transactions on Information Theory, (Volume:51, Issue:5) 2005

[43] Yuxiao Huang, Haiyang Jia, Yungang Zhu and Dayou Liu, *A Factor Graph Inference Algorithm for Diagnostic Bayesian Networks*, Seventh International Conference on Natural Computation (ICNC),(Volume 1), 2011

[44] A. Rogers, A. Farinelli, R. Stranders and N. R. Jennings, *Bounded Approximate Decentralised Coordination via the Max-Sum Algorithm*, Artificial Intelligence, Volume 175, Issue 2, February 2011, Pages 730–759

[45] Marc Pujol-Gonzalez, Jesus Cerquides, Gonzalo Escalada-Imaz, Pedro Meseguer and Juan A. Rodriguez-Aguilar , *On Binary Max-Sum and Tractable HOPs*, 11th European Workshop on Multi-Agent Systems, Toulouse (FR), 2013

[46] Daniel Tarlow, Inmar E. Givoni and Richard S. Zemel, *HOP-MAP: Efficient Message Passing with High Order Potentials*, Proceedings of 13th Conference on Artificial Intelligence and Statistics (AISTATS), 2010

[47] I. E. Givoni and B. J. Frey. *A binary variable model for affinity propagation.* Neural Computation 21, 2009.

[48] Marc Pujol-Gonzalez, Jesus Cerquides, Pedro Meseguer, Juan A. Rodriguez-Aguilar1, and Milind Tambe, *Engineering the decentralized coordination of UAVs with limited communication range*,

[49] Chang-Dong, Wang Jian-Huang and Lai Wei-Shi Zheng, *Message-Passing for the Traveling Salesman Problem*,

[50] James F. Kurose, Eric Lyons, David McLaughlin, David Pepyne, Brenda Philips, David Westbrook, and Michael Zink, *An End-User-Responsive Sensor Network Architecture for Hazardous Weather Detection*, Prediction and Response in Proceedings of the Second Asian Internet Engineering Conference, AINTEC, pages 1–15, 2006

[51] Yoonheui Kim, Michael Krainin and Victor Lesser, *Effective Variants of the Max-Sum Algorithm for Radar Coordination and Scheduling*, 2011 IEEE/-WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology

[52] F. M. Delle Fave, A. Farinelli, A. Rogers and N. R. Jennings, *A Methodology for Deploying the Max-Sum Algorithm and a Case Study on Unmanned Aerial Vehicles*, IAAI 2012: The Twenty-Fourth Innovative Applications of Artificial Intelligence Conference, Toronto, CA, 24 - 26 Jul 2012.

Ringraziamenti

Desidero ringraziare i professori *Enrico Pagello* e *Alessandro Farinelli* per la disponibilità e l'interesse dimostrato, l'Ing. *Alberto Pretto* insieme ai responsabili e coordinatori di RoCKIn per le opportunità che il loro progetto mi ha offerto.

Grazie a mamma *Carmela* e a papà *Esterino* per l'amore, per aver assecondato le mie scelte e per non avermi mai fatto mancare nulla.

Grazie a *Gianmarco* per la stima, l'appoggio incondizionato, per averci creduto spesso più di me, per aver contribuito ad accrescere la fiducia in me stessa e per tutte le volte che ha cercato di strapparmi un sorriso nei modi più inusuali.

Grazie a *Stefano* per non aver mai smesso di esserci, per gli abbracci forti e per avermi sempre spinta a guardare oltre.

Grazie a *Raffaele* perché nonostante tutto non ha mai smesso di sostenermi, starmi vicino e tendermi la mano.

Grazie a *Lisa* che mi ha permesso di credere nell'amicizia, per tutte le risate e le telefonate infinite.

Grazie ad *Andrea* per aver condiviso i mesi in laboratorio, per avermi aiutata e aver posto le domande più interessanti.

Grazie a *Erica* per la compagnia nelle lunghe giornate passate a scrivere la tesi e per avermi fornito molti spunti di riflessione.

Grazie a chi ha saputo starmi accanto, a chi non ha mai smesso di credere in me e nei miei desideri, a chi ha sospirato con me, a chi mi ha donato amore e affetto, a chi ha ascoltato e curato il mio dolore, a chi ha saputo perdonarmi e comprendermi, a chi ha rispettato il mio silenzio denso di parole, a chi ha lasciato fluire fiumi di lacrime e le ha asciugate una per volta, a chi nel mio essere donna non ha mai visto un ostacolo.
Grazie a tutti coloro che si sono mostrati sempre orgogliosi della loro piccola INGEGNERA.