Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

Tesi di Laurea Triennale in Ingegneria Informatica

Controllo della copertura per sistemi di comunicazione assistiti da UAV

Candidato
Andrea Innocenti

Relatore
Prof. Giorgio Battistelli

Correlatore
Nicola Forti

Anno Accademico 2022/2023

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli snippet

Elenco degli Acronimi

UAV Unmanned Aerial Vehicle

LoS Line of Sight

RCR Region Coverage Ratio

SINR Signal to Interference plus Noise Ratio

PSDN Power Spectral Density of Noise

Introduzione

Gli aereomobili a pilotaggio remoto, meglio conosciuti come Unmanned Aerial Vehicle (UAV), sono apparecchi volanti controllati a distanza, privi di pilota e di equipaggio. Nati in ambito militare per svolgere funzioni di sorveglianza e ricognizione, si sono poi diffusi con grande successo in contesto civile, grazie alla loro semplicità e versatilità di utilizzo, con gli usi più disparati, come la mappatura, le consegne, la ricerca e il soccorso, la comunicazione, la fotografia, ma anche l'uso hobbystico dei droni giocattolo.
In particolare, gli UAV possono essere usati come **nodi di una rete wireless aerea**, in grado di fornire segnale ad utenti all'interno di una determinata area.

L'obiettivo di questa tesi è quello di sviluppare un algoritmo di controllo cooperativo della posizione degli UAV nello spazio, in funzione della domanda di segnale degli utenti; un problema di questo tipo richiede che gli UAV comunichino tramite la rete, la quale permette loro di scambiarsi dati circa la propria posizione, quella dei punti di interesse, lo stato dell'ambiente circostante, ecc...

In letteratura, il controllo della copertura rientra tipicamente in due categorie principali: statico e dinamico. Il primo ha come obiettivo quello di determinare il posizionamento ottimale degli agenti con il fine di garantire

la piena copertura delle aree di interesse. D'altra parte, il controllo della copertura dinamico si concentra sullo sfruttamento della mobilità degli agenti per monitorare in modo continuo ed efficace le aree di interesse tempo. In questo lavoro è proposto un algoritmo di controllo della copertura statico, che può risultare utile in vari contesti reali. Si pensi ad esempio ad un grande spazio vuoto privo in infrastrutture fisse, che viene usato occasionalmente per grandi eventi, come concerti, in cui si riunisce un gran numero di persone che necessitano di una connessione internet; o ancora si immagini una situazione di emergenza, come un terremoto o un disastro naturale, in cui le infrastrutture di comunicazione tradizionali sono danneggiate o interrotte, in cui le squadre di soccorso, i medici e gli operatori di emergenza devono comunicare tra loro per coordinare gli sforzi di soccorso.
Grazie ad un algoritmo di controllo gli UAV possono essere dispiegati in modo ottimale per creare una rete di comunicazione temporanea, agendo come ripetitori mobili e permettendo così la fruizione del servizio.

La trattazione è organizzata come segue: nel **capitolo 1** si illustrano i vantaggi dell'uso degli UAV come strumento di copertura di segnale, oltre d una panoramica sulle possibili topologie di rete. Nel **capitolo 2** viene presentata una formalizzazione rigorosa del problema ed i modelli matematici utilizzati. Nel **capitolo 3** si mostra l'implementazione dell'algoritmo e la struttura del codice; inoltre si entra nel dettaglio delle strategie utilizzate. Nel **capitolo 4** si illustrano ed analizzano i risultati dell'esecuzione dell'algoritmo, sviluppato in Python, in termini di efficacia e prestazioni. Nell' **ultimo capitolo (5)** si traggono le conclusioni, analizzando i successi ottenuti e i possibili miglioramenti futuri.

Capitolo 1

Reti di UAV

1.1 Cosa sono e perchè usarle

Una rete di UAV (UAV network) è un'insieme di **droni**, posti ad una certa altitudine, in grado di comunicare tra loro e con le **Base Stations**, stazioni fisse a terra. Una rete di questo tipo, se resa stabile, ovvero in grando di mantenere la comunicazione con continuità, veloce e sicura, risulta molto utile in vari contesti, in particolare come **rete wireless**, nella quale i droni (e in alcuni casi anche la Base Stations) svolgono la funzione di **ripetitori di segnale** che gli utenti in quell'area possono utilizzare per esempio per connettersi a internet (Fig. 1.1).
L'utilizzo di questo tipo di rete ha vari vantaggi, perchè i droni possono:

- regolare la propria altitudine, così da evitare ostacoli ed aumentare la probabilità di stabilire collegamenti in linea d'aria, in inglese Line of Sight (LoS), con gli utenti di terra;
- posizionarsi in luoghi difficili o troppo pericolosi per essere raggiunti con mezzi tradizionali;

- essere dispiegati rapidamente in aree specifiche prive di infrastrutture fisse; ciò può essere utile o in contesti emergenziali in cui le Base Stations sono andate distrutte, ma anche semplicemente in aree che occasionalmente ospitano un numero elevato di utenti, per cui non sarebbe conveniente dal punto di vista economico mantenere infrastrutture fisse. [3]

In generale reti di questo tipo sono molto flessibili e scalabili, infatti il numero UAV coinvolti può essere modificato in modo dinamico e in base al caso d'uso.

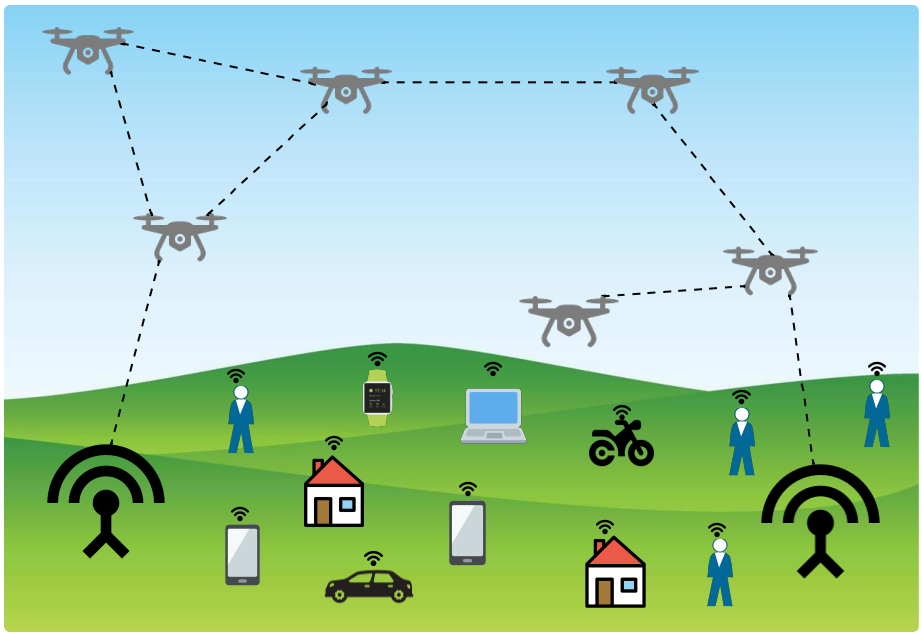

Figura 1.1: Una rete di UAV e Base Stations con utenti di vario tipo

1.2 Topologie di rete

Nelle reti di telecomunicazioni esistono diversi modi di gestire la comunicazione tra i nodi della rete, ognuno dei quali si basa su una topologia, ovvero una specifica disposizione spaziale dei nodi e dei collegamenti fra essi. Nel caso delle reti wireless di UAV parliamo di topologia centralizzata e decentralizzata. [7]

- **Topologia centralizzata**: ogni nodo ha un unico collegamento diretto con una Base Station che svolge il ruolo di tramite per la comunicazione tra UAV. Il problema principale di questa topologia è che limita l'estensione della rete al raggio di copertura di segnale della Base Station, oltre al fatto che un suo guasto interrompe qualsiasi collegamento nella rete (Fig. 1.2).

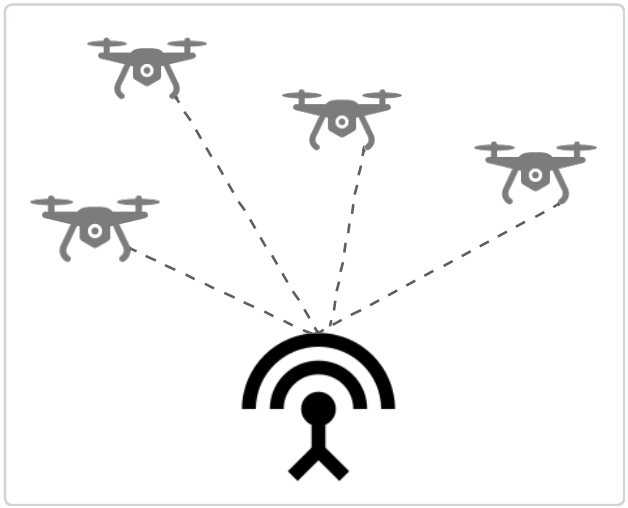

Figura 1.2: Topologia centralizzata

- **Topologia decentralizzata**: non tutti i nodi hanno un collegamento diretto con la Base Station, alcuni comunicano direttamente solo con i

nodi vicini che fanno da intermediari con la stazione a terra. Il vantaggio in questo caso è che l'estensione della rete aumenta notevolmente e di fatto è sufficiente che ogni nodo abbia un collegamento attivo per risultare connesso alla rete. (Fig. 1.3).

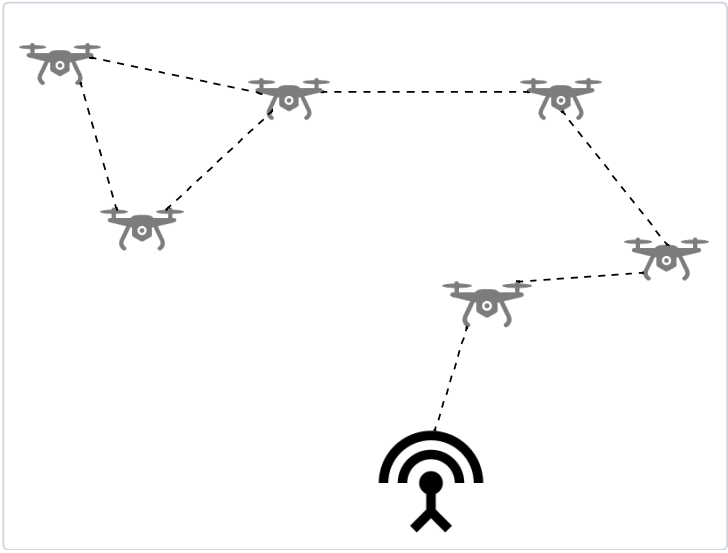

Figura 1.3: Topologia decentralizzata

Capitolo 2

Modello matematico

In questo capitolo è proposta una formalizzazione del problema, prima con un modello matematico del sistema, poi con un medello di copertura del segnale. Infine, si discute schematicamente l'idea dell'algoritmo di controllo, fornendone una formulazione matematica.

2.1 Modello matematico del sistema

Consideriamo M utenti distribuiti a terra in un'area bidimensionale A ed indichiamo con $u_j \in \mathbb{R}^2, j \in \{1, ..., M\}$ le posizioni degli utenti. Un gruppo di N agenti mobili (UAV) ha il compito di fornire segnale agli utenti ed indichiamo la loro posizione con $x_i \in \mathbb{R}^3, i \in \{1, ..., N\}$. Sono presenti anche B Base Station fisse a terra in grado di fornire segnale e di comunicare. Indichiamo la loro posizione con $b_k \in \mathbb{R}^2, k \in \{1, ..., B\}$. D'ora in poi parleremo genericamente di sensore per riferirsi indifferentemente ad un agente o una Base Station, in quanto entrambi hanno capacità di copertura e comunicazione.

Ora facciamo le seguenti assunzioni:

1. Il numero di agenti è inferiore rispetto a quello degli utenti ed il numero di Base Station è inferiore a quello degli agenti, dunque $M>N>B$.

2. Ogni agente possiede una limitata capacità di comunicazione, modellata come una sfera di raggio R; dunque dati due sensori i e j, essi riescono a comunicare direttamente solo se $d(i,j) < R$, detto **raggio di comunicazione**.

3. L'interazione tra sensori è modellata tramite un grafo indiretto $G = (V, E(t))$, dove l'insieme dei nodi V rappresenta l'insieme dei sensori, mentre l'insieme degli archi $E(t)$ rappresenta l'insieme dei collegamenti attivi al tempo t. Dati due sensori i e j, l'arco $(i,j) \in E(t)$, se al tempo t $d(i,j) < R$. Lo stato dei collegamenti varia per il fatto che gli agenti si muovono nel tempo.

4. La rete di sensori si ipotizza strutturata con topologia decentralizzata (1.2), per cui è sufficiente assicurarsi che ad ogni t il grafo risulti connesso affinchè ogni sensore conosca la posizione degli altri.

5. Gli N agenti si muovono orizzontalmente ognuno ad un'altezza diversa h_i, così da evitare gli urti. Dunque la distanza tra un UAV i ed un utente j al tempo t può essere scritta come $d^t_{i,j} = \sqrt{||x_i - u_j||^2 + h_i^2}$.

6. Consideriamo i collegamenti tra utenti e sensori in linea d'aria (LoS). Tale assunzione trascura la presenza di ostacoli ma risulta comunque una buona approssimazione se gli agenti sono ad una sufficiente altezza da terra.

2.2 Modello di copertura

Nel contesto di una rete wireless possono essere presi in considerazione requisiti di diverso tipo: garantire una comunicazione ad ampio raggio, un'equa allocazione della qualità del servizio, una comunicazione efficiente dal punto di vista energetico, ecc. Ciò si traduce nella valutazione di diversi parametri: il rapporto di copertura della regione, l'indice di equità, il consumo energetico, ecc. come parti dell'obiettivo di ottimizzazione.
In questa tesi ci concentriamo su massimizzare il Region Coverage Ratio (RCR), ovvero il rapporto tra il numero di utenti coperti M_c e il numero di utenti totali nell'area di interesse A:

$$RCR = \frac{M_c}{M}. \tag{2.1}$$

Un tipico modo per misurare il livello di copertura fornito da un sensore ad un utente è il Signal to Interference plus Noise Ratio (SINR) [2], [1]. Consideriamo un sensore i ed un utente j; grazie all'assunzione 6 possiamo definire il guadagno di canale al tempo t come $g^t_{i,j} = \frac{\rho_0}{d^t_{i,j}}$, dove ρ_0 rappresenta il guadagno del collegamento. A questo punto il SINR del canale tra i e j al tempo t può essere definito come:

$$\gamma^t_{i,j} = \frac{\psi^t_i \, g^t_{i,j}}{\mu^t_{i,j} + \beta\nu_0} \tag{2.2}$$

dove ψ^t_i è la potenza di trasmissione del sensore i, $\mu^t_{i,j} = \sum_{l \in V \setminus \{i\}} \psi^t_l \, g^t_{l,j}$ è la potenza di interferenza relativa al collegamento i, j, dovuta al fatto che i sensori, comunicando alla stessa frequenza, interferiscono tra loro; β è la larghezza di banda del canale e ν_0 è la densità spettrale di potenza del rumore, o Power Spectral Density of Noise (PSDN), valore che indica la potenza media del rumore da cui è affetto il segnale a diverse frequenze, all'interno della larghezza di banda di interesse. Tale modello rappresenta bene una

situazione reale in cui i collegamenti sono affetti da interferenza e da rumore. A questo punto il RCR al tempo t può essere scritto come:

$$c_t = \frac{\sum_{j=1}^{M} c_j^t}{M} = \frac{\sum_{j=1}^{M} \mathbb{1}(\gamma_j^t - \tau)}{M} \tag{2.3}$$

dove τ rappresenta il valore di soglia oltre il quale un utente è considerato coperto, $\gamma_j^t = max_i \left\{\gamma_{i,j}^t\right\}$ e $\mathbb{1}(x) = \begin{cases} 1 & \text{se } x > 0 \\ 0 & \text{se } x < 0 \end{cases}$ è la funzione indicatrice.

2.3 Algoritmo di controllo

Rimane ora da modellare un algoritmo che regoli il movimento degli agenti nel tempo, in modo da massimizzare il RCR. La difficoltà sta nel fatto che cercare il punto ottimo in un'area grande può risultare computazionalmente costoso; inoltre gli agenti devono riuscire a coordinare i propri movimenti in base a ciò che fanno gli altri, sfruttando la conoscenza della posizione altrui garantita dalle assunzioni 2 e 3. L'idea è la seguente:

1. Ad ogni t, per ogni agente, si individua la posizione in A che massimizza il RCR. Tale posizione viene scelta confrontando NUM OF SAMPLES punti, campionati con una certa distribuzione di probabilità. Ciò limita la ricerca dell'ottimo ad un numero fissato di punti, riducendo di molto il costo computazionale. Formalmente, siano $x_1, ..., x_N$, $x_i \in \mathbb{R}^3$ le posizioni degli agenti al tempo t e siano $S_1, ..., S_N$, con $S_i \subset A$ gli insiemi di punti campionati per l'agente i al tempo t, secondo una qualche distribuzione di probabilità. Allora per l'agente i viene scelto il punto obiettivo $p_i^* = argmax_{p \in S_i}(RCR)$.

2. Al tempo $t+1$, ogni agente si muove nella direzione del proprio punto obiettivo, aggiornando la sua posizione secondo

$$x_i = \begin{cases} x_i + \varepsilon\,\delta_i & \text{se } \varepsilon\,||\delta_i|| < \Delta \\ x_i + \Delta\,\frac{\delta_i}{||\delta_i||} & \text{altrimenti} \end{cases}, \tag{2.4}$$

dove $\varepsilon \in (0,1)$ indica la percentuale di spostamento che l'agente compie verso il punto obiettivo, $\delta_i = p_i^* - x_i$, $||\delta_i||$ è la distanza dal punto obiettivo e $\Delta \in \mathbb{R}$ il massimo spostamento che l'agente può compiere in un singolo passo. Da notare che lo spostamento avviene sempre in orizzontale, per cui la coordinata verticale non viene mai modificata.

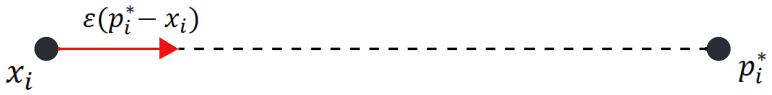

Figura 2.1: Spostamento verso il punto obiettivo

3. L'algoritmo termina dopo NUM OF ITERATIONS iterazioni oppure se trova una configuarazione degli agenti in A tale che RCR=1.

La natura iterativa di questo algoritmo fa sì che gli agenti riescano ad auto coordinarsi nel tempo, scegliendo ad ogni passo un nuovo punto obiettivo, che massimizzi il RCR in base alla nuova configurazione data dal movimento al passo precedente. L'aspetto chiave che discrimina il buon funzionamento o meno dell'algoritmo è il calcolo del punto obiettivo $p_i^* = argmax_{p \in S_i}(RCR)$, argomento discusso in 3.

Capitolo 3

Implementazione

L'algoritmo esposto in 2 è stato implementato in linguaggio Python, con il supporto dell'IDE PyCharm Professional. In questo capitolo si discuteranno i dettagli dell'algoritmo di controllo, in particolare per quanto riguarda il calcolo del punto obiettivo $p_i^* = argmax_{p \in S_i}(RCR)$, per cui verranno proposte 4 strategie. Prima saranno mostrate le scelte implementative, con il supporto del diagramma di classe UML ed alcuni snippet di codice.

3.1 Schema delle classi

In figura 3.1 è rappresentato lo schema UML delle classi del progetto, disegnato con il software Lucidchart (`https://www.lucidchart.com`).

3.1.1 class `Sensor`, `Agent`, `Base_station`

La classe `Sensor` rappresenta un sensore astratto, con una posizione tridimensionale x,y,z, un raggio di comunicazione (`communication_radius`) ed una potenza di trasmissione (`transmitting_power`). Le implementazioni concrete `Agent` e `Base_station` estendono `Sensor`; la prima aggiunge l'at-

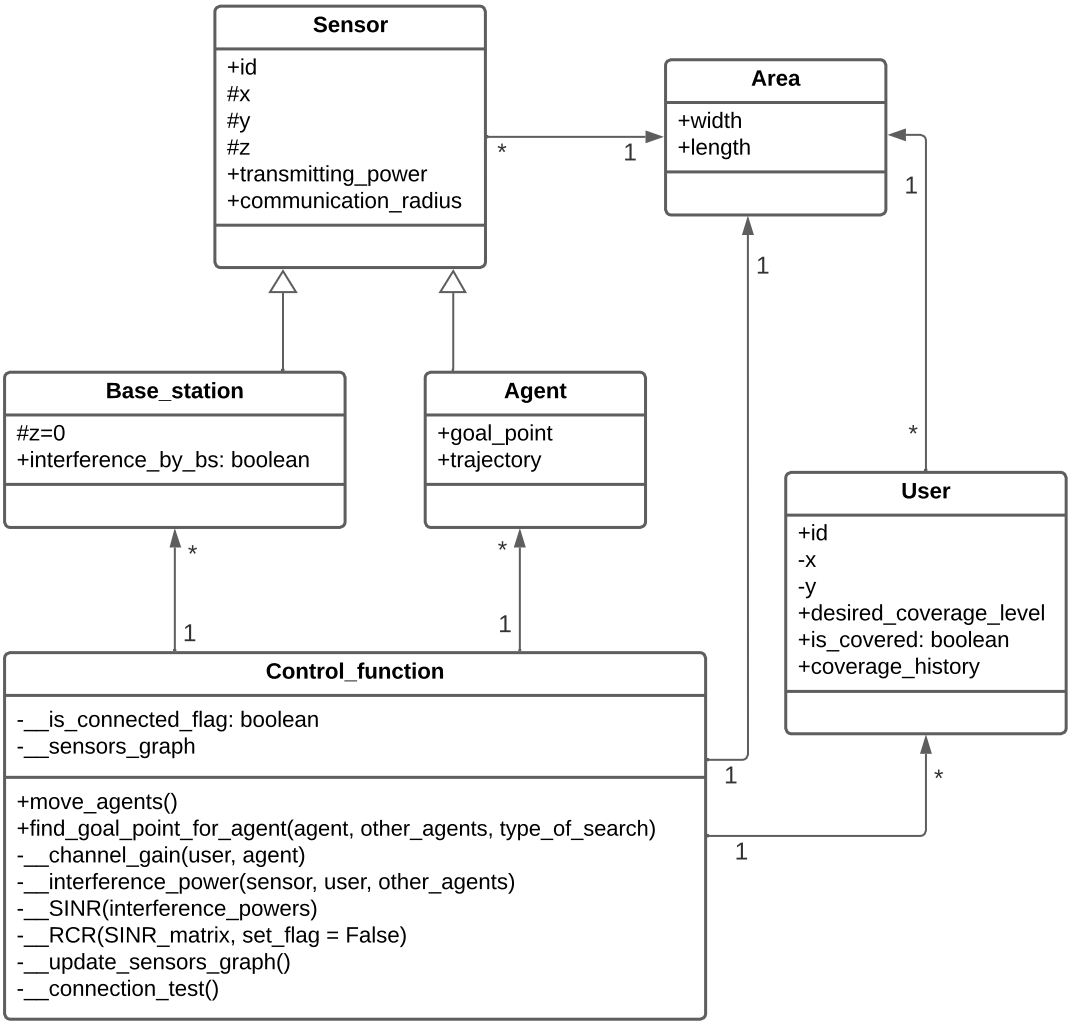

Figura 3.1: Diagramma di classe UML

tributo `goal_point`, una tupla che indica la coordinata sul piano ad altezza z posta come punto obiettivo e l'attributo `trajectory`, la lista dei punti che compongono la traiettoria totale compiuta dall'agente; la seconda fa uso invece di un booleano `interference_by_bs`, che se vero permette l'interferenza tra Base Stations ed agenti; nella maggior parte delle situazioni reali

i due tipi di sensori operano su frequenze di segnale diverse e dunque non interferiscono, per questo il valore di default della variabile è `False`.

3.1.2 class `User`

La classe `User` rappresenta un utente posto a terra (z=0), con una posizione x,y, un attributo `desired_coverage_level` che indica il livello di copertura desiderato, un booleano `is_covered`, che indica se l'utente è coperto o meno ed una lista con la storia delle coperture `coverage_history`.

3.1.3 class `Area`

La classe `Area` presenta solamente due attributi, `length` e `width`. Nelle simulazioni è stata usata un area quadrata, ma si può facilmente adattare ad una forma generica, specificando le funzioni per il caclolo delle due dimensioni.

3.1.4 class `Control_function`

Questa classe racchiude tutti i metodi utilizzati nell'algoritmo di controllo, oltre a comporre una lista di `Agent`, una di `Base_station` ed una di `User`; mantiene poi un `__sensor_graph`, ovvero il grafo dei sensori (vedi assunzione 3) implementato con una matrice di adiacenza ed aggiornato ad ogni iterazione dell'algoritmo con il metodo `__update_sensor_graph()`; un attributo booleano `__is_connected_flag` viene tenuto aggiornato sullo stato di connessione del grafo, tramite la funzione esterna alla classe `is_connected()`, analizzata in 3.1.4.

Funzione `is_connected()`

Tale funzione analizza la matrice laplaciana $L = D - G$, dove G è la matrice di adiacenza del grafo e D è la matrice di grado [5]. Da [6] sappiamo che se il secondo più piccolo autovalore di L (contando quelli multipli separatamente) è positivo, allora il grafo risulta connesso (listing 1).

Snippet 1 : is_connected()

```
1     def is_connected(graph):
2         #...inizializzazione matrice...#
3         for i in range(len(graph)):
4             for j in range(len(graph)):
5                 laplacian[i][j] = -graph[i][j]
6                 if i != j else sum(graph[i])
7         # controlla se il secondo piu' piccolo
8         # autovalore della matrice laplaciana e'
9         # positivo, se si' il grafo e' connesso.
10        return True if sorted(scipy.linalg.eigvals(laplacian)
11        )[1] > 0 else False
```

Metodo `move_agents()`

Questo metodo effettua il movimento degli agenti come analizzato nel capitolo 2; alla fine richiama `__update_sensors_graph` per ricalcolare il grafo sei sensori (listing 2).

Metodo `find_goal_point_for_agent()`

Questo metodo calcola il punto obiettivo di un agente secondo quattro diverse strategie di calcolo, specificate dalla variabile `type_of_search`,

Snippet 2 : move_agents()

```
1     def move_agents(self):
2         for agent in self.agents:
3             # viene aggiornata la posizione degli agenti.
4             # Se lo spostamento e' troppo grande si limita
5             # a MAX_DISPLACEMENT.
6             delta_x = agent.goal_point[0]-agent.get_x()
7             delta_y = agent.goal_point[1]-agent.get_y()
8             distance = math.dist(agent.goal_point,
9             agent.get_2D_position())
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(agent.get_x() +
15                (MAX_DISPLACEMENT * delta_x) / distance)
16                agent.set_y(agent.get_y() +
17                (MAX_DISPLACEMENT * delta_y) / distance)
18            agent.trajectory.append(agent.get_2D_position())
19        # ricalcola il grafo dei sensori dopo gli
20        # spostamenti.
21        self.__update_sensors_graph()
```

che si differenziano per la scelta dei punti da analizzare (vedere il metodo `get_points()`). La funzione riceve in ingresso l'agente per cui si vuole trovare il punto obiettivo (`agent`), insieme alla lista degli agenti rimanenti (`other_agents`); si itera su una lista di punti, ottenuti con il metodo `get_points()` e si modifica momentaneamente la posizione di `agent` con quella del punto corrente campionato; in questa nuova configuarazione, si calcola il RCR: se il valore è migliore di quello corrente, si aggiorna la migliore copertura finora trovata (snippet 3).

A riga 10 dello snippet 3, notiamo che la funzione che calcola il RCR riceve in ingresso una SINR_matrix, ovvero una matrice con M colonne e N+B righe i cui elementi sono i contributi $\gamma_{i,j}^t$ analizzati in 2.2; da notare che tale matrice è calcolata tenendo conto dello spostamento temporaneo di `agent` nel punto campionato. Inoltre, il RCR viene considerato nullo se lo spostamento dell'agente al punto campionato causa la disconnessione del grafo; ciò non assicura che il grafo non si disconnetta al prossimo movimento combinato degli agenti, però dato che gli spostamenti sono piccoli, si suppone che un'euristica di questo tipo basti a mantenere il grafo connesso. Nel capitolo 4 vedremo come nella simulazione è stato scelto un raggio di comunicazione sufficentemente grande ad evitare possibili disconnessioni.

Metodo `get_points()`

Dato un agente, questo metodo restituisce una lista di punti in base al tipo di ricerca specificato. Inizialmente per tutti e quattro i metodo vengono campionati NUM_OF_SAMPLES punti con distribuzione normale, centrata sulla posizione corrente dell'agente. Vediamo ora come ogni strategia tratta i punti campionati:

1. Ricerca "*systematic*": si considerano tutti i punti senza ulteriori restri-

Snippet 3 : scelta del punto obiettivo

```
    #... altro codice ...
    for point in [agent.get_2D_position()] + self.get_points(
    agent, other_agents, type_of_search, t):
            # Temporaneamente si sposta l'agente al punto
            # campionato.
            original_position = agent.get_2D_position()
            agent.set_2D_position(point[0], point[1])

            #... altro codice ...
            # Calcola il l'RCR.
            total_coverage_level = self.__RCR(SINR_matrix)

            # Ripristina la posizione originale dell'agente.
            agent.set_2D_position(original_position[0],
            original_position[1])

            # Se la copertura per questo punto e' la migliore
            # finora, aggiorna il miglior punto e la migliore
            # copertura; a parita' di copertura si sceglie il
            # punto piu' vicino.
            if total_coverage_level > best_coverage or
            total_coverage_level == best_coverage and
            math.dist(agent.get_2D_position(), point) < math.
            dist(agent.get_2D_position(), best_point):
                best_coverage = total_coverage_level
                best_point = point

        return best_point
```

zioni. Il problema di questa strategia è che nonostante la distribuzione gaussiana assicuri che la maggior parte dei punti saranno campionati vicino all'agente, ce ne sarà un certo numero lontani dalla sua posizione attuale; ora se il punto che massimizza il RCR è uno di questi, vorrà dire che l'agente inizierà a muoversi in quella direzione inutilmente, dato che se quello è un ottimo punto molto probabilmente verrà coperto in fretta da un agente più vicino. Inoltre ciò favorisce la formazione di un gruppo di agenti che vista l'ottimalità del punto, si muoveranno tutti nella stessa direzione.

2. Ricerca "*local*": l'idea è quella di scegliere solo quei punti che sono più vicini all'agente corrente rispetto agli altri agenti; in questo modo si crea una suddivisione dell'area in zone di monitoraggio che, se da una parte evita che due agenti vadano inutilmente verso lo stesso punto, di contro essi rimangono facilmente intrappolati in un massimo locale, senza possibilità di esplorare zone lontane (snippet 4).

Snippet 4 : scelta dei punti per la strategia "*local*"

```
1    #...altro codice...
2    for point in points:
3        for other_agent in other_agents:
4            delta_distance = math.dist(point, other_agent.
5            get_2D_position()) - math.dist(point, agent.
6            get_2D_position())
7            if delta_distance < 0:
8                new_points.remove(point)
9    #...altro codice...
```

3. Ricerca "*penalty*": così come nella ricerca "*local*" si favorisce una suddivisione in aree di monitoraggio, penalizzando di un valore PENALTY il RCR in quei punti che sono più vicini ad altri agenti rispetto all'agente corrente.

4. Ricerca "*annealing forward*" e "*annealing reverse*": l'idea è ripresa dall'algoritmo di ricerca locale *simulated annealing* [4]. In metallurgia, l'annealing (tempra) è il processo usato per indurire i metalli o il vetro riscaldandoli ad altissime temperature e raffreddandoli gradualmente, permettendo così al materiale di cristallizzare in uno stato a bassa energia. Per spiegare il simulated annealing, cambiamo il punto di vista dalla massimizzazione alla minimizzazione del RCR (lo stesso ragionamento sarà poi valido anche per la massimizzazione). Un punto di minimo della funzione di RCR è graficamente un avvallamento; ora immaginiamoci alle prese con il problema di far entrare una pallina da ping pong nell'avvallamento più profondo di una superficie molto corrugata (figura 3.2).

   Se lasciamo semplicemente rotolare la pallina, questa si fermerà in corrispondenza del primo avvallamento (un minimo locale). Se scuotiamo la superficie, possiamo far uscire la pallina da questo avvallamento, rischiando però di farla entrare in un altro minimo più profondo, dove starà ancora per più tempo. Il trucco è scuotere abbastanza da spostare la pallina fuori dai minimi locali, ma non così tanto da farla uscire anche dal minimo globale. La soluzione proposta dal simulated annealing è di cominciare scuotendo molto (alta temperatura) e poi ridurre gradualmente l'intensità dello scuotimento (riducendo la temperatura).

   Quest'idea è stata usata per risolvere il problema della ricerca "*local*",

in cui gli agenti rimangono facilmente "intrappolati" in un massimo locale. In "*annelaing forward*", con probabilità 1-t/NUM_OF_ITERATION si mantengono i punti lontani, favorendo inizialmente una ricerca "*systematic*"; in questo modo gli agenti escono dai massimi locali. Con il crescere delle iterazioni la probabilità diminuisce (raffreddamento), così che iniziano a ricrearsi le zone di monitoraggio della startegia "*local*". Al contrario, in "*annealing reverse*" si favorisce inizialmente una suddivisione in aree di monitoraggio, per poi con l'aumentare di t scegliere con probabilità più alta anche i punti lontani. (snippet 5).

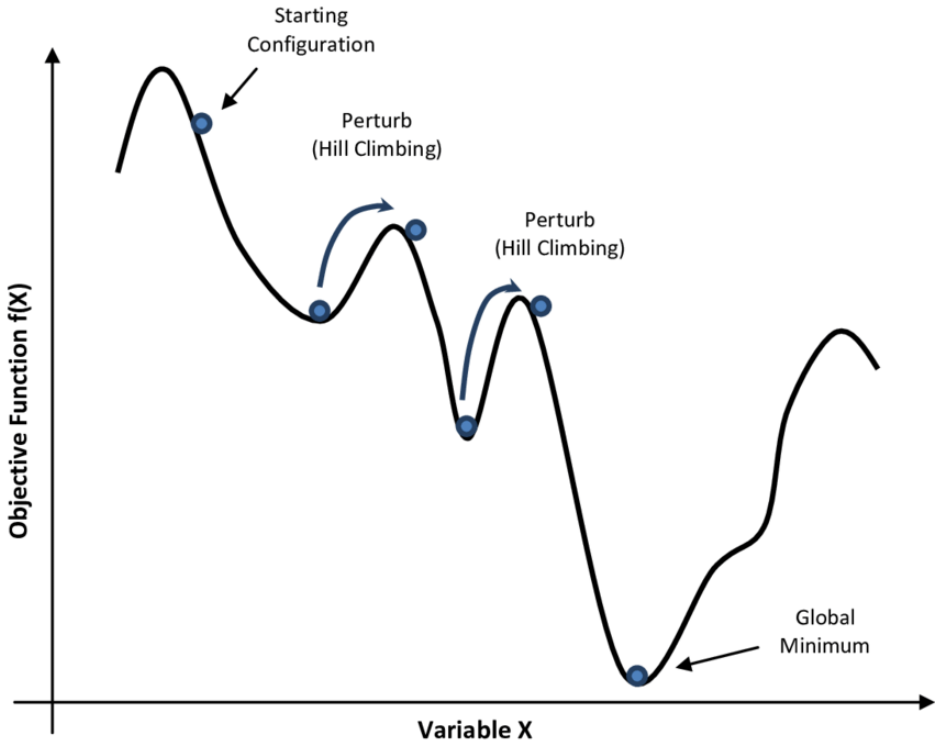

Figura 3.2: Idea intuitiva del funzionamento del "simulated annealing"

Snippet 5 : scelta dei punti per la strategia "*annealing*"

```
    #...altro codice...
    for other_agent in other_agents:
        #...altro codice...
        # man mano che t aumenta, la probabilita' di
        # rimuovere un punto lonatno aumenta in
        # annealing forward, diminuisce in reverse.
        if delta_distance < 0 and random.random() <
        (t / NUM_OF_SAMPLES if type_of_search == "annealing
        forward" else if type_of_search == "annealing reverse
        " 1 - t / NUM_OF_SAMPLES):
            new_points.remove(point)
    #...altro codice...
```

Capitolo 4

Simulazione e risultati

In questo capitolo analizzeremo i risultati sperimentali dell'esecuzione dell'algoritmo, confrontando le diverse strategie discusse in 3.1.3.

4.1 Valore delle costanti

I valori delle costanti sono stati scelti in riferimento a [1], in modo da avvicinare il modello ad una configuarazione realistica.
Costanti varie:

- NUM_OF_SAMPLES = 250
- NUM_OF_ITERATIONS = 100
- AREA_WIDTH = 1000m
- AREA_LENGTH = 1000m
- M = 30, N = 10, B = 4
- AGENT_HEIGHT = 0.15 m, l'altezza di un UAV
- MIN_VERTICAL_DISTANCE = 0.15 m, la distanza verticale tra 2 UAV
- ALTITUDE = 50 m, l'UAV i è dispiegato ad altezza ALTITUDE + i * SENSOR_HEIGHT + MIN_VERTICAL_DISTANCE
- COMMUNICATION_RADIUS = 200 m (R)

Modello di copertura:

- PSDN = 7.164E-16 mW/Hz, che corrisponde a -174 dBm/Hz (ν)
- 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
- TRANSMITTING_POWER = 0.2 W (μ_i), uguale per tutti i sensori
- DESIRED_COVERAGE_LEVEL = 0.5 (τ)

Movimento:

- EPSILON = 0.1 (ε)
- MAX_DISPLACEMENT = 10m (Δ)
- PENALTY = 1/M

4.2 Simulazione e risultati

I test sono stati effettuati su una macchina acer Aspire 5 con SO Windows 10, processore intel CORE i3 10th Gen e 8 GB di RAM. L'obiettivo dei test è quello di mettere a confronto le prestazioni dell'algoritmo con le varie strategie presentate, in termini di RCR raggiunto in funzione del tempo. Dunque è stata fatta una media su 30 esecuzioni, ciascuna delle quali con una specifica configurazione delle posizioni iniziali di agenti e utenti, scelte con distribuzione di probabilità uniforme all'interno dell'area di interesse.

In tabella 4.1, sono mostrati i risultati, evidenziando alcuni parametri statistici come la media, la deviazione standard, il minimo e il massimo valore di RCR raggiunto da ciascuna strategia, insieme al tempo di esecuzione. In figura 4.1, ancora un confronto in termini di RCR in funzione del numero di iterazioni. Notiamo che nonostante la diversità di approccio di ogni strategia, i risultati sono devvero molto simili; le differenze maggiori si hanno nel

tempo di esecuzione, dove "*systematic*" e "*penalty*" risultano essere le meno efficienti, le "*annealing*" una via di mezzo, mentre la "*local*" nettamente la migliore. Ciò è dovuto al metodo con cui vengono selezionati i punti, che vede nel caso della strategia local, la rimozione di tutti i punti lontani dall'agente corrente (nel caso delle "*annealing*" ciò è vero solo in parte), per un totale di campioni inferiore a 250 (NUM_OF_SAMPLES). Dunque la scelta dell'algortimo da utilizzare dipende da caso a caso: se la velocità non è un parametro fondamentale, meglio puntare su una delle strategie "*annealing*", che garantiscono una copertura media, seppur di poco, più alta della "*local*", altrimenti questa seconda strategia è preferibile.

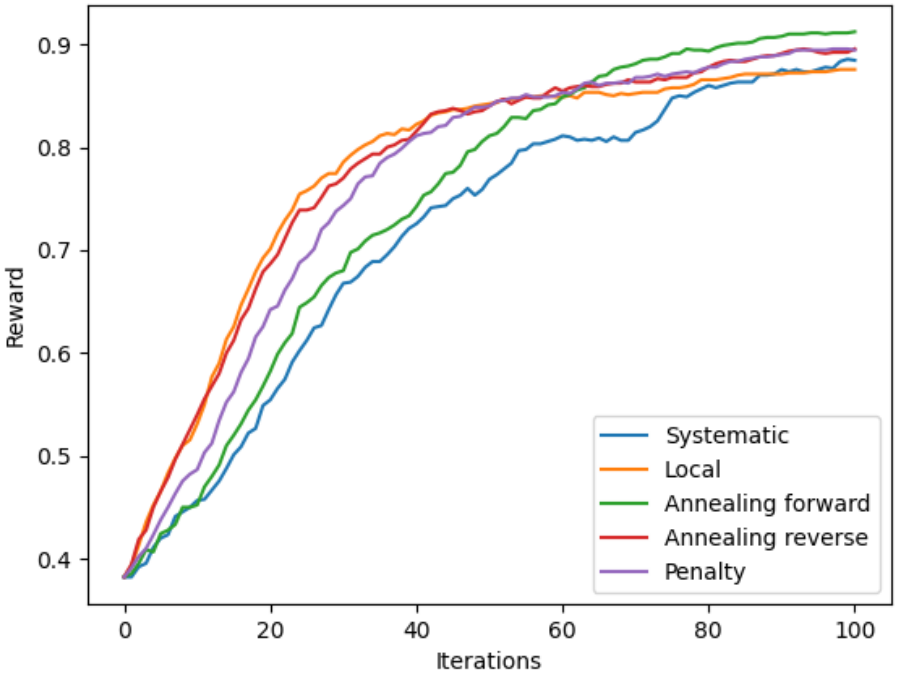

Figura 4.1: Confronto fra le varie strategie, in termini di RCR(t)

	Systematic	*Local*	*Penalty*	*Ann. forward*	*Ann. reverse*
Media	0.88	0.88	0.89	0.91	0.90
Dev. standard	0.06	0.05	0.05	0.05	0.05
Minimo	0.77	0.80	0.77	0.80	0.80
Massimo	1	1	1	1	0.97
Tempo di esec. (s)	237.3	75.6	232.5	120.7	122.0

Tabella 4.1: Risultati della simulazione

In figura 4.2 è presentato un esempio di esecuzione, sulla stessa configurazione iniziale, di un istanza di "*systematic*" e di "*annealing forward*". Notiamo come in questo caso la copertura finale premia la seconda strategia, che ottiene una copertura del 93%, contro il 77% della prima; questo risultato può essere letto notando che c'è un'alta densità di utenti nella zona centrale, fatto che favorisce un approccio più statico nella seconda metà delle iterazioni, così da evitare che due o più agenti si ostacolino a vicenda, andando nella stessa zona ad alta densità.

4.3 Codice sorgente

Il codice sorgente è disponibile alla seguente repository GitHub, dove è possibile consultare i grafici di ogni simulazione, con aggiunta di video animazione del movimento degli UAV: `https://github.com/andreainno22/thesis_project`.

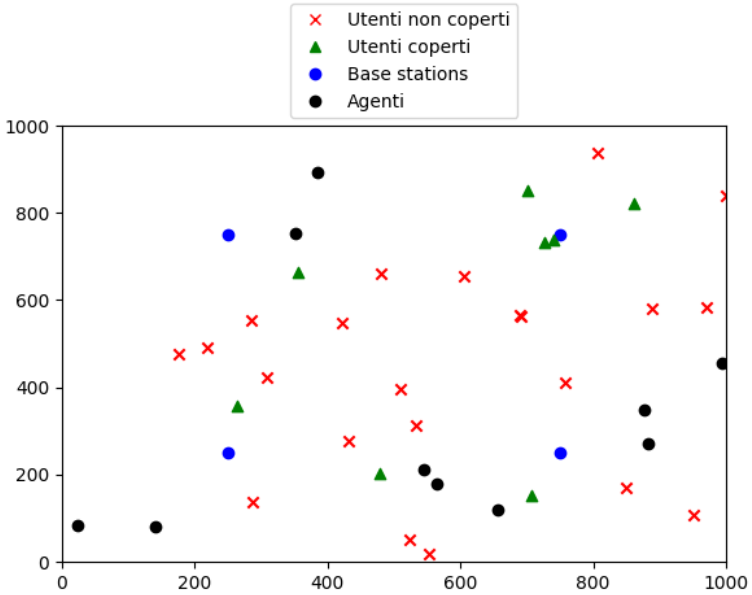

Figura 4.2: Esempio di situazione iniziale

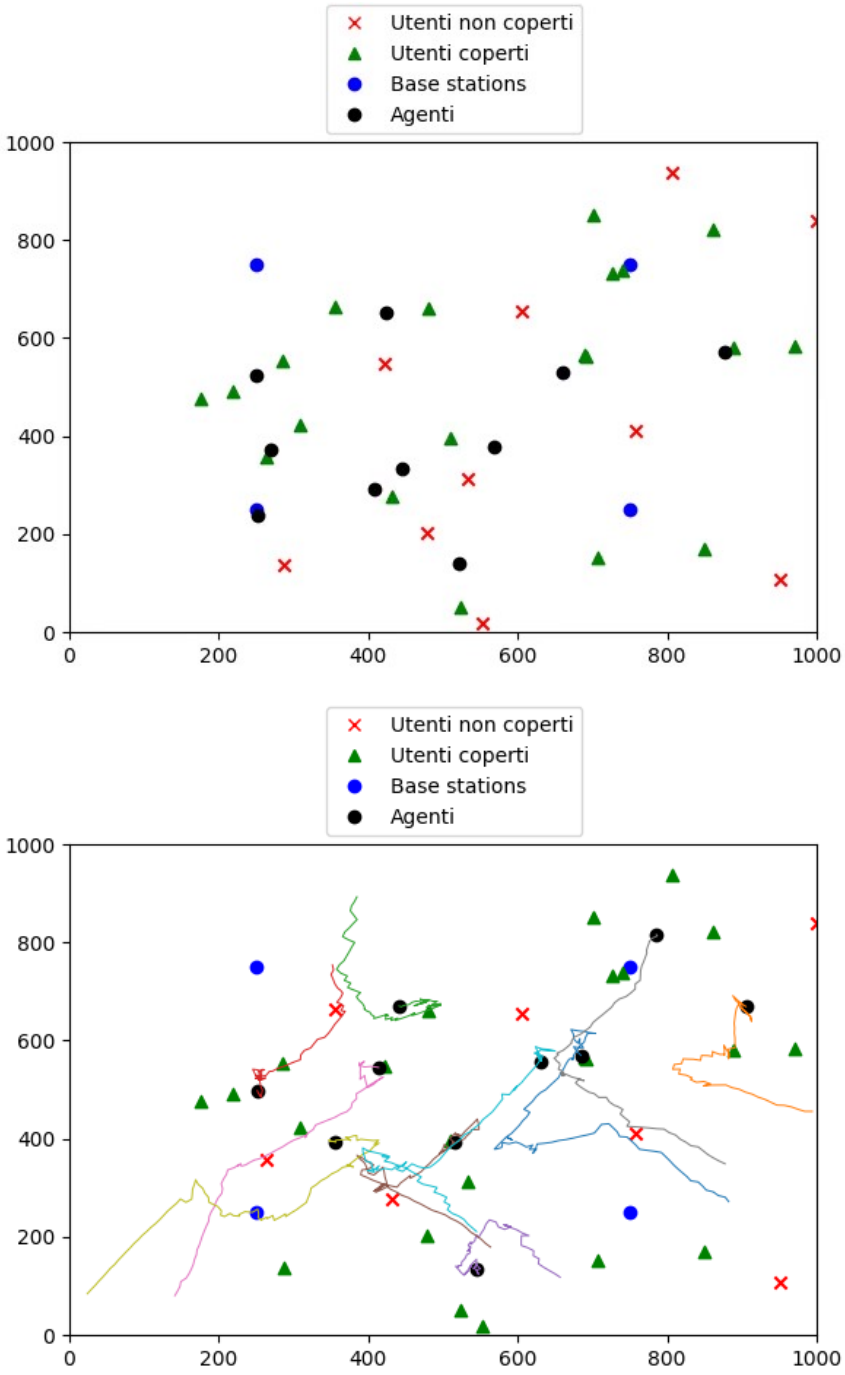

Figura 4.3: Strategia "*systematic*": esempio di situazione dopo 50 iterazioni (sopra) e finale (sotto), con traiettoria percorsa

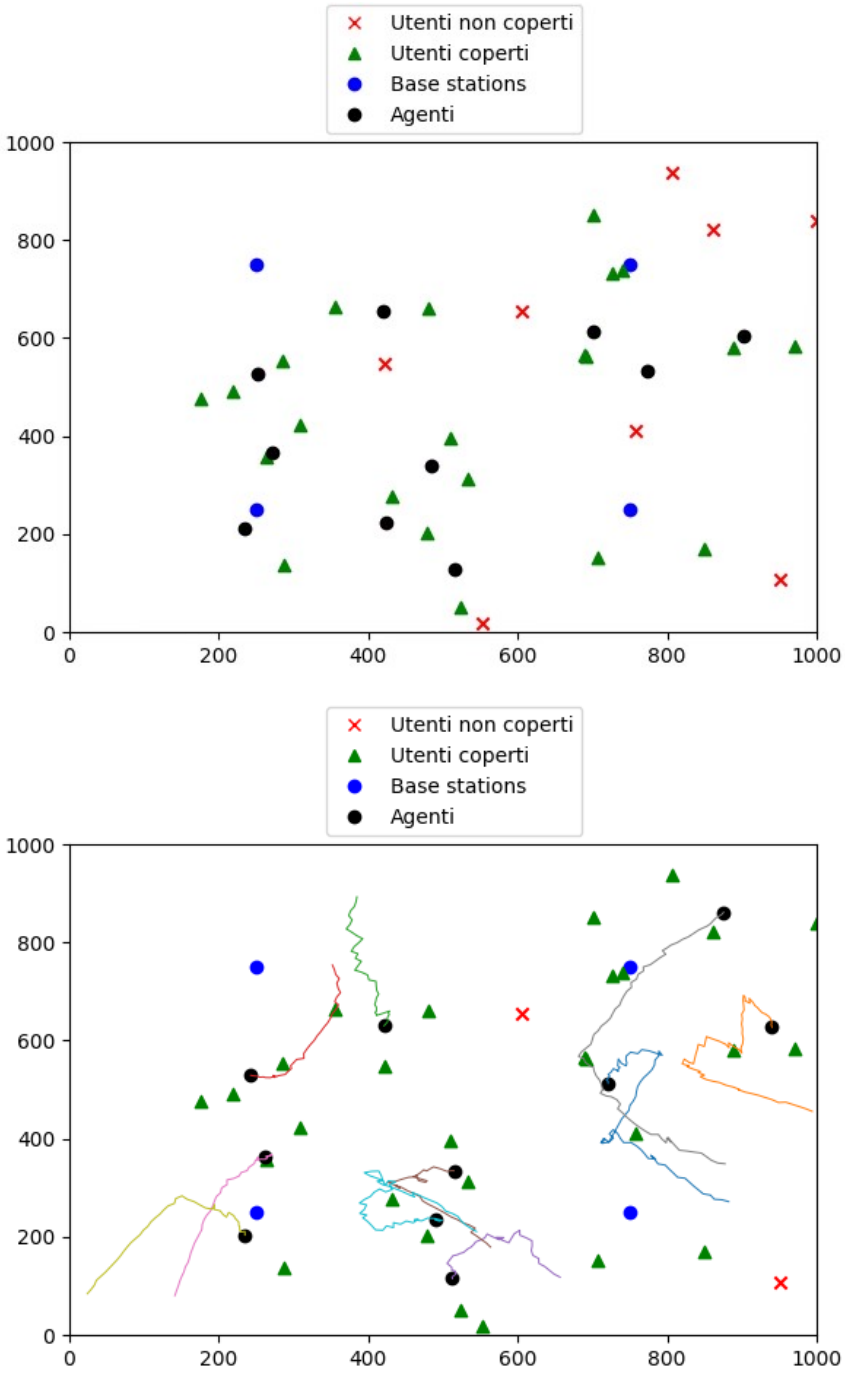

Figura 4.4: Strategia "*annealing forward*": esempio di situazione dopo 50 iterazioni (sopra) e finale (sotto), con traiettoria percorsa

Capitolo 5

Conclusioni

In questo lavoro è stata presentata una possibile soluzione al problema della copertura statica, fornendo diverse stategie di auto coordinamento degli agenti, ciascuna delle quali ha mostrato i suoi vantaggi e svantaggi. Una possibile estensione riguarda l'applicazione del modello al caso di copertura dinamica, ipotizzando dunque che gli utenti si spostino all'interno dell'area, adattando di conseguenza il movimento coordinato degli UAV. Ciò potrebbe essere fatto eseguendo l'algoritmo di controllo ricalcolando con una certa frequenza le posizioni degli utenti. Un altro aspetto interessante da esplorare è quello dell'ottimizzazione dal punto di vista energetico, aspetto senza dubbio cruciale, che intuitivamente sembra premiare strategie più statiche come la "*local*" e la "*reverse annealing*". Inoltre sarebbe interessante sviluppare una strategia efficiente per evitare completamente disconnessioni del grafo dei sensori, aspetto complesso ma allo stesso tempo importante. Volendo è poi possibile modificare il modello di copertura con strategie più sofisticate, che tengano conto del fatto che in molte situazioni reali, soprattutto in città, la massiccia presenza di ostacoli fisici riduce la probabilità di collegamenti LoS.

Ringraziamenti

Bibliografia

[1] Nicola Forti. Dynamic coverage control for uav-assisted communication systems. 2024.

[2] Yaxi Liu, Wei Huangfu, Huan Zhou, Haijun Zhang, Jiangchuan Liu, and Keping Long. Fair and energy-efficient coverage optimization for uav placement problem in the cellular network. *IEEE Transactions on Communications*, 70(6):4222–4235, 2022.

[3] Mohammad Mozaffari, Walid Saad, Mehdi Bennis, Young-Han Nam, and Mérouane Debbah. A tutorial on uavs for wireless networks: Applications, challenges, and open problems. *IEEE Communications Surveys Tutorials*, 21(3):2334–2360, 2019.

[4] Norvig Russell. *Intelligenza Artificiale, un approccio moderno*. Pearson, 2021.

[5] Wikipedia. Matrice laplaciana — wikipedia, l'enciclopedia libera, 2023. [Online; in data 13-marzo-2024].

[6] Wikipedia contributors. Algebraic connectivity — Wikipedia, the free encyclopedia, 2023. [Online; accessed 13-March-2024].

[7] Chenxi Zhao, Junyu Liu, Min Sheng, Wei Teng, Yang Zheng, and Jiandong Li. Multi-uav trajectory planning for energy-efficient content cove-

rage: A decentralized learning-based approach. *IEEE Journal on Selected Areas in Communications*, 39(10):3193–3207, 2021.