



UNIVERSITÀ DEGLI STUDI DI FIRENZE
SCUOLA DI INGEGNERIA - DIPARTIMENTO DI INGEGNERIA
DELL'INFORMAZIONE

Tesi di Laurea Triennale in Ingegneria Informatica

**METODI DI ESPLORAZIONE FINALIZZATI ALLA
COPERTURA DI UTENTI MEDIANTE AGENTI
AUTONOMI**

Candidato
Leonardo Guglielmi

Relatore
Prof. Giorgio Battistelli

Correlatore
Nicola Forti

Anno Accademico 2024/25

Abstract

I sistemi multiagente, quali le reti UAV, sono un paradigma tecnologico molto diffuso che con il passare del tempo ha visto la sua adozione in diversi contesti applicativi. Tra questi, si annovera quello della copertura di utenti mediante sistemi di telecomunicazione, specialmente in quei casi in cui la posizione e il numero di tali utenti sono ignoti. Inserendosi in tale scenario, la seguente tesi si prefigge l'obiettivo di sviluppare un algoritmo di esplorazione dell'area di interesse finalizzato alla ricerca di utenti disposti a terra.

Introduzione

Le reti UAV sono dei sistemi che, grazie alla loro scalabilità, flessibilità e adattabilità, hanno trovato un grande impiego in numerosi ambiti, anche molto eterogenei tra loro, come le comunicazioni nelle fasi di soccorso [1]; il monitoraggio delle coltivazioni [2]; il monitoraggio ambientale [3]; la sorveglianza [4]; l'erogazione di servizi di comunicazione [5]; l'esplorazione di pianeti [6]; e molti altri. In particolare essi possono formare una rete wireless aerea, con la possibilità di scambiarsi reciprocamente informazioni riguardanti l'ambiente circostante, la presenza di eventuali ostacoli, o la posizione di utenti a cui deve essere garantita la copertura di segnale: perciò, oltre alle comunicazioni UAV-to-UAV generalmente questo tipo di rete ha anche la capacità di comunicare con dispositivi localizzati a terra.

Partendo dal modello di copertura esaminato in [7], e supponendo che il numero di utenti totali non sia noto, e che anzi potrebbe variare nel tempo, questa tesi si pone l'obiettivo di sviluppare un algoritmo di esplorazione collaborativo per reti UAV avente il compito di guidare i droni nell'esplorazione dell'area alla ricerca di nuovi utenti, e che al contempo riesca a garantire la copertura di segnale di quelli già connessi alla rete.

In letteratura esistono varie metriche su cui basarsi per costruire un algoritmo atto all'esplorazione di una regione. Questo comporta che al variare della metrica cambi il comportamento medio che gli agenti esibiscono; per esempio alcuni algoritmi potrebbero avere come obiettivo la massimizzazione dell'estensione geografica coperta nel tempo (Mean Square Distance, MSD), favorendo la decisione da parte degli agenti di spostamenti lunghi, mentre altri algoritmi potrebbero concentrarsi sul disincentivare l'esplorazione ripetuta delle stesse aree (rilascio di feromoni virtuali, [8]). La diversità di

questi approcci comporta una serie di vantaggi e svantaggi nel loro utilizzo, che possono andare dal consumo energetico per gli spostamenti, alla reciproca interferenza tra il segnale degli agenti, alla flessibilità verso variazioni dell'ambiente.

In questo progetto si è deciso di sviluppare un algoritmo di esplorazione basandosi sulla probabilità che, nei punti dell'area di interesse non coperti dal sistema in un certo istante, vi possa essere un utente. Il sistema quindi, equipaggiato con tale algoritmo, si focalizzerà sul cercare di ridurre la probabilità media, sia aumentando la propria estensione totale, riducendo quindi le zone non coperte, sia visitando regioni con associata un'alta probabilità.

Struttura della tesi

Il contenuto di questa tesi è strutturato nel seguente modo: all'interno del **capitolo 1** vengono introdotti alcuni concetti fondamentali, legati agli argomenti inerenti al contesto della tesi, quali una breve definizione di sistema multiagente e dei tipi di problema che vengono affrontati. Il **capitolo 2** si concentra sul presentare il modello matematico usato per formalizzare il problema e il sistema; successivamente nel **capitolo 3** si mostra l'implementazione dell'algoritmo di controllo e le strategie risolutive adottate. Il **capitolo 4** riporta lo svolgimento degli esperimenti e l'analisi dei risultati da essi ottenuti. Infine, nel **capitolo 5** vengono tratte le conclusioni sull'esperimento, soffermandosi anche su possibili futuri sviluppi della ricerca sul tema.

Indice

1 Copertura ed esplorazione multiagente	1
1.1 Caratteristiche dei sistemi multiagente	1
1.2 Introduzione alle reti UAV	3
1.3 Formulazione dei problemi	7
1.3.1 Problema della copertura	7
1.3.2 Problema dell'esplorazione	8
2 Modello matematico	10
2.1 Modello del sistema	10
2.2 Modello di copertura	11
2.3 Modello di esplorazione	12
2.4 Algoritmo di controllo	14
3 Implementazione	16
3.1 Schema del progetto	17
3.1.1 Classi Sensor, Agent e Base_stations	17
3.1.2 Classe User	17
3.1.3 Classe Control_function	19
3.2 Esecuzione concorrente	24
3.3 Valutazione dell'esplorazione	25

3.3.1	Metodi sperimentali per la valutazione dell'esplorazione	26
3.3.2	Valutazione dell'esplorazione tramite frontiera con controllo delle celle adiacenti	29
3.4	Controllo per il disaccoppiamento di agenti	32
4	Simulazioni	35
4.1	Valori dei parametri	35
4.2	Simulazioni e analisi dei risultati	37
4.2.1	Esperimento 1: verifica dell'efficacia dell'algoritmo di esplorazione	38
4.2.2	Esperimento 2: valutazione del metodo di campionamento migliore	41
4.2.3	Esperimento 3: variazione della dinamicità dell'ambiente	44
5	Conclusioni	47
Bibliografia		49
Elenco delle figure		51
Elenco delle tabelle		53
Elenco degli snippet		53
Ringraziamenti		55

Capitolo 1

Copertura ed esplorazione multiagente

In questo capitolo viene data una spiegazione introduttiva dei sistemi presi in analisi e dei problemi che vengono affrontati.

1.1 Caratteristiche dei sistemi multiagente

I sistemi multiagente sono una classe di sistemi composti da un insieme entità autonome, dette agenti, capaci di interagire tra loro e con l'ambiente circostante. La definizione di agente, come riporta [9], non è unica e concordata universalmente, ma in generale può essere definito come *un'unità situata in un ambiente, capace di eseguire delle azioni autonome, con la finalità di raggiungere un obiettivo prefissato*.

Un esempio naturale, a cui sono ispirati anche alcuni algoritmi di esplorazione, è rappresentato dalle colonie di formiche: in tali sistemi, migliaia di individui (agenti) agiscono in modo indipendente l'uno dagli altri, collaborando per la ricerca di cibo. Ciascuna formica prende delle decisioni finalizzate

al bene della colonia, basandosi sulla propria visione dell'ambiente che la circonda, e sulle informazioni trasmesse dai suoi simili tramite segnali chimici, come i feromoni.

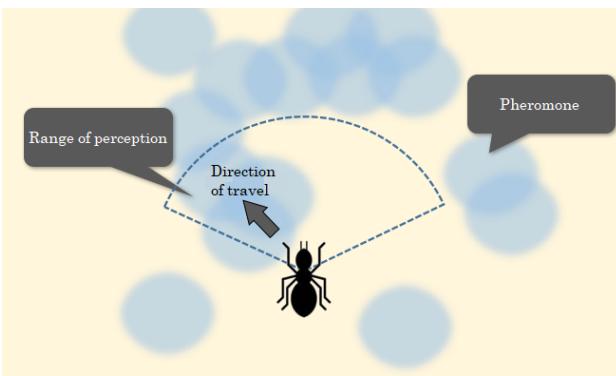


Figura 1.1: Schema di comportamento di una formica.

A partire da questo esempio di sistema biologico, è possibile evidenziare alcuni elementi chiave dei sistemi multiagente:

- **Obiettivo:** ogni agente ha un obiettivo e agisce in modo da raggiungerlo.
- **Autonomia/Indipendenza:** ogni agente compie delle scelte e delle azioni in modo autonomo rispetto agli altri agenti del sistema.
- **Collaborazione:** gli agenti possono collaborare tra loro per il raggiungimento di fini comuni nel caso che essi lo ritengano più utile al raggiungimento dell'obiettivo (non tutti i sistemi risultano collaborativi, si immagini per esempio, una partita di poker in cui ogni agente è un giocatore [10]).
- **Interazione con l'ambiente:** ogni agente è in grado di interagire con l'ambiente circostante, sia per compiere azioni, sia per raccogliere informazioni propedeutiche alla scelta delle azioni da eseguire.

- **Comunicazione:** gli agenti possono avere dei sistemi per comunicare tra loro, con finalità di coordinazione o di scambio di informazioni.

Un'altra caratteristica fondamentale è quella dell' **ambiente** di essere **statico o dinamico**: nel primo caso si intende un ambiente che non varia nel tempo, anche a seguito delle azioni degli agenti, mentre per ambiente dinamico si intende un ambiente variabile nel tempo, e che dunque richiede un'analisi continua da parte degli agenti che lo abitano.

L'elenco esposto non è assolutamente esaustivo di tutte le proprietà che permettono la classificazione dei sistemi multiagente e del loro ambiente [10], dato il loro elevato numero si è deciso di evidenziare solo quelle più rilevanti per il caso d'analisi esaminato.

Analogamente all'esempio riportato precedentemente, il sistema multiagente utilizzato in questa tesi è un sistema collaborativo, con obiettivo globale e condiviso tra gli agenti, inserito all'interno di un ambiente parzialmente osservabile e dinamico, che varia sia in modo autonomo, sia a causa dell'azione degli agenti stessi.

1.2 Introduzione alle reti UAV

I veicoli aerei senza pilota UAV (Unmanned Aerial Vehicle) sono dei mezzi volanti controllati da remoto, o in alcuni casi, completamente autonomi (Figura 1.2).

Le reti UAV [11] sono dei sistemi multiagente composti da un insieme di **droni**, eventualmente anche da stazioni a terra (**Base Stations**, BS), capaci di comunicare tra di loro, che vengono usati come **nodi di una rete wireless** e svolgono la funzione di **ripetitori di segnale**; questo sistema è



Figura 1.2: Esempi di dispositivi UAV

in grado di configurarsi in modo tale da fornire una connessione agli utenti dislocati a terra (Figura 1.3).

L'adozione di questo paradigma per fornire accesso alla rete presenta diversi vantaggi, ad esempio:

- regolando l'altitudine dei droni è possibile aggirare agevolmente ostacoli presenti a terra, permettendo di stabilire collegamenti in linea d'area (Line of Sight, LoS)
- la limitata necessità di strutture fisse a terra permette il dispiegamento dell'infrastruttura di rete in modo rapido, rendendola particolarmente adatta a situazioni di emergenza
- data l'elevata mobilità dei nodi è possibile riconfigurare facilmente la rete a fronte di variazioni delle necessità da parte degli utenti, ottenendo un'alta flessibilità
- data la struttura della rete si ha un'alta scalabilità del sistema

Un altro aspetto rilevante di questo paradigma è quello che, essendo presente una connessione tra gli agenti, è possibile l'interscambio di informazioni riguardanti l'area circostante di ciascun drone, riuscendo quindi a dare ad ogni singolo agente una visione più completa dell'area attualmente esplorata.

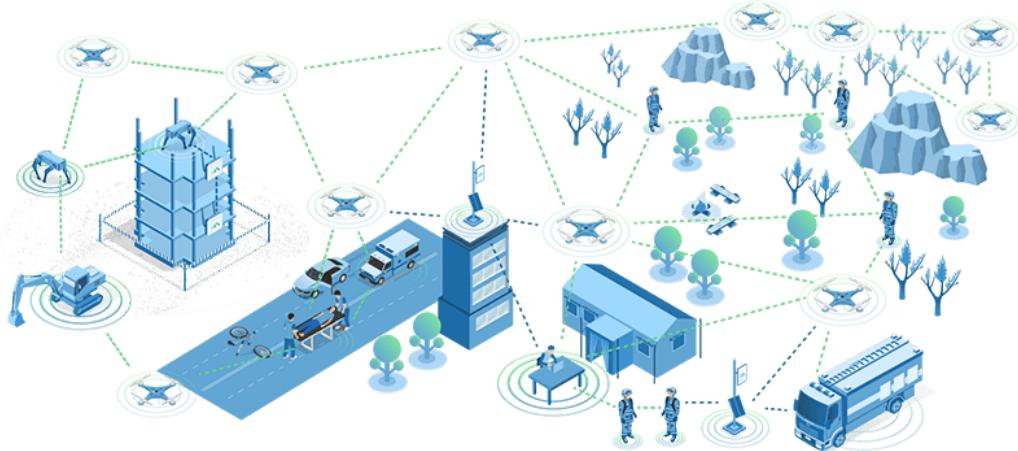


Figura 1.3: Esempio schematico di una rete UAV

Nelle reti di telecomunicazioni esistono diversi paradigmi per gestire la comunicazione tra i nodi della rete; uno degli aspetti caratterizzanti di questi paradigmi è la topologia adottata, ovvero la disposizione spaziale dei nodi e dei collegamenti fra di essi. Per quanto riguarda le reti wireless UAV si possono adottare due tipi di topologie:

- **topologia centralizzata:** ogni nodo deve avere un collegamento diretto con un elemento centrale della rete per essere connesso, ad esempio le BS. Questa tipologia pone in modo evidente un grosso limite all'esplorazione dell'area di interesse, in quanto gli agenti non potranno allontanarsi dai nodi centrali, limitando le zone esplorabili.
- **topologia decentralizzata:** non si identifica un nodo avente maggiore rilevanza, e affinché un nodo sia connesso non importa che abbia un collegamento diretto con un nodo master, ma è sufficiente che vi sia un collegamento indiretto. Questo approccio permette di disporre in maniera più ampia nella regione da esplorare e di adottare una configurazione maggiormente mutabile nel tempo.

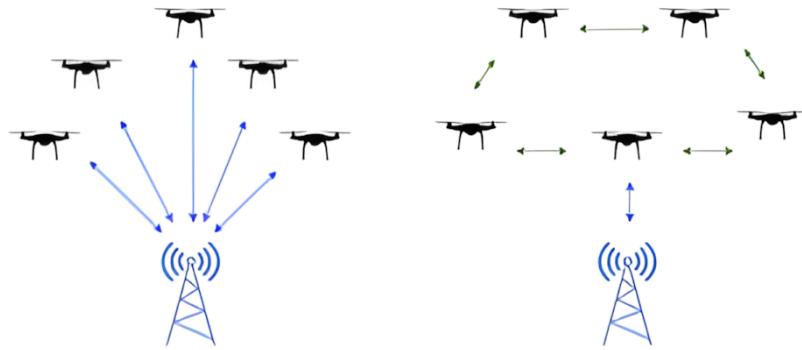


Figura 1.4: Confronto tra le due topologie di rete: a sinistra vengono mostrati schematicamente dei collegamenti tipici di un network UAV con topologia centralizzata; a destra, lo stesso sistema ma con topologia decentralizzata

Al fine di instaurare una connessione tra agenti può sembrare ragionevole utilizzare la stessa rete utilizzata per la comunicazione con gli utenti. Tale approccio tuttavia rischia di limitare l'efficienza complessiva, in quanto la tecnologia utilizzata nelle connessioni tra utenti ed agenti potrebbe non supportare comunicazioni a lunga distanza, riducendo di conseguenza la distribuzione geografica del sistema. Una strada alternativa per gestire la comunicazione tra UAV può essere quella di utilizzare una rete apposita, detta rete di **backhaul** [1][12]; questo comporta una divisione della rete complessiva in due sottoreti, in un certo grado indipendenti tra di loro. Il vantaggio di utilizzare questa divisione è quello che la rete di backhaul può sfruttare tecnologie di telecomunicazioni a più ampio raggio, come ad esempio la Free Optical Space (FOS) [13] o le comunicazioni satellitari [14] (Figura 1.5), in modo che la comunicazione tra agenti non risulti un fattore limitante per la performance del sistema.

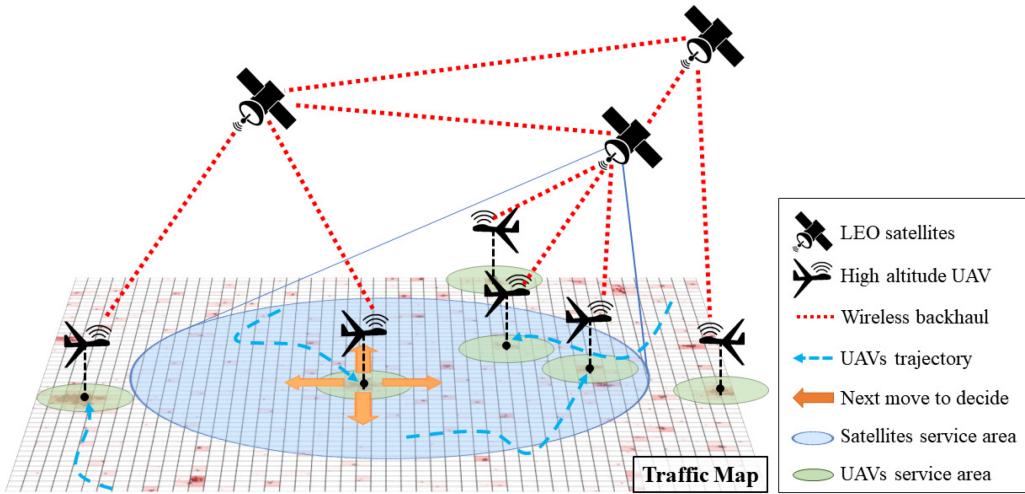


Figura 1.5: Rappresentazione schematica delle comunicazioni UAV-UAV mediante una rete backhaul satellitare.

1.3 Formulazione dei problemi

Nelle seguenti sezioni viene data una breve descrizione dei due principali problemi che verranno affrontati in questa tesi.

1.3.1 Problema della copertura

Si consideri un insieme di utenti, posizionati a terra, di cui non si conosce il numero complessivo né la posizione, a cui si deve fornire connessione tramite la rete di droni. Il problema di copertura si pone l’obiettivo di trovare una disposizione delle fonti di segnale (droni ed eventuali BS) tale che il segnale ricevuto da ciascun utente noto soddisfi una serie di requisiti.

Il controllo della copertura può ricadere sotto due categorie: statico e dinamico. Nel primo, lo scopo è quello di trovare il posizionamento ottimale degli agenti per garantire la copertura degli utenti, mentre il secondo usufruisce della mobilità delle fonti di segnale, cercando di connettere dei soggetti che possono muoversi nel tempo. In questa tesi, l’algoritmo di copertura

statico [7] utilizzato nel lavoro precedente è stato riadattato per renderlo compatibile con lo scenario in cui il numero totale di utenti non è noto.

1.3.2 Problema dell'esplorazione

Si consideri una regione in cui sono presenti alcuni elementi, ma di cui non si conoscono alcune informazioni; per esempio le posizioni degli utenti e il loro numero in un determinato istante di tempo. Ci si pone quindi il problema di come, utilizzando le stesse fonti di segnale utilizzate per il problema di copertura, esplorare le porzioni di regione che in un determinato istante non sono coperte dal segnale della rete.

Come per la copertura, in ambito scientifico si vanno ad individuare due categorie di algoritmi per l'esplorazione di un'area: algoritmi statici, in cui si determina a priori il percorso che un agente deve percorrere affinché ogni zona della regione venga esplorata almeno una volta, pensati per contesti in cui l'ambiente non varia o le cui variazioni sono sufficientemente lente, e algoritmi dinamici, in cui invece l'esplorazione di una zona potrebbe ripetersi più volte nel tempo, al seguito della variazione di alcune condizioni (Figura 1.6).

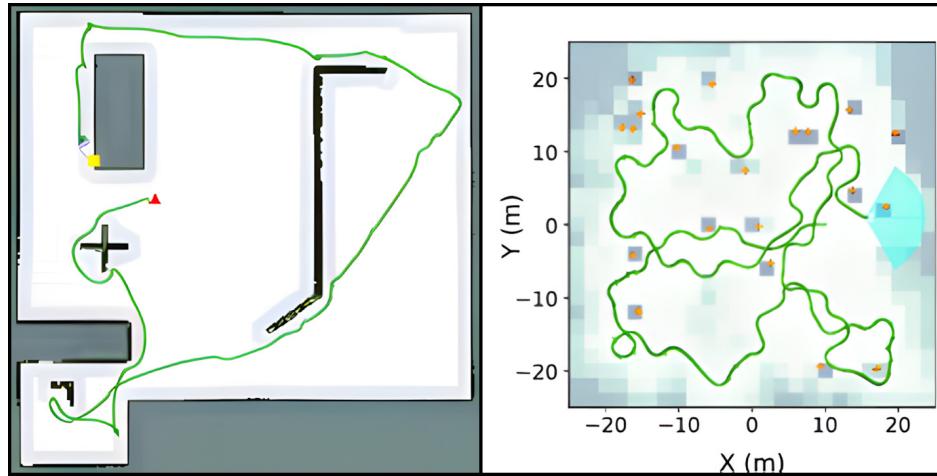


Figura 1.6: Esempi di traiettorie in un processo di esplorazione: a sinistra, in ambiente statico; a destra, in ambiente dinamico. Si osserva come l'ambiente dinamico provoca una traiettoria molto più caotica.

In questa tesi si affronterà un problema in cui l’ambiente da esplorare verrà considerato dinamico, e quindi verrà adottato un algoritmo di esplorazione da eseguire ripetutamente nel tempo. Si suppone infatti che all’interno della regione di interesse le posizioni degli utenti non siano note, e che il numero degli stessi possa variare: un nuovo utente quindi, posizionato in una zona già visitata ma non attualmente coperta, potrebbe richiedere la connessione alla rete, oppure un utente già presente e coperto potrebbe disconnettersi dalla rete.

Capitolo 2

Modello matematico

In questo capitolo verranno formulati i modelli matematici del sistema, della copertura e dell'esplorazione. Inoltre, verrà presentato lo schema dell'algoritmo di controllo utilizzato negli esperimenti che seguiranno.

2.1 Modello del sistema

Si considerino M **utenti** a cui deve essere fornita connessione alla rete, distribuiti a terra all'interno di un'area bidimensionale A , con posizioni $u_j \in \mathbb{R}^2$, $j \in \{1, \dots, M\}$. Sono presenti N **agenti** mobili UAV ed eventualmente B **Base Stations**, aventi l'obiettivo di formare una rete mobile UAV e di fornire segnale agli utenti, mentre esplorano quelle porzioni di area che non sono coperte. Si indicano con $x_i \in \mathbb{R}^3$, $i \in \{1, \dots, N\}$ le posizioni degli agenti mobili, e con $b_k \in \mathbb{R}^2$, $k \in \{1, \dots, B\}$ le posizioni delle BS. Per limitare la problematica della connessione tra agenti, si ipotizza l'impiego di una rete di backhaul che permetta di considerare il grafo di connessione tra agenti sempre connesso. D'ora in poi, per semplificare la trattazione, si userà il termine

sensore per indicare sia gli agenti mobili che le BS, in quanto entrambi hanno la capacità di fornire copertura ed esplorare.

Vengono quindi fatte le seguenti ipotesi sul sistema:

1. il numero di agenti mobili è strettamente minore del numero di utenti, così come il numero di BS, se presenti, è minore di quello degli agenti:

$$B < N < M$$

2. al fine di evitare collisioni, gli agenti sono disposti ad altitudini diverse, da cui si ha che, per un sensore i e un utente j , $d_{i,j}(t) = \sqrt{\|x_i - u_j\|^2 + h_i^2}$
3. tra sensori e utenti vengono considerati i collegamenti LoS

2.2 Modello di copertura

Nei problemi di copertura ci sono vari requisiti ed indicatori su cui basare la funzione obiettivo; in questa tesi si considera come obiettivo del problema di copertura la massimizzazione del **Region Coverage Ratio** (RCR), ossia il rapporto tra il numero di utenti coperti M_C e il numero totale di utenti M_D di cui si conosce l'esistenza:

$$RCR = \frac{M_C}{M_D} \quad (2.1)$$

Per quantificare il livello di copertura di un utente da parte di un sensore si è deciso di usare il **Signal to Interference plus Noise Ratio**, definito tra un sensore i e un utente j come:

$$\gamma_{i,j}(t) = \frac{\psi_i(t)g_{i,j}(t)}{\mu_{i,j}(t) + \beta\nu_0} \quad (2.2)$$

I termini che compaiono nell'equazione sono:

- $\psi_i(t)$ è la potenza di trasmissione del sensore i
- $g_{i,j}(t) = \frac{\rho_0}{d_{i,j}(t)}$ rappresenta il guadagno di canale al tempo t tra il sensore i e l'utente j , con ρ_0 guadagno del collegamento, $d_{i,j}(t)$ la distanza tra i due elementi
- $\mu_{i,j}(t) = \sum_{k \in N \setminus \{i\}} \psi_k(t) g_{k,j}(t)$ è la potenza di interferenza relativa al collegamento (i, j) , dovuta alle interferenze tra sensori
- β è la larghezza di banda del canale, ν_0 è la Power Spectral Density of Noise (PSDN)

Usando questo modello per il SINR, si definisce l'RCR come:

$$C(t) = \frac{\sum_{j=1}^{M_D} c_j(t)}{M_D} = \frac{\sum_{j=1}^{M_D} \mathbb{1}(\gamma_j(t) - \tau)}{M_D} \quad (2.3)$$

con τ rappresenta il valore di soglia oltre al quale un utente è considerato coperto, $\gamma_j(t) = \max_{i=1 \dots N} \{\gamma_{i,j}(t)\}$ e $\mathbb{1}(x)$ è la funzione indicatrice.

2.3 Modello di esplorazione

In questa tesi si pone come obiettivo del problema di esplorazione quello di minimizzare la probabilità media che un utente possa trovarsi in una regione non coperta dal segnale di un sensore. Per questo motivo, si divide l'area di interesse in K celle uguali, e a ciascuna cella k , se non coperta, si associa la probabilità $P_k(t)$ che al suo interno vi sia un utente. A tal proposito si definiscono:

- \mathbb{P}_D come la probabilità che un utente già connesso alla rete si disconnetta
- \mathbb{P}_B come la probabilità che un nuovo utente voglia connettersi alla rete

si definisce quindi la probabilità associata alla cella k nell'istante t come:

$$P_k(t) = \begin{cases} \bar{P}_k(t-1) \mathbb{P}_B + P_k(t-1) (1 - \mathbb{P}_D) & \text{se } k \text{ non coperta} \\ 0 & \text{altrimenti} \end{cases} \quad (2.4)$$

dove \bar{P}_k è la proprietà complementare di P_k , ovvero $\bar{P}_k = 1 - P_k$. Variando i valori di \mathbb{P}_B , \mathbb{P}_D è possibile modificare la variabilità dell'ambiente, adattandolo allo scenario reale qualora si abbiano alcune informazioni a priori su di esso. Inoltre, al fine di avere un *feedback* sull'effettiva copertura degli utenti nella fase di esplorazione, si va a porre ad 1 la probabilità di una cella qualora essa contenga un utente che, a seguito di una variazione della disposizione degli agenti, passi dallo stato coperto allo stato non coperto.

Come criterio di valutazione della copertura di una cella si è deciso di adottare lo stesso usato nel problema di copertura degli utenti; si va quindi a considerare il **SINR** calcolato nel **centro** della cella, riassumendo la probabilità in quella regione con quella del suo centro, e considerandola coperta quando $\gamma_k(t) > \tau'$, dove $\gamma_k(t) = \max_{i=1\dots N} \{\gamma_{i,k}(t)\}$ e τ' è la soglia minima richiesta.

Definendo in tal modo la probabilità associata a ciascuna cella, si definisce il **livello di esplorazione globale** dell'area A come:

$$\Pi(t) = 1 - \frac{\sum_k P_k(t)}{K} \quad (2.5)$$

in tal modo, minimizzando la probabilità media, si va ad aumentare il livello di esplorazione, trasformando quindi il problema di minimo in un problema di massimo e rendendolo compatibile con il problema di copertura.

2.4 Algoritmo di controllo

L'algoritmo di controllo ha il compito di indirizzare ciascun agente verso quelle zone che massimizzano la funzione obiettivo $R(t)$, considerando anche le posizioni degli altri agenti. In questo progetto la funzione obiettivo si compone di due contributi: l'obiettivo di copertura, che mira a massimizzare l'RCR degli utenti noti, e l'obiettivo di esplorazione, che cerca di massimizzare il livello di esplorazione (o in altri termini, di diminuire la probabilità media). Per combinare i due obiettivi, i valori di copertura ed esplorazione vengono sommati, applicando a quest'ultimo un coefficiente ρ , il quale indica la rilevanza dell'esplorazione all'interno dell'obiettivo globale:

$$R(t) = C(t) + \rho \Pi(t) \quad (2.6)$$

Schematicamente, l'algoritmo di controllo si articola nei seguenti passaggi:

1. per ogni agente, campiona `NUM_SAMPLES` punti nello spazio, secondo una certa distribuzione di probabilità, successivamente ciascuno di essi viene selezionato o scartato secondo una certa regola. Formalmente, date le posizioni $x_1 \dots x_N \in \mathbb{R}^3$ degli agenti nel tempo t , a ognuno di essi viene associato un insieme di punti $S_i \subset A$.
2. per ogni agente i , si individua tra i punti $p_i \in S_i$ quello che, se assunto come nuova posizione dell'agente, massimizzerebbe $R(t)$, ovvero $p_i^* = \arg \max_{p_i \in S_i} R(t)$, valutando quindi per ogni punto campionato $C(t)$ e $\Pi(t)$.
3. All'istante successivo $t+1$ si muove l'agente i verso il suo punto obiettivo p_i^* , aggiornando la sua posizione secondo la formula:

$$x_i(t+1) = \begin{cases} x_i(t) + \varepsilon \delta_i & \text{se } \varepsilon \|\delta_i\| < \Delta \\ x_i(t) + \Delta \frac{\delta_i}{\|\delta_i\|} & \text{altrimenti} \end{cases} \quad (2.7)$$

dove $\delta_i = p_i^* - x_i$ è la distanza tra il punto obiettivo e la posizione attuale dell'agente; $\varepsilon \in (0, 1)$ è la percentuale di spostamento che l'agente compie verso l'obiettivo; $\Delta \in \mathbb{R}$ è la massima distanza che un agente può percorrere ad ogni iterazione. Limitando la massima distanza di spostamento si evita di perturbare eccessivamente la configurazione, favorendo la coordinazione del sistema [15].

4. l'algoritmo ripete i punti precedenti per `NUM_OF_ITERATIONS` iterazioni.

La natura iterativa di questo algoritmo permette agli agenti di coordinarsi tra di loro: in ciascuna iterazione ogni agente rivaluta la propria direzione al seguito dello spostamento degli altri agenti, evitando di esplorare zone più facilmente raggiungibili da altri e migliorando la resa del sistema in termini di esplorazione e copertura utente. Affinché l'algoritmo possa funzionare correttamente si rende necessario l'utilizzo di alcune strutture dati, che verranno aggiornate a ciascuna iterazione: più precisamente, è necessaria una lista contenente gli utenti noti al tempo t , e una matrice che divida l'area in celle e associa a ciascuna di queste la relativa probabilità.

Osservando il bilanciamento dei due obiettivi all'interno di $R(t)$ durante la fase di valutazione di p_i^* per ciascun agente si nota come il livello di copertura non possa mai essere maggiore di 1, ed è minore solo nei casi in cui un agente, spostandosi, non riesca più a coprire uno o più utenti. Applicando il coefficiente ρ al valore di esplorazione si riducono le possibilità che ciò accada, evitando che si perdano completamente informazioni su alcuni utenti e favorendo quindi la copertura di quelli noti. Tuttavia, tale perdita è concessa nel caso in cui il guadagno in termini di esplorazione sia molto alto.

Capitolo 3

Implementazione

In questo capitolo viene discussa l'implementazione dei modelli presentati nel capitolo 2, evidenziando le diverse strategie adottate per affrontare alcune problematiche. In particolar modo, verrà data rilevanza alle tecniche e alle strutture dati impiegate nel calcolo dell'esplorazione, riportando le varianti utilizzate durante il processo di sperimentazione del modello, quali i vari metodi di valutazione del guadagno di esplorazione di un punto campionario da un agente (Sezione 3.3).

Il codice del sistema e delle simulazioni, così come la cronologia delle modifiche, è disponibile su GitHub nel seguente repository:

<https://github.com/leonardo-guglielmi/Multiagent-exploration-system.git>

Il linguaggio di programmazione utilizzato è **Python**, che grazie alla sua semplicità e alla vasta gamma di librerie disponibili si rende particolarmente adatto alle fasi prototipali di un progetto.

3.1 Schema del progetto

In Figura 3.1 si riporta il diagramma UML delle classi del progetto (per la sua realizzazione è stato utilizzato il tool online [draw.io](#)).

3.1.1 Classi Sensor, Agent e Base_stations

La classe **Sensor** rappresenta un sensore, astraendo le differenze tra agente e BS. Contiene le caratteristiche comuni ai due elementi del sistema, ovvero la posizione tridimensionale (x, y, z); la potenza di trasmissione **transmitting_power**; il raggio di esplorazione **exploration_radius**, implementati come attributi della classe. La classe **Agent** estende **Sensor**, aggiungendovi gli attributi propri degli agenti, ovvero **trajectory**, una lista cronologica delle posizioni occupate dall'agente, e **goal_point**, il punto obiettivo che l'agente vuole raggiungere in un certo istante di tempo. La classe **Base_station** estende anch'essa **Sensor**, ed aggiunge semplicemente un attributo booleano **interference_by_bs**: se impostato a **True**, indica che il segnale di quella BS interferisce con quello degli **Agent**. Questo flag è stato aggiunto poiché, in molti casi reali, le frequenze sui cui operano i due sensori sono diverse e non interferiscono tra di loro.

3.1.2 Classe User

La classe **User** rappresenta un utente a terra a cui deve essere fornita copertura. Gli attributi rilevanti sono la posizione (x, y); il livello di SINR **desired_coverage_level** tale che l'utente sia considerato coperto; il booleano **is_covered**, che indica se in un certo istante di tempo tale utente è coperto o meno; una lista **coverage_history** che contiene la cronologia dello stato di copertura dell'utente nel tempo.

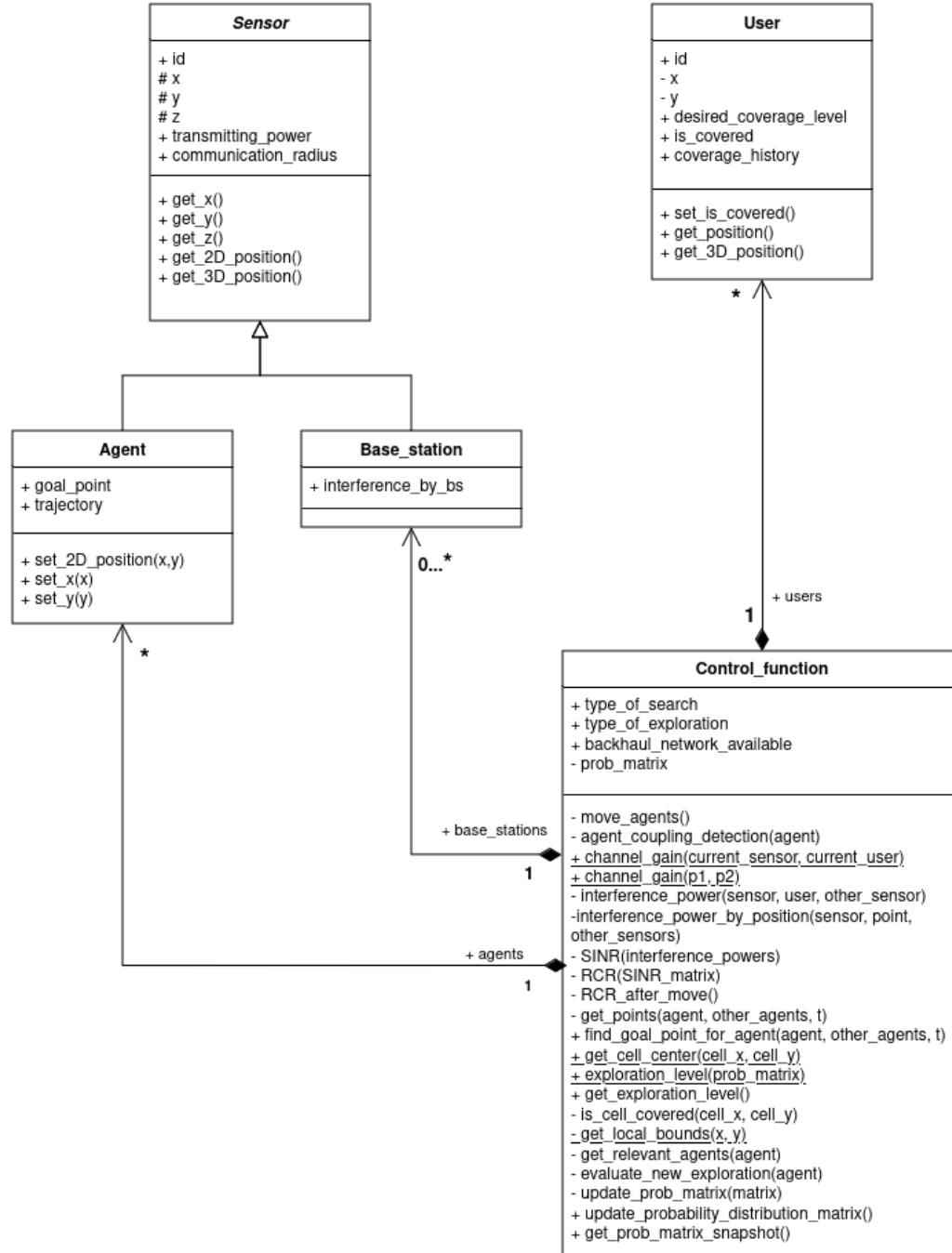


Figura 3.1: Schema UML delle classi usate nella simulazione

3.1.3 Classe Control_function

Questa classe contiene i metodi utilizzati dalla funzione di controllo. Al suo interno sono presenti i riferimenti a tutti gli attori del sistema: troviamo infatti una lista di `Agent`, una di `Base_station` e una di `User`. Inoltre, contiene tutta una serie di attributi usati durante i processi decisionali, come il criterio per la scelta tra i punti campionati, oppure la matrice di probabilità utilizzata per l'esplorazione, descritta meglio in 3.1.3.

Metodo `move_agents()`

Questa funzione, mostrata nello Snippet 3.1, esegue lo spostamento degli agenti come descritto in 2.4. Inoltre, nel caso in cui venga riconosciuta una situazione di accoppiamento tra agenti, viene applicata una deviazione; tale processo è descritto approfonditamente in 3.4.

Metodo `get_points()`

Dato un agente, questa funzione campiona un insieme di punti e ne restituisce un sottoinsieme secondo una certa regola. Più precisamente, vengono campionati `NUM_OF_SAMPLES` punti secondo una **distribuzione normale**, centrata nell'attuale posizione dell'agente specificato. Le strategie di selezione dei punti adottate sono le seguenti:

1. Ricerca **systematic**: vengono considerati tutti i punti campionati.

Questa strategia ha il problema che, occasionalmente, restituisce dei punti lontani dall'agente, e se tra questi vi fosse il punto ottimo, porterebbe l'agente a muoversi verso zone che, probabilmente, verrebbero esplorate prima da altri agenti, rendendo lo spostamento vano.

```

1 def move_agents(self):
2     for agent in self.agents:
3         # ...
4         delta_x = agent.goal_point[0] - agent.get_x() # ...
5         delta_y = agent.goal_point[1] - agent.get_y() # ...
6         distance = math.dist(agent.goal_point, agent.get_2D_position())
7
8         # if the displacement is too big, it
9         # is limited to MAX_DISPLACEMENT
10        if EPSILON * distance < MAX_DISPLACEMENT:
11            agent.set_x(agent.get_x() + EPSILON * delta_x)
12            agent.set_y(agent.get_y() + EPSILON * delta_y)
13        else:
14            agent.set_x(
15                agent.get_x() + (MAX_DISPLACEMENT * delta_x)
16                / distance
17            )
18            agent.set_y(
19                agent.get_y() + (MAX_DISPLACEMENT * delta_y)
20                / distance
21            )
22        agent.trajectory.append(agent.get_2D_position())

```

Listing 3.1: Funzione per lo spostamento degli agenti.

2. Ricerca **local**: tra i punti campionati, vengono selezionati solo quelli che sono più vicini all'agente specificato rispetto ad altri sensori (Snippet 3.2). Questa strategia predilige quindi un controllo locale dell'area, impedendo che un sensore vada a esplorare una zona che è più facilmente esplorabile da un altro.
3. Ricerca **penalty**: come nella strategia *local*, questa tecnica favorisce quei punti prossimi all'agente. A differenza della precedente non va ad escludere quelli più vicini ad altri, bensì va ad applicare una penalità ai livelli di $C(t)$ e $\Pi(t)$ calcolati in tali punti.

```

1 # code that samples points
2 # ...
3 if self.type_of_search == "local":
4     new_points = copy.deepcopy(points)
5     for point in points:
6         for other_agent in other_agents:
7             if (math.dist(point, other_agent.get_2D_position()) <
8                 math.dist(point, agent.get_2D_position()))
9             ):
10                new_points.remove(point)
11                break
12    points = new_points
13 # ...

```

Listing 3.2: Tecnica di selezione *local*.

4. Ricerca **annealing forward**: riprendendo l'idea dell'algoritmo di ricerca locale *simulated annealing*, si va ad aggiungere alla strategia *local* una probabilità aggiuntiva, decrescente con il tempo, che il punto venga selezionato anche se è più vicino ad altri agenti. Questa tecnica cerca di evitare lo stallo in punti di massimo locale.
5. Ricerca **annealing reverse**: simile alla ricerca *annealing forward*, in questa tecnica si va ad avere una probabilità crescente nel tempo. In questo modo, con il passare del tempo, il valore atteso della lunghezza degli spostamenti aumenterà.

`find_goal_point_for_agent()`

Dato un agente, questo metodo esamina tutti i punti forniti da `get_points()`, più la posizione attuale dell'agente, cercando tra di essi l'ottimo. Per fare ciò la posizione del sensore viene temporaneamente modificata con quella del punto in esame e viene ricalcolata la funzione obiettivo nella nuova configurazione del sistema; nel caso in cui il valore ottenuto sia il migliore trovato tale

punto viene salvato come nuovo p_i^* . Poiché la funzione obiettivo, come esposto nel capitolo 2, si divide in due parti, per valutare $R(t)$ nel nuovo punto dovranno essere calcolati nuovamente l'RCR e il valore di esplorazione.

`is_cell_covered()`

Questo metodo, fornite le coordinate di una cella di esplorazione, ritorna lo stato di copertura di quest'ultima. Come indicato in 2.3, per determinare lo stato si prende come punto di riferimento il centro della cella; questo implica un'approssimazione sulla reale probabilità che in quella zona vi sia un utente, approssimazione che cresce all'aumentare della dimensione delle celle.

Matrice di distribuzione delle probabilità

Questa matrice rappresenta l'area di interesse vista dal processo di esplorazione, come riportato in 2.3. Essa associa a ciascuna cella, di forma quadrata di lati `EXPLORATION_CELL_WIDTH` e `EXPLORATION_CELL_HEIGHT`, la probabilità che al suo interno vi sia un utente non coperto.

Ad ogni iterazione la matrice di mappatura deve essere aggiornata in modo da riflettere le variazioni dello stato di copertura delle celle; a tale scopo è stata definita una procedura (Snippet 3.3) che esamina ogni cella, valuta il suo stato di esplorazione con il metodo `is_cell_covered()`, e in caso non sia coperta va ad aggiornarne la probabilità secondo la Formula 2.4. Inoltre tale funzione, nel caso in cui al seguito dello spostamento di un sensore un utente che si trova in una cella k passasse dallo stato coperto a quello non coperto, aggiorna la probabilità di quella cella, impostando $P_k = 1$; infatti al seguito del cambio di configurazione si ha la certezza che in quella zona vi sia un utente.

```
1 # probability distribution matrix definition
2 self.prob_matrix = numpy.zeros((int(AREA_WIDTH / EXPLORATION_CELL_WIDTH),
3                                 int(AREA_LENGTH / EXPLORATION_CELL_HEIGHT)))
4
5
6 # ...
7
8 def update_prob_matrix(self, matrix):
9     for i in range(matrix.shape[0]):
10         for j in range(matrix.shape[1]):
11             if self.__is_cell_covered(i, j):
12                 matrix[i, j] = 0
13
14             # ... initialization case
15
16         else:
17             matrix[i, j] = (1 - matrix[i, j]) *
18                             USER_APPEARANCE_PROBABILITY \
19                             + matrix[i, j] * (1 - USER_DISCONNECTED_PROBABILITY
20 )
21
22         for user in self.users:
23             if len(user.coverage_history) >= 2 \
24                 and not user.coverage_history[-1] \
25                 and user.coverage_history[-2]:
26                 user_x, user_y = user.get_position()
27                 cell_x = int(user_x / EXPLORATION_CELL_WIDTH)
28                 cell_y = int(user_y / EXPLORATION_CELL_HEIGHT)
29                 matrix[cell_x][cell_y] = 1
```

Listing 3.3: Metodo di aggiornamento della matrice di probabilità.

```

1 @staticmethod
2 # used to elaborate global exploration level
3 def exploration_level(prob_matrix):
4     expl = prob_matrix.size
5     for i in range(prob_matrix.shape[0]):
6         for j in range(prob_matrix.shape[1]):
7             expl -= prob_matrix[i, j]
8     return expl / prob_matrix.size

```

Listing 3.4: Funzione per il calcolo dell'esplorazione globale.

Funzione per l'esplorazione

Questa funzione (Snippet 3.4), data la matrice di probabilità, va a calcolare il livello globale di esplorazione, andando a fare la media delle probabilità di tutte le celle e trasformando il risultato in modo che, ad una probabilità complessiva minore, venga associato un livello di esplorazione maggiore, rendendolo compatibile con un problema di massimizzazione (Formula 2.5).

Questo metodo può essere usato anche per valutare l'esplorazione in un punto campionario da un agente; tuttavia nella pratica quest'ultimo esplorerà solo quelle celle vicino al `goal_point`, rendendo quindi il calcolo globale dell'esplorazione poco conveniente a causa del suo costo computazionale. Per tale scopo sono stati quindi creati dei metodi di valutazione dell'esplorazione locale, descritti approfonditamente in 3.3.

3.2 Esecuzione concorrente

In un contesto reale, ciascun agente eseguirebbe la funzione obiettivo sul proprio sistema di calcolo in modo indipendente dagli altri. Si è dunque ritenuto opportuno implementare un meccanismo di parallelizzazione per il calcolo del `goal_point` di ciascun agente, riducendo sensibilmente i tempi

```

1 # ...
2 with Manager() as manager:
3     shared_dict = manager.dict()
4     processes = [Process( target=concurrent_find_goal_point
5                           , args=(cf, ag, t, shared_dict, ))
6                   for ag in agents ]
7     for p in processes:
8         p.start()
9     for p in processes:
10        p.join()
11        p.close()
12    for ag in agents:
13        ag.goal_point = shared_dict[ag.id]
14 # ...

```

Listing 3.5: Esecuzione parallela della ricerca del `goal_point`.

di esecuzione delle simulazioni. Data la presenza nell’interprete Python del *Global Interpreter Lock*, la scelta del modulo da impiegare è ricaduta su `multiprocessing`, un package Python che espone delle API per la creazione e gestione di processi paralleli.

Per parallelizzare la simulazione, si crea un processo distinto per ciascun agente, il quale eseguirà il metodo `goal_point_agent()`. Una volta calcolato il punto ottimo, il processo lo inserisce in un dizionario condiviso, associandogli come chiave l’id del proprio agente. Una volta terminati tutti i processi agente, nel processo principale viene estratto dal dizionario ciascun `goal_point` e associato al relativo agente (Snippet 3.5).

3.3 Valutazione dell’esplorazione

Come accennato in 3.1.3, andare a valutare per ogni punto campionato l’esplorazione **globale** risulta estremamente costoso, implicando dei tempi di simulazione estremamente lunghi. Inoltre, l’uso di questo approccio è poco

utile; infatti dopo il movimento degli agenti nelle zone lontane dal punto campionario in esame avrà verosimilmente uno stato di esplorazione diverso, rendendo la previsione fatta su quelle celle durante la valutazione del punto ottimo poco attendibile. A fronte di queste problematiche, durante le fasi sperimentali sono stati progettati e implementati diversi metodi per valutare localmente il livello di esplorazione di un punto, in modo che tale operazione risulti computazionalmente valida.

3.3.1 Metodi sperimentali per la valutazione dell'esplorazione

In questa sotto-sezione si riportano i vari metodi sviluppati durante le fasi di test, ma che sono stati rimpiazzati da una versione successiva più accurata, fino ad arrivare a quello utilizzato negli esperimenti, ed esposto in 3.3.2.

Simple Exploration

Questo metodo è stato utilizzato nelle fasi iniziali per verificare la variazione di probabilità delle celle in tempi di simulazione ragionevoli. Esso non tiene conto del SINR per valutare la copertura di un cella, ma controlla soltanto che il suo centro sia abbastanza vicino ad un sensore. Per ovvi motivi, tale metodo è stato rapidamente sostituito dai successivi.

Local Square Interference Exploration (LSIE)

In questa variante, si va a valutare la copertura di una cella tramite il livello di SINR calcolato nel suo centro, rendendo il requisito di copertura della cella simile a quello della copertura utente. La novità principale è quella di non considerare tutte le celle della matrice, ma soltanto quelle che

rientrano all'interno di un'area quadrata centrata nella posizione dell'agente, di lato $2 \cdot \text{EXPLORATION_RADIUS}$ (Figura 3.2).

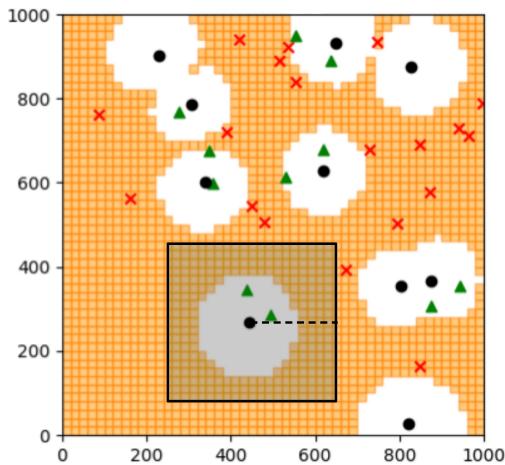


Figura 3.2: Esempio dell'area considerata dal metodo LSIE; si osserva come le celle presenti negli angoli dell'area siano difficilmente raggiungibili dal segnale dell'agente.

Inoltre si vanno a considerare soltanto le interferenze provenienti da sensori relativamente vicini all'agente, in quanto una eccessiva distanza rende il segnale d'interferenza trascurabile; questa considerazione permette di avere un minor numero di agenti nel calcolo del SINR, alleggerendo il costo computazionale. Si osserva come la scelta del valore di `EXPLORATION_RADIUS` sia critica, in quanto un raggio maggiore implica una visione più approfondita dell'ambiente da parte dell'agente, comportando però una minore velocità del metodo. **Eperimentalmente** si è osservato che ponendo `EXPLORATION_RADIUS=200` si ha un'esplorazione completa dell'area intorno ad un punto, mantenendo un buon tempo di simulazione. Il problema di questa variante è che, data la forma quadrata dell'area, si andranno a considerare anche le celle negli angoli, che molto probabilmente non saranno mai coperte dal segnale dell'agente (Figura 3.2).

Local Circle Interference Exploration (LCIE)

Rispetto al metodo precedente questa variante modifica la forma dell'area osservata, passando dall'utilizzo di un'area quadrata ad una circolare, rimuovendo quindi le celle che prima si trovavano negli angoli; questo porta ad un'ulteriore aumento di velocità del processo di valutazione, essendosi ridotto il numero di celle da osservare.

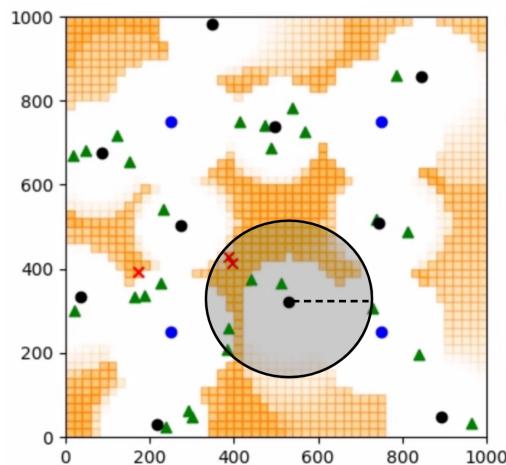


Figura 3.3: Esempio di area considerata dal metodo LCIE.

Local Square Interference Exploration, Neighbour Cell Control (LSIENCC)

Implementata prima della *Local Circle Interference Exploration*, questo metodo aggiunge una predizione sul livello di copertura di alcune celle quando sono soddisfatte alcune condizioni. Tale predizione si basa sull'osservazione che, se il SINR calcolato in una cella è abbastanza alto, allora con molta probabilità anche quello delle celle adiacenti supererà la soglia richiesta. Seguendo tale euristica, è possibile ridurre drasticamente i tempi di valutazione dell'esplorazione di quegli agenti isolati rispetto al resto della rete: infatti,

la distanza da gli altri agenti fa sì che il livello di SINR rimanga abbastanza uniforme lungo l'area.

3.3.2 Valutazione dell'esplorazione tramite frontiera con controllo delle celle adiacenti

Il metodo riportato in questa sotto-sezione è quello che verrà usato nelle simulazioni esposte nel Capitolo 4. Esso è la combinazione delle varianti *LCIE* e *LSIENCC*, esposte nella sotto-sezione precedente; tale metodo include dunque i vantaggi di avere una regione da valutare il più ridotta possibile, senza però rinunciare alla precisione della stima, e di poter predire il livello di copertura di alcune celle, senza doverne calcolare esplicitamente il valore.

I passaggi principali in cui l'algoritmo si articola sono i seguenti:

1. Per prima cosa, il metodo seleziona quelle celle che rientrano nell'area da valutare, inserendo le informazioni rilevanti (posizione del centro e probabilità) in una struttura dati simile ad una matrice, ma con lunghezza delle righe variabile (Snippet 3.6). Inoltre, identifica gli agenti abbastanza vicini al punto campionario.
2. Successivamente, per ogni cella inclusa, calcola il SINR di ciascun agente precedentemente selezionato. Dal calcolo del SINR vengono escluse quelle celle aventi $P_k = 0$, in quanto sono già coperte, e non apporterebbero nessun contributo all'esplorazione (Snippet 3.7). Se il livello di SINR di una cella supera una certa soglia `NEIGHBOUR_SINR_THRESHOLD`, la cella sovrastante e quella a destra (quella sotto e quella a sinistra sono state già esaminate) vengono inserite in `already_checked_cells`, ossia una lista che permette di escludere tali celle dalla valutazione del livello di SINR. Prima di tale inserimento, viene fatta una serie di controlli per escludere quelle celle che

```

1 # inside LCIENCC method ...
2 inf_x, inf_y, sup_x, sup_y = self.__get_local_bounds(agent.get_x()
3                                         , agent.get_y()
4                                         )
5 cells = [] # this "matrix" will contain both
6         # coordinates and probability of the cell
7 num_cells = 0
8 for i in range(inf_x, sup_x):
9     cells_column = []
10    for j in range(inf_y, sup_y):
11
12        if math.dist(self.get_cell_center(i, j), agent.get_2D_position())
13            <= agent.communication_radius:
14
15            cells_column.append(
16                {"pos": self.get_cell_center(i, j) +(0,0),
17                 "prob": self.__prob_matrix[i, j]}
18            )
19            num_cells += 1
20
21    if len(cells_column) > 0:
22        cells.append(cells_column)
23
24 relevant_agents = self.__get_relevant_agents(agent)

```

Listing 3.6: Inizializzazione del metodo LCIENCC.

eccedono l'area di valutazione, e per evitare di etichettare come coperta una cella avente probabilità zero.

3. Infine si calcola il livello di esplorazione, basandosi sulla matrice che associa a ciascuna cella il valore di SINR di ogni agente. (Snippet 3.8).

```

1 # .. iter through cells ...
2 if cells[i][j]["prob"] != 0:
3
4     # if this cell's coverage is predicted, mark it as covered
5     # and skip it
6     if cells[i][j]["pos"] in checked_cells:
7         SINR_matrix.append(1)
8     else:
9         for k in range(len(relevant_agents)):
10             SINR = (
11                 self.channel_gain_by_position(
12                     relevant_agents[k].get_3D_position()
13                     , cells[i][j]["pos"]
14                 )
15                 * relevant_agents[k].transmitting_power
16                 / (interference_powers[i][j][k] + PSDN * BANDWIDTH)
17             )
18             SINR_matrix[i][j].append(SINR)
19
20         if SINR >= NEIGHBOUR_SINR_THRESHOLD:
21             # check and mark upper cell
22             if j + 1 < len(cells[i]) \
23                 and cells[i][j + 1]["prob"] != 0:
24                 checked_cells.append(cells[i][j+1]["pos"])
25
26             # check if there exist the right column
27             if i + 1 < len(cells):
28                 # search if that cell it's inside the
29                 # right row
30                 for cell in cells[i+1]:
31                     if cell["pos"] == \
32                         self._sum_triple(
33                             cells[i][j]["pos"] ,
34                             (EXPLORATION_CELL_WIDTH, 0, 0)
35                         ):
36                         # if that cell esists, check if it's
37                         # not already covered, and mark it
38                         if cell["prob"] != 0:
39                             checked_cells.append(cell["pos"])
40                             break # exit from right cell search
41             break # exit from agents cycle

```

Listing 3.7: Calcolo della copertura nel metodo LCIENCC.

```

1 max_SINR_per_cell = [
2     [ max(SINR_matrix[i][j]) if len(SINR_matrix[i][j]) != 0
3      else 0
4
5      for j in range(len(cells[i]))
6  ]
7  for i in range(len(cells))
8 ]
9
10 for i in range(len(max_SINR_per_cell)):
11     for j in range(len(max_SINR_per_cell[i])):
12         if max_SINR_per_cell[i][j] > DESIRED_COVERAGE_LEVEL:
13             exploration_level += cells[i][j]["prob"]
14 exploration_level /= num_cells

```

Listing 3.8: Valutazione dell'esplorazione nel metodo LCIENCC.

3.4 Controllo per il disaccoppiamento di agenti

L'accoppiamento è un fenomeno causato dalla ridotta distanza tra due o più agenti, e provoca una minore capacità di copertura ed esplorazione da parte dei soggetti coinvolti. Tale problematica si verifica quando agenti molto vicini identificano ripetutamente la stessa direzione di spostamento, causando un allineamento delle loro traiettorie; questo fa sì che il loro raggio di copertura si riduca a causa delle interferenze mutue. Ciò ha come effetto quello di restringere l'orizzonte osservabile dagli agenti e conseguentemente un aumento della probabilità che gli agenti selezionino lo stesso punto ottimo.

Per cercare di limitare la durata dell'accoppiamento è stato implementata una funzione in grado di riconoscere l'evento ed intervenire, cercando di orientare gli agenti coinvolti in direzioni opposte (Snippet 3.9). L'algoritmo

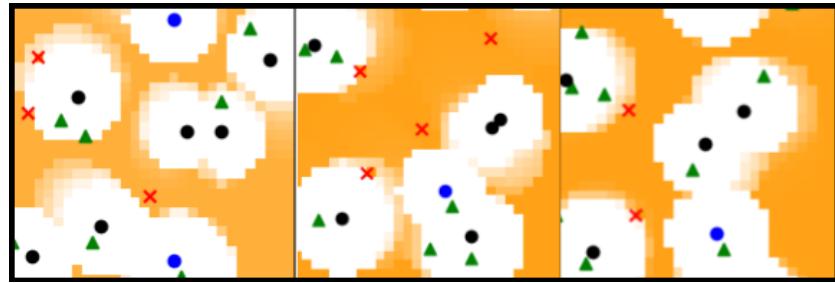


Figura 3.4: Un esempio di agenti accoppiati, rispettivamente negli istanti 15, 50 e 95. Si vede come, anche se la problematica sembra risolversi alla fine, essa ha drasticamente limitato il contributo degli agenti coinvolti.

implementato riconosce l'accoppiamento controllando ad ogni iterazione le posizioni pregresse degli agenti, confrontando le distanze mutue con un valore di soglia; nel caso questo venga raggiunto si crea una deviazione che cerca di allontanare gli agenti coinvolti.

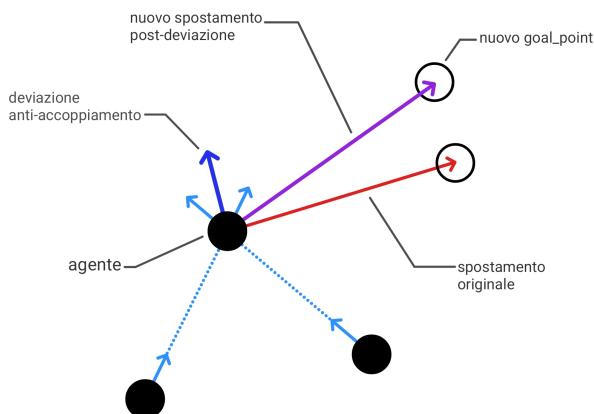


Figura 3.5: Rappresentazione schematica del sistema anti-accoppiamento

Tale deviazione viene applicata sommandola alla direzione di spostamento del punto ottimo, come se fosse presente una forza repulsiva che agisce quando degli agenti sono vicini da troppo tempo (Figura 3.5).

```

1 def agent_coupling_detection(self, agent):
2     deviation = (0,0)
3     if len(agent.trajectory) > DECOUPLING_HISTORY_DEPTH:
4         for other_agent in self.agents:
5             if other_agent != agent:
6                 distance_history = []
7                 for i in range(DECOUPLING_HISTORY_DEPTH):
8                     distance_history.append(
9                         math.dist(agent.trajectory[i]
10                                , other_agent.trajectory[i]
11                                ) )
12             if sum(distance_history) <= len(distance_history)*
13                 COUPLING_DISTANCE:
14                 deviation += (
15                     ( (agent.trajectory[0])[0]
16                       - (other_agent.trajectory[0])[0] )
17                     , ( (agent.trajectory[0])[1]
18                       - (other_agent.trajectory[0])[1] )
19                     )
20
21     return deviation
22
23
24 def move_agents(self):
25     for agent in self.agents:
26         coupling_deviation = self._agent_coupling_detection(agent)
27         delta_x = ( agent.goal_point[0]
28                     - agent.get_x()
29                     + coupling_deviation[0]
30                     )
31         delta_y = ( agent.goal_point[1]
32                     - agent.get_y()
33                     + coupling_deviation[1]
34                     )
35         # ...

```

Listing 3.9: Funzione per il riconoscimento dell'accoppiamento.

Capitolo 4

Simulazioni

Di seguito vengono descritti gli esperimenti svolti e le relative simulazioni, riportando e valutando i risultati.

4.1 Valori dei parametri

Il valore dei parametri del sistema e del modello di copertura sono stati presi da [16], in modo da avere simulazioni il più vicino possibile a uno scenario reale.

Costanti del sistema:

- NUM_OF_SAMPLES = 250
- NUM_OF_ITERATIONS = 100
- AREA_WIDTH = 1000 m
- AREA_LENGTH = 1000 m
- MIN_VERTICAL_DISTANCE = 0.15 m
- ALTITUDE = 50 m, ovvero la minima altitudine dei droni
- MIN_VERTICAL_DISTANCE = 0.15 m
- N = 10, B = 4, M = 30

Parametri del modello di copertura:

- PSND = $7.164e^{-16}$ mW/Hz, che corrisponde a - 174 dBm/Hz (ν)
- TRANSMITTING_POWER = 0.2 W (μ_i) uguale per tutti i sensori
- BANDWIDTH = 2 MHz (β)
- PATH_GAIN = $\frac{\lambda^2}{(4\pi)^2} = 0.0001$,
dove $\lambda = \frac{c}{f}$ è la lunghezza d'onda del segnale
- DESIRED_COVERAGE_LEVEL = 0.5 (τ) uguale per tutti gli utenti

Costanti di movimento:

- EPSILON = 0.1 (ε)
- MAX_DISPLACEMENT = 10 m (Δ)

Parametri del modello di esplorazione:

- EXPLORATION_CELL_WIDTH = 20 m
- EXPLORATION_CELL_HEIGHT = 20 m
- EXPLORATION_RADIUS = 200 m
- EXPLORATION_WEIGHT = 0.4 (ρ)
- USER_APPEARANCE_PROBABILITY = 0.015 (\mathbb{P}_N)
- USER_DISCONNECTION_PROBABILITY = 0.008 (\mathbb{P}_D)
- INIT_PROBABILITY = 0.5, ovvero la probabilità iniziale di ciascuna cella
- NEIGHBOUR_SINR_THRESHOLD = 0.85, ovvero il valore SINR di soglia oltre al quale considero come coperte anche le celle adiacenti

- DECOUPLING_HISTORY_DEPTH = 5, ovvero il numero di posizioni della traiettoria che considero per la verifica dell'accoppiamento
- COUPLING_DISTANCE = $3 \cdot \text{EXPLORATION_CELL_WIDTH}$, ovvero la distanza minima al di sotto del quale considero due agenti accoppiati in un certo istante

4.2 Simulazioni e analisi dei risultati

Le simulazioni sono state eseguite su una macchina Asus Vivobook S15 avente le seguenti caratteristiche: processore Intel Core i7-8565U 4 core 1.8 GHz; 8GB di RAM; SSD Kingstone da 256 GB.

Tutti i risultati degli esperimenti, come i grafici di andamento medio o le animazioni delle singole simulazioni, sono disponibili sul repository GitHub del progetto <https://github.com/leonardo-guglielmi/Multiagent-exploration-system.git>.

Negli esperimenti che seguono per ciascun tipo di scenario sono state effettuate 30 prove, ognuna con una configurazione delle posizioni iniziali degli agenti e degli utenti assegnata in modo casuale secondo una distribuzione uniforme lungo l'area di interesse.

Simulazione	Massimo	Minimo	Media	Dev. standard
Esplorazione con BS	0.97	0.73	0.87	0.0645
No esplorazione con BS	0.93	0.63	0.79	0.0756
Esplorazione senza BS	0.87	0.63	0.75	0.0677
No esplorazione senza BS	0.8	0.5	0.64	0.0848

Tabella 4.1: Primo esperimento, indici dei valori finali di copertura

4.2.1 Esperimento 1: verifica dell'efficacia dell'algoritmo di esplorazione

In questo esperimento si vuole verificare che l'algoritmo di esplorazione apporti effettivamente un miglioramento in termini di RCR totale rispetto al solo algoritmo per la copertura. Per confermare ciò sono state effettuate delle simulazioni alternando l'esecuzione del metodo di esplorazione e la presenza delle BS, per un totale di 4 scenari diversi per ciascuna simulazione eseguita, utilizzando la tecnica di campionamento *local*.

Nelle Tabelle 4.1, 4.2 si mostrano alcuni indici statistici sui valori finali di RCR ed esplorazione raggiunti, mentre in Figura 4.1, 4.2 vengono confrontati gli andamenti medi delle due grandezze durante le simulazioni. Si osserva che, quando in uso, l'algoritmo di esplorazione effettivamente porta ad avere dei valori finali di copertura migliori. Si nota invece che quando esso non viene eseguito, una volta raggiunta una certa configurazione gli agenti smettono di muoversi, in quanto l'algoritmo di controllo non permette loro di abbandonare la copertura di alcuni utenti per la ricerca di altri, limitando quindi il numero massimo di utenti raggiungibili; a fronte di questo, si ipotizza dunque che se si fosse aumentato il numero di iterazioni di ciascuna simulazione, negli scenari con esplorazione si sarebbero potuti raggiungere valori di copertura migliori.

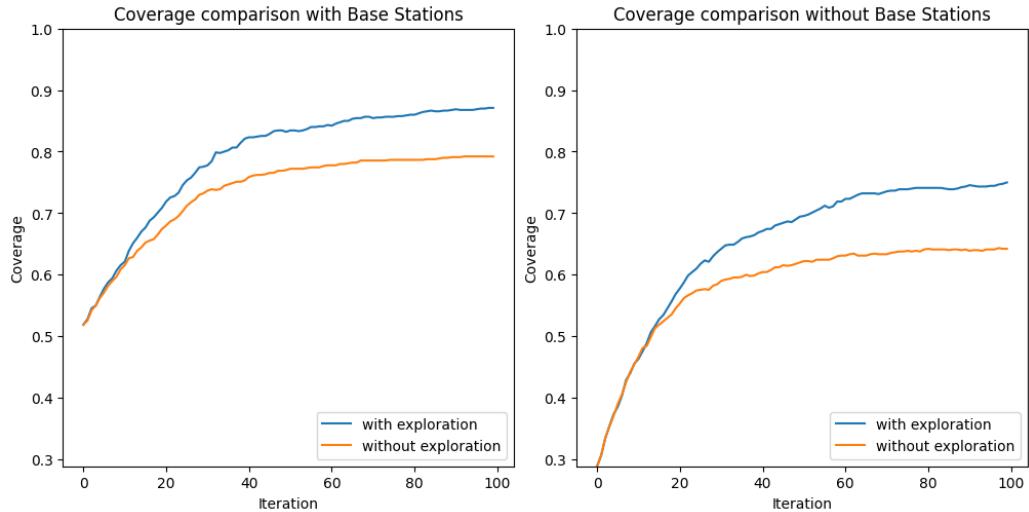


Figura 4.1: Confronto tra gli andamenti medi della copertura negli scenari di simulazione del primo esperimento.

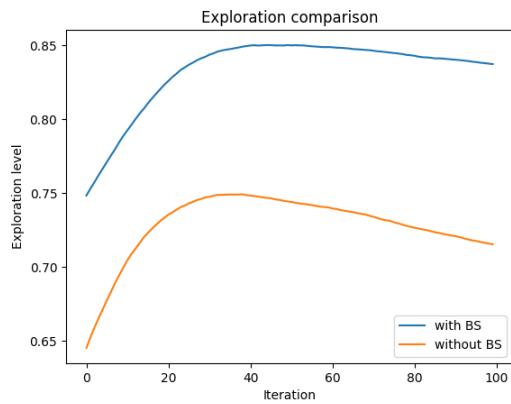


Figura 4.2: Confronto tra gli andamenti medi dell'esplorazione negli scenari di simulazione del primo esperimento.

Inoltre, per fornire una migliore interpretazione dei dati, si fa presente che nelle simulazioni senza BS, nella maggior parte dei casi, il numero di agenti risulta troppo basso per avere una qualsiasi configurazione tale da permettere la copertura di tutti gli utenti.

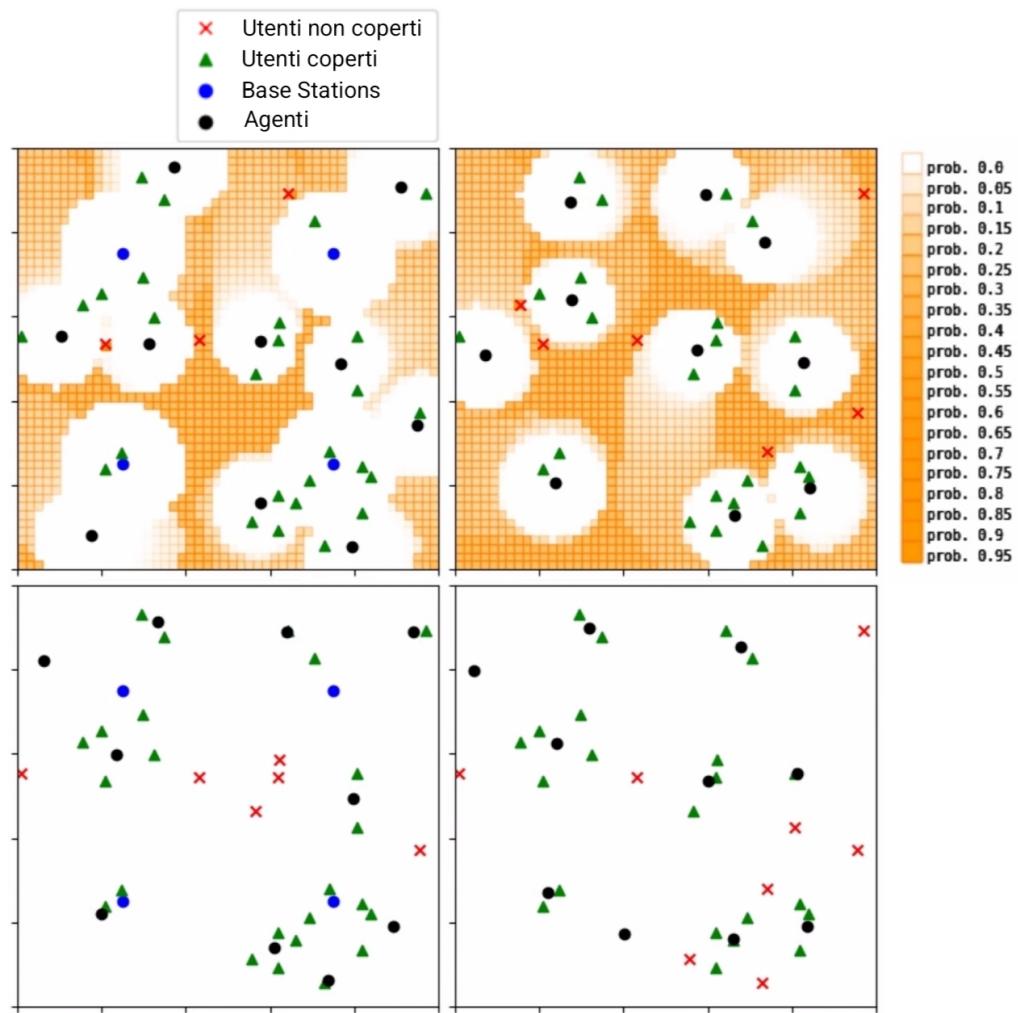


Figura 4.3: Simulazioni del primo esperimento. In alto a sinistra, lo scenario con esplorazione e con BS; in alto a destra lo scenario con esplorazione ma senza BS; in basso gli scenari senza esplorazione.

Simulazione	Massimo	Minimo	Media	Dev. standard
Con BS	0.87	0.81	0.83	0.0129
Senza BS	0.76	0.67	0.71	0.0198

Tabella 4.2: Primo esperimento, indici dei valori finali di esplorazione.

4.2.2 Esperimento 2: valutazione del metodo di campionamento migliore

Una volta verificata l’efficacia dell’algoritmo di esplorazione si vuole capire quale, tra le varie tecniche di campionamento, permette di avere dei valori di RCR ed esplorazione migliori. Data l’uniformità dei risultati precedenti, i seguenti esperimenti sono stati svolti in scenari in cui erano presenti le BS.

Osservando i valori di **esplorazione** (Figura 4.5), si constata che i metodi *local* e *annealing reverse* raggiungono dei livelli molto alti in breve tempo, per poi favorire le posizioni raggiunte dagli agenti. La tecnica *Annealing forward* invece, a fronte di un transitorio meno rapido, riesce a raggiungere valori finali più alti, come si vede anche in Tabella 4.3. Analizzando i livelli di **copertura** (Figura 4.4), si nota come i metodi *local* e *annealing reverse* risultino i più performanti, sia per i livelli finali raggiunti (Tabella 4.4) sia per la velocità di convergenza. Queste due tecniche sembrano quindi del tutto equivalenti; tuttavia analizzando i tempi di esecuzione in Tabella 4.5 si osserva che il tempo della tecnica *local* è circa del 28% minore rispetto a quello della tecnica *annealing reverse*, portando quindi a favorire la prima per la sua maggiore velocità di esecuzione.

Esplorazione	Systematic	Local	Penalty	Ann.forward	Ann.reverse
Massima	0.86	0.87	0.86	0.87	0.86
Minima	0.79	0.79	0.79	0.81	0.81
Media	0.83	0.84	0.82	0.85	0.84
Dev. standard	0.017	0.017	0.020	0.015	0.014

Tabella 4.3: Secondo esperimento, indici dei valori finali di esplorazione.

Copertura	Systematic	Local	Penalty	Ann.forward	Ann.reverse
Massima	0.97	0.97	0.93	1	0.97
Minima	0.7	0.73	0.6	0.7	0.77
Media	0.83	0.86	0.82	0.86	0.88
Dev. standard	0.067	0.060	0.087	0.072	0.056

Tabella 4.4: Secondo esperimento, indici dei valori finali di copertura.

	Systematic	Local	Penalty	Ann.forward	Ann.reverse
tempo medio	849.3	384	835.3	515	495.1

Tabella 4.5: Secondo esperimento, tempi medi di esecuzione.

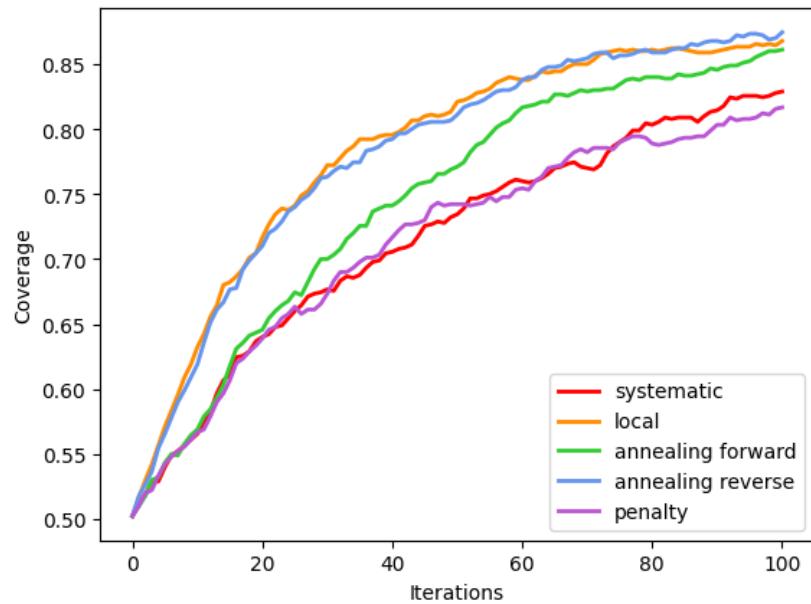


Figura 4.4: Secondo esperimento, confronto tra i valori medi di copertura utente.

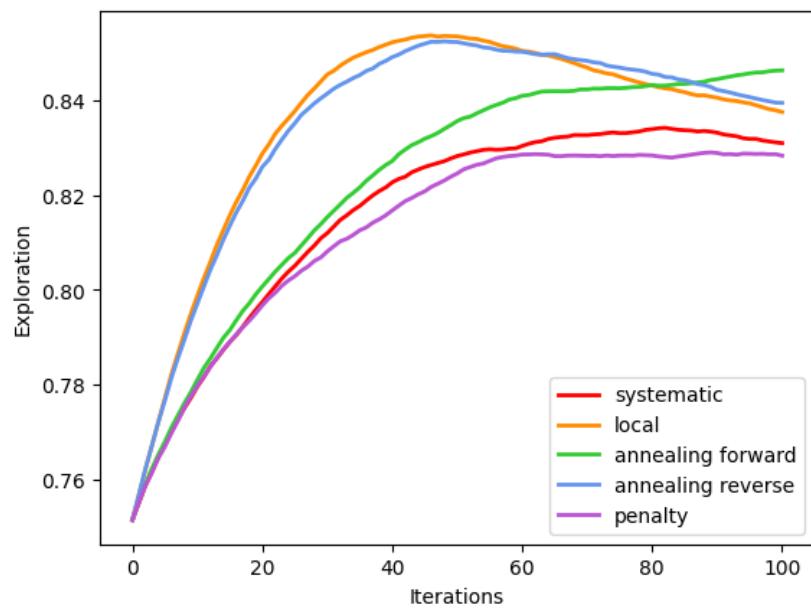


Figura 4.5: Secondo esperimento, confronto dell'andamento medio di esplorazione tra le varie tecniche di campionamento.

4.2.3 Esperimento 3: variazione della dinamicità dell’ambiente

Fino ad ora gli esperimenti effettuati hanno usato valori di \mathbb{P}_D e \mathbb{P}_B coerenti con un contesto in cui l’informazione relativa al livello di esplorazione in una regione rimane attendibile per un intervallo di tempo relativamente lungo, ovvero coerenti con ambienti aventi bassa dinamicità. Con quest’ultimo esperimento dunque si vuole verificare le performance del sistema quando opera in un ambiente maggiormente mutabile nel tempo. Dati i risultati del precedente esperimento, si è deciso di eseguire queste simulazioni usando solamente la ricerca *local*, confrontando nelle varie prove lo stesso scenario iniziale, ma variando le probabilità di apparizione e disconnessione degli utenti: in un caso si usano i valori definiti in 4.1, mentre nell’altro si usano $\mathbb{P}_D = 0.01$ e $\mathbb{P}_B = 0.07$.

Analizzando i risultati in Figura 4.7 notiamo come i livelli di **esplorazione** diminuiscano drasticamente: questo tuttavia era prevedibile, infatti con questa configurazione la probabilità che appaia un utente in una cella cresce molto rapidamente, e rende impossibile raggiungere alti livelli di esplorazione con la quantità di agenti usati nelle simulazioni. Per quanto riguarda la **copertura** (Figura 4.6) invece, si riescono a raggiungere dei livelli abbastanza buoni: benché l’algoritmo di esplorazione non impedisca di esplorare ripetutamente alcune zone, è abbastanza flessibile da guidare gli agenti attraverso l’area, disincentivando spostamenti brevi e ripetuti in favore di una traiettoria più pulita.

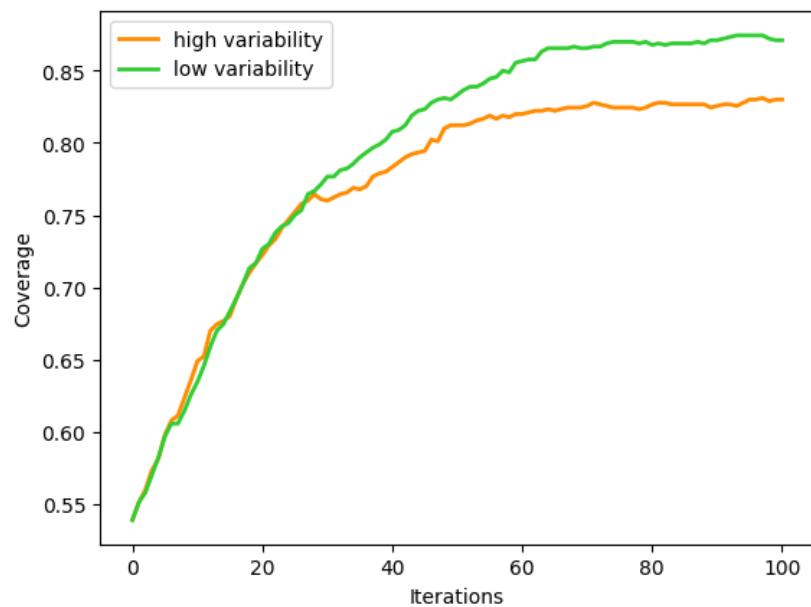


Figura 4.6: Confronto tra l’andamento medio della copertura negli scenari di simulazione del terzo esperimento.

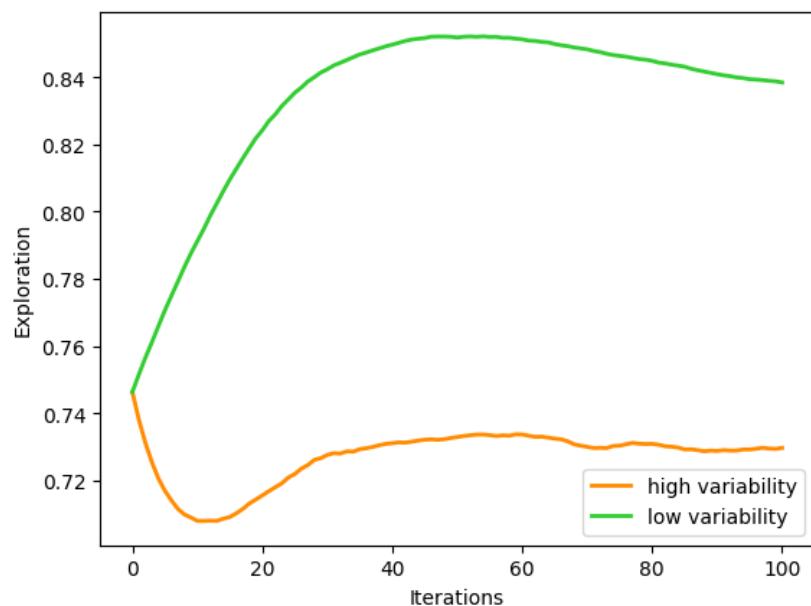


Figura 4.7: Confronto tra l’andamento medio dell’esplorazione negli scenari di simulazione del terzo esperimento.

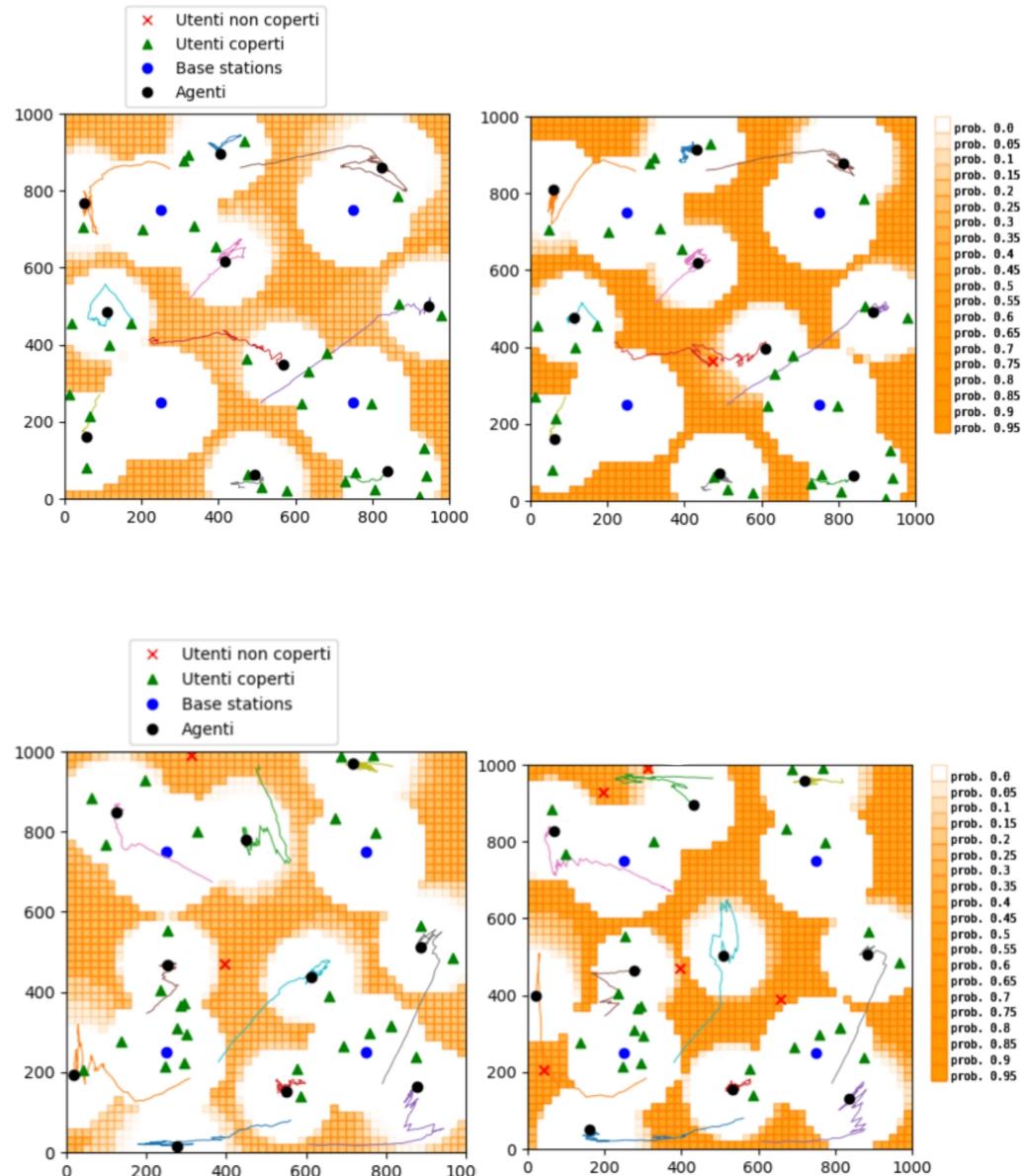


Figura 4.8: Confronti tra simulazioni con probabilità diverse. Nello scenario di simulazione in alto, nonostante il cambio di parametri le due prove hanno raggiunto livelli di copertura simili. Nello scenario in basso invece la maggiore variabilità dell’ambiente provoca una diminuzione dei risultati raggiunti.

Capitolo 5

Conclusioni

Gli esperimenti condotti hanno mostrato come, sotto le ipotesi di non conoscere il numero totale di agenti e che tale numero possa variare, l'algoritmo di esplorazione migliora i risultati ottenuti in termini di utenti coperti rispetto al sistema con il solo algoritmo di copertura. Inoltre si è verificata l'adattabilità del sistema nel caso in cui venga usato con ambienti molto mutevoli e, nonostante un effettivo calo dei risultati, permette di raggiungere dei risultati accettabili.

Il sistema esposto in questa tesi e le relative analisi coprono solo alcuni casi d'uso, e pertanto le estensioni che è possibile effettuare sono molteplici. Si potrebbe, per esempio, andare a verificarne il comportamento nel caso in cui gli utenti possano muoversi, senza però che il sistema conosca la frequenza o la lunghezza di tali spostamenti, includendo all'interno dell'algoritmo di controllo un metodo che permetta la stima di tali grandezze mediante un sistema in retroazione. Oppure si potrebbe rilassare l'ipotesi che l'altitudine sia sufficiente da trascurare eventuali ostacoli orizzontali, e quindi verificare il comportamento del sistema in ambienti più complessi, come quelli urbani, facendo eseguire ai droni spostamenti non più solo bidimensionali, ma anche

lungo l'asse verticale. Un altro aspetto non verificato da questa tesi è quello del consumo energetico, che tuttavia gioca un ruolo fondamentale nei sistemi reali e che potrebbe essere incluso nel modello bilanciando il guadagno che si ottiene a seguito di uno spostamento con il suo consumo energetico. Infine, è possibile sperimentare nuovi metodi per l'esplorazione, usando anche tecniche di machine learning, fra cui il reinforcement learning.

Bibliografia

- [1] Yue Guan et al. «Cooperative UAV Trajectory Design for Disaster Area Emergency Communications: A Multiagent PPO Method». In: 11.5 (2024), pp. 8848–8859. ISSN: 2327-4662.
- [2] Chika Yinka-Banjo e Olasupo Ajayi. «Sky Farmers: Applications of Unmanned Aerial Vehicle (UAV) in Agriculture». In: dic. 2019. ISBN: 978-1-83880-728-3. DOI: [10.5772/intechopen.89488](https://doi.org/10.5772/intechopen.89488).
- [3] Riccardo Dainelli et al. «Recent Advances in Unmanned Aerial Vehicle Forest Remote Sensing—A Systematic Review. Part I: A General Framework». In: *Forests* 12.3 (2021). ISSN: 1999-4907. DOI: [10.3390/f12030327](https://doi.org/10.3390/f12030327). URL: <https://www.mdpi.com/1999-4907/12/3/327>.
- [4] Rachna Jain et al. «Towards a Smarter Surveillance Solution: The Convergence of Smart City and Energy Efficient Unmanned Aerial Vehicle Technologies». eng. In: *Development and Future of Internet of Drones (IoD): Insights, Trends and Road Ahead* (2021), pp. 109–140.
- [5] Azade Fotouhi et al. «Survey on UAV Cellular Communications: Practical Aspects, Standardization Advancements, Regulation, and Security Challenges». In: *IEEE Communications Surveys & Tutorials* 21.4 (2019), pp. 3417–3442. DOI: [10.1109/COMST.2019.2906228](https://doi.org/10.1109/COMST.2019.2906228).

- [6] David St-Onge et al. «Planetary Exploration With Robot Teams: Implementing Higher Autonomy With Swarm Intelligence». eng. In: *IEEE robotics & automation magazine* 27.2 (2020), pp. 159–168. ISSN: 1070-9932.
- [7] Innocenti Andrea. *Controllo della copertura per sistemi di comunicazione assistiti da UAV*. 2024.
- [8] Adam Schroeder et al. «Efficient spatial coverage by a robot swarm based on an ant foraging model and the Lévy distribution». eng. In: 11.1 (2017), pp. 39–69. ISSN: 1935-3812.
- [9] Michael J. 1966- Wooldridge e Michael Wooldridge. *An introduction to multiagent systems / Michael Wooldridge*. eng. Chichester: Wiley & Sons, 2002. ISBN: 047149691X.
- [10] Stuart J Russell. *Artificial intelligence : a modern approach / Stuart J. Russell and Peter Norvig ; contributing writers: Ming-Wei Chang ... [et al.]* eng. 4. ed. AI. Hoboken: Pearson Education, 2022. ISBN: 0134610997.
- [11] Shumaila Javaid et al. «Communication and Control in Collaborative UAVs: Recent Advances and Future Trends». eng. In: *IEEE transactions on intelligent transportation systems* 24.6 (2023), pp. 1–21. ISSN: 1524-9050.
- [12] Mohamed Alzenad et al. «FSO-based Vertical Backhaul/Fronthaul Framework for 5G+ Wireless Networks». eng. In: (2016).
- [13] Nirwan Ansari et al. «SoarNet». In: *IEEE Wireless Communications* 26.6 (2019), pp. 37–43. DOI: [10.1109/MWC.001.1900126](https://doi.org/10.1109/MWC.001.1900126).

- [14] Nozhan Hosseini et al. «UAV Command and Control, Navigation and Surveillance: A Review of Potential 5G and Satellite Systems». In: (2020).
- [15] Bao Pang et al. «Effect of random walk methods on searching efficiency in swarm robots for area exploration». eng. In: *Applied intelligence (Dordrecht, Netherlands)* 51.7 (2021), pp. 5189–5199. ISSN: 0924-669X.
- [16] Forti Nicola. *Coverage control for uav-assisted communication systems*. 2024.

Elenco delle figure

1.1	Schema di comportamento di una formica.	2
1.2	Esempi di dispositivi UAV	4
1.3	Esempio schematico di una rete UAV	5
1.4	Esempi di topologie di rete	6
1.5	Comunicazioni UAV-to-UAV con backhaul satellitare	7
1.6	Esempi di traiettorie in ambiente statico e dinamico	9
3.1	Schema UML	18
3.2	Esempio dell'area valutata con il metodo LSIE	27
3.3	Esempio dell'area valutata con il metodo LCIE	28
3.4	Esempio di accoppiamento tra agenti	33
3.5	Rappresentazione schematica del sistema anti-accoppiamento	33
4.1	Grafici di copertura nel primo esperimento	39
4.2	Grafici di esplorazione nel primo esperimento	39
4.3	Simulazioni del primo esperimento	40
4.4	Grafici di copertura nel secondo esperimento	43
4.5	Grafici di esplorazione nel secondo esperimento	43
4.6	Grafici di copertura nel terzo esperimento	45
4.7	Grafici di esplorazione nel terzo esperimento	45
4.8	Confronti tra simulazioni con probabilità diverse	46

Elenco delle tabelle

4.1	Primo esperimento, indici dei valori finali di copertura	38
4.2	Primo esperimento, indici dei valori finali di esplorazione.	41
4.3	Secondo esperimento, indici dei valori finali di esplorazione. . .	42
4.4	Secondo esperimento, indici dei valori finali di copertura.	42
4.5	Secondo esperimento, tempi medi di esecuzione.	42

Listings

3.1	Funzione per lo spostamento degli agenti.	20
3.2	Tecnica di selezione <i>local</i>	21
3.3	Metodo di aggiornamento della matrice di probabilità.	23
3.4	Funzione per il calcolo dell'esplorazione globale.	24
3.5	Esecuzione parallela della ricerca del <i>goal_point</i>	25
3.6	Inizializzazione del metodo LCIENCC.	30
3.7	Calcolo della copertura nel metodo LCIENCC.	31
3.8	Valutazione dell'esplorazione nel metodo LCIENCC.	32
3.9	Funzione per il riconoscimento dell'accoppiamento.	34

Ringraziamenti