ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

MECCANISMI EVOLUTIVI PER LA PROGETTAZIONE AUTOMATICA ONLINE DI RETI BOOLEANE PER SCIAMI DI ROBOT

Tesi in
SISTEMI INTELLIGENTI ROBOTICI

Relatore
Prof. ANDREA ROLI

Co-relatore
Dott. MICHELE BRACCINI

Presentata da
GIULIA LUCCHI

Seconda Sessione di Laurea
Anno Accademico 2019 – 2020

PAROLE CHIAVE

Progettazione Automatica

Meccanismi Evolutivi

Evoluzione Online

Sciami di Robot

Reti Booleane

Ai miei genitori e al mio ragazzo.

Indice

Introduzione

La progettazione automatica del software di controllo di un robot è un argomento importante nella robotica attuale che mira a superare le caratteristiche del design umano classico, le quali potrebbero risultare limitate in quei casi in cui il compito del robot o l'ambiente in cui si trova l'agente non possa essere formalmente specificato o definito in anticipo.

La computazione evolutiva è stata ampiamente studiata e applicata per sintetizzare software di controllo per robot autonomi nel campo della robotica evolutiva (ER). Negli approcci di ER online, un algoritmo evolutivo (EA) viene eseguito sui robot durante l'esecuzione delle attività per ottimizzarne continuamente il comportamento. L'evoluzione online dei software di controllo è un processo di adattamento continuo che potenzialmente fornisce ai robot la capacità di rispondere a cambiamenti o circostanze impreviste modificando il loro comportamento. Le componenti principali dell'EA (valutazione, selezione e riproduzione) sono svolte in autonomia dai robot senza alcuna supervisione esterna, in questo modo, i robot possono essere in grado di autoadattarsi a lungo termine in modo completamente autonomo.

Negli ultimi decenni sono stati sviluppati numerosi approcci di evoluzione online per sistemi multi-robot. Gli esempi includono l'approccio di Bianco e Nolfi per i robot autoassemblanti [Bianco and Nolfi, 2004], mEDEA di Bredeche et al. [Bredeche et al., 2012] e odNEAT di Silva et al. [Silva et al., 2012].

La motivazione principale alla base dell'uso di sistemi multi-robot è stata quella di sfruttare la potenziale accelerazione dell'evoluzione dovuta a robot che evolvono i controller in parallelo e che scambiano soluzioni candidate per l'attività. Con l'utilizzo di sistemi multi-robot, si distinguono tre approcci di evoluzione online [Eiben et al., 2010a]: l'evoluzione incapsulata, l'evoluzione distribuita e l'evoluzione ibrida.

L'obiettivo principale di questa tesi è quello di implementare e analizzare diversi meccanismi evolutivi di progettazione automatica online, in grado di permettere ad uno sciame di robot di imparare autonomamente a svolgere determinati task, attraverso l'utilizzo di controller basati su Random Boolean Network.

In particolare, sono stati sviluppati tre meccanismi, il primo utilizza un approccio incapsulato e tramite esso si esegue una evoluzione senza scambio di informazioni tra i robot, il secondo e il terzo meccanismo utilizzano un approccio ibrido e, con tecniche differenti, permettono allo sciame di sincronizzarsi, ovvero di condividere tra tutti i robot la migliore rete trovata durante l'evoluzione fino ad un determinano momento. I meccanismi di aggiornamento guideranno l'evoluzione delle reti dei robot, che verranno valutate, durante l'esecuzione, da una apposita funzione obiettivo in base al task da risolvere. Il primo task studiato consiste nel distribuirsi uniformemente all'interno di un'arena libera, mentre il secondo nel distribuirsi uniformemente all'interno di un'arena nella quale sono presenti alcune zone proibite.

Al termine di questo studio si analizzeranno i risultati per identificare quale meccanismo ottiene i risultati migliori e per valutare se la sincronizzazione può essere d'aiuto o meno per l'apprendimento.

Nel Capitolo 1 verrà fornito il background e discuteremo lo stato dell'arte da cui deriva questa tesi, con particolare attenzione ai concetti di progettazione automatica offline e online, di meccanismi per il processo evolutivo online e di reti booleane.

Nel Capitolo 2 verranno descritte le strategie e le metodologie utilizzate per affrontare il nostro problema di progettazione di controller per uno sciame di robot. In particolare, approfondiremo la descrizione dei task, delle funzioni obiettivo, della configurazione dei controller dei robot, del modello delle BN adottate e dei meccanismi evolutivi ideati.

Infine, all'interno del Capitolo 3 descriveremo i vari esperimenti e le configurazioni delle simulazioni, ed effettueremo l'analisi dei risultati ottenuti da ciascun meccanismo evolutivo sia in fase di addestramento che in fase di test.

Capitolo 1

Background e Stato dell'Arte

In questo primo capitolo, verranno introdotti gli argomenti trattati e le principali opere che hanno contribuito alla realizzazione della tesi. In particolare, verrà approfondito lo stato dell'arte riguardante la progettazione automatica, le Reti Booleane e la possibilità di applicarle alla progettazione di sistemi robotici. Verranno, inoltre, introdotti i principali meccanismi evolutivi per la progettazione automatica online di sciami di robot.

L'obiettivo è quello di fornire lo sfondo per il capitoli successivi e per sintetizzare lo stato dell'arte degli argomenti principali coperti in questa tesi.

1.1 Progettazione Automatica di Sciami di Robot

Nella robotica collettiva, un grande numero di robot cooperano per compiere un compito che va oltre le capacità di un singolo robot [Dorigo and Roosevelt, 2004]. Lo sciame di robot opera in maniera auto-organizzata e distribuita, senza la presenza di un leader, e la coordinazione avviene tramite interazione tra i robot individuali. Il comportamento collettivo dello sciame è il risultato delle interazioni locali che ogni robot ha con i suoi vicini e con l'ambiente. Il singolo robot agisce sulla base delle informazioni locali ottenute attraverso i suoi sensori o fornite da robot vicini tramite comunicazione locale.

La natura auto-organizzata e distribuita degli sciami di robot li rende difficili da progettare. La *progettazione automatica* è un approccio promettente per lo sviluppo di software di controllo per questo tipo di sistemi. In questo caso, il problema di progettare un software di controllo per uno sciame diventa un problema di ottimizzazione, in cui le scelte di progettazione delle differenti istanze del controller definiscono una spazio di ricerca che viene esplorato utilizzando un algoritmo di ottimizzazione.

La maggior parte dei metodi automatici per sciami di robot proposti fino ad ora in letteratura fanno parte della *robotica evolutiva* (ER) [Nolfi et al., 2000]. La robotica evolutiva è un approccio altamente generale, in quanto consente la sintesi del controllo data solo una specifica dell'attività e non è legata a specifici algoritmi evolutivi, sistemi di controllo o tipi di robot [Silva et al., 2017]. Questo approccio ha quindi il potenziale per consentire la progettazione automatica di sistemi di controllo senza la necessità di specifiche manuali e dettagliate del comportamento desiderato [Floreano and Keller, 2010, Nolfi, 1998] e permette di sfruttare il modo in cui il mondo viene percepito tramite i sensori dei robot [Floreano and Mondada, 1994]. La robotica evolutiva, infatti, è stata adottata con successo per progettare sciami di robot che svolgono diversi task ed i risultati ottenuti mostrano che la progettazione automatica è un approccio praticabile e promettente per progettare il software di controllo degli sciami di robot [Francesca and Birattari, 2016].

Generalmente, gli approcci basati sulla robotica evolutiva mantengono una popolazione di genomi, ciascuno dei quali codifica il sistema di controllo per uno o più robot. Questi approcci utilizzano tecniche di computazione evolutiva per generare robot che si adattano all'ambiente attraverso un processo analogo all'evoluzione naturale [Holland, 1962]. In maniera simile ai metodi evolutivi, un tradizionale processo di robotica evolutiva richiede un feedback dato da una misura della prestazione complessiva, che indica il valore di *fitness* di un individuo della popolazione. Questo valore di fitness è calcolato tramite una funzione di fitness o funzione obiettivo che premia il miglioramento verso un obiettivo dipendente dal task [Silva et al., 2017].

In un approccio evolutivo, ogni robot esegue autonomamente il proprio algoritmo evolutivo (EA) al suo interno per ottimizzare il proprio controller. In linea di principio, questo EA può essere di qualsiasi tipo, purché corrisponda alla rappresentazione dei controller del robot [Eiben et al., 2010b]. Considerando la differenza tra il controller di un robot e il codice evolutivo che rappresenta quel controller, in generale, un controller di un robot è un'entità strutturalmente e proceduralmente complessa che determina direttamente il comportamento del robot. Quando si utilizzano metodi evolutivi per la progettazione dei controller, i controller vengono percepiti come *fenotipi* rappresentati da parti di codice relativamente semplici, chiamati *genotipi*. La fitness di un controller di un robot, tipicamente le performance del task, viene quindi determinata dal fenotipo, mentre, in conformità ai principi dell'evoluzione biologica, sono solo i genotipi a subire operazioni di evoluzione: mutazione e/o crossover. In generale, un robot può contenere più di un genotipo, ma in ogni momento solo uno di questi può essere attivo, cioè decodificato in un fenotipo/controller funzionante.

La Figura 1.1 illustra questa concetto.

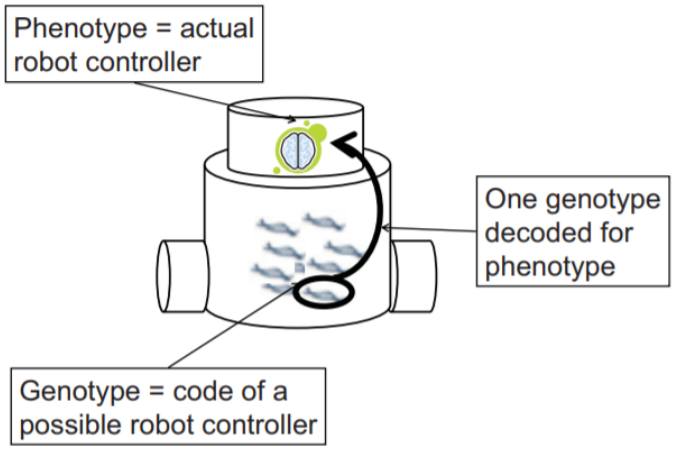

Figura 1.1: Esempio di genotipo e fenotipo di un controller di un robot. Immagine tratta da [Eiben et al., 2010a] e riprodotta con il permesso degli autori.

L'obiettivo di questi approcci è quello di evolvere il comportamento di robot autonomi in ambienti impredicibili, dinamici o parzialmente sconosciuti.

1.1.1 Metodologie di Progettazione Automatica: Offline e Online

Le metodologie di progettazione automatica possono essere suddivise in due classi [Francesca and Birattari, 2016]: *offline* e *online*.

Metodi offline Tradizionalmente, la robotica evolutiva si concentra su applicazioni offline, in cui un algoritmo evolutivo viene utilizzato per progettare e ottimizzare i controller dei robot prima della distribuzione.

Nei metodi offline, si ha una fase preliminare dedicata in cui prende luogo il processo di progettazione del software di controllo dello sciame di robot. Questa fase di progettazione si verifica e termina prima che lo sciame di robot sia inserito nel suo ambiente operativo.

Questa tipologia di progettazione valuta un ampio numero di differenti istanze di software di controllo, solitamente attraverso una simulazione eseguita su un computer esterno al robot. I controller si evolvono in simulazione per un certo numero di generazioni, o fino a quando non viene soddisfatto un determinato criterio di prestazione, dopo di che il controller che ha ottenuto i

risultati migliori verrà implementato su robot reali, che successivamente eseguiranno il loro compito senza ulteriore adattamento. La limitazione principale della simulazione offline è data dal fatto che nessuna evoluzione o adattamento avviene online, per cui i controller sono soluzioni fisse che rimangono statiche per tutta la vita del robot. Se le condizioni ambientali, oppure i parametri dell'attività, cambiano da quelli riscontrati durante l'evoluzione offline, i software di controllo evoluti potrebbero non essere in grado di risolvere l'attività, poiché non hanno alcun mezzo per adattarsi [Silva et al., 2014].

Nonostante ciò, la simulazione offre alcuni benefici [Francesca and Birattari, 2016, Silva et al., 2016]:

- in primo luogo, l'evoluzione offline in genere richiede meno tempo dell'evoluzione online, anche se è necessario sviluppare in anticipo ambienti di simulazione adeguati;

- in secondo luogo, l'evoluzione offline consente al ricercatore di concentrarsi sullo sviluppo del metodo di controllo, senza dover affrontare problemi intrinsecamente associati ai robot fisici, quali usura, danni potenziali, calibrazione e così via. La simulazione, infatti, impedisce ai robot di essere danneggiati da una possibile istanza di bassa qualità del software di controllo.

Metodi online L'evoluzione online è un processo di adattamento continuo che potenzialmente offre ai robot la capacità di rispondere ai cambiamenti nel task o nelle condizioni ambientali, modificandone il comportamento. Il processo di progettazione ha luogo quando lo sciame di robot è già stato distribuito nel suo ambiente operativo. Un algoritmo evolutivo viene eseguito sui robot stessi per fornire un adattamento continuo, mentre i robot svolgono i loro compiti nella vita reale. I componenti principali dell'algoritmo evolutivo (valutazione, selezione e riproduzione) vengono eseguiti autonomamente da e tra i robot senza alcuna supervisione esterna, in questo modo, i robot possono essere in grado di auto-adattarsi a lungo termine in maniera completamente autonoma [Silva et al., 2014]. In altre parole, i metodi online aspirano a produrre un design che sia più su misura per il task specifico rispetto a quello prodotto con un metodo offline.

In una impostazione online realistica, il processo di progettazione è completamente distribuito sui robot e non può contare su alcuna entità centralizzata per misurare le prestazioni e guidare la ricerca, come può in un'impostazione offline. Pertanto, il processo di progettazione è vincolato all'utilizzo di un indicatore delle prestazioni che può essere valutato in modo distribuito e locale.

L'apparente vantaggio dei metodi online rispetto ai metodi offline è che i metodi online possono beneficiare della disponibilità di informazioni sull'am-

biente operativo effettivo, che non sarebbero disponibili al momento della progettazione offline [Francesca and Birattari, 2016]. Sul lato negativo, i metodi online sono vincolati a uno spazio di ricerca ridotto rispetto ai metodi offline [Francesca and Birattari, 2016], poiché, con il processo di progettazione eseguito dai robot stessi mentre sono operativi, le risorse computazionali ed il tempo potrebbero essere fattori limitanti, inoltre, controller che potrebbero risultare potenzialmente pericolosi per i robot dovrebbero essere rimossi a priori dallo spazio di ricerca per evitare i potenziali pericoli.

Nel contesto della progettazione automatica online di software di controllo per sciami di robot, sono state individuate le seguenti caratteristiche [Francesca and Birattari, 2016]:

- In genere, ogni robot esplora una porzione dello spazio di ricerca, valutando in modo asincrono una sotto-popolazione di istanze di software di controllo, tenendo traccia delle loro prestazioni;

- I robot potrebbero scambiare informazioni per coordinare il processo di ricerca, tipicamente, condividendo le istanze di software di controllo con le migliori prestazioni;

- Gli sciami di robot sono eterogenei dal punto di vista comportamentale, cioè ogni robot esegue una diversa istanza di software di controllo. Ciò non esclude necessariamente che i robot possano eventualmente convergere nell'esecuzione della stessa istanza del software di controllo;

- Poiché ogni robot dello sciame è tenuto a valutare le istanze del software di controllo, la funzione obiettivo deve essere calcolabile basandosi solo sulle informazioni disponibili localmente per il robot. Ciò limita la classe di compiti che sono affrontati in letteratura utilizzando l'approccio online. In questo caso, i compiti che possono essere affrontati sembrano essere più semplici e meno diversificati rispetto a quelli affrontati nell'approccio offline;

- Gli esperimenti di evoluzione online possono essere condotti unicamente in simulazione (come spesso si trova in letteratura), oppure esclusivamente sui robot. Un possibile compromesso potrebbe essere quello di progettare il software di controllo in parte tramite simulazione ed in parte sui robot.

In questa tesi, considereremo l'applicazione online della robotica evolutiva per la progettazione di software di controllo di sciami di robot.

1.2 Meccanismi per il Processo Evolutivo Online

La computazione evolutiva è stata ampiamente studiata nel campo della robotica come mezzo per automatizzare la progettazione di sistemi robotici [Floreano and Keller, 2010].

I tradizionali algoritmi computazionali evolutivi funzionano in modo discreto e centralizzato. Un componente esterno crea una popolazione iniziale ed è responsabile della selezione, della mutazione e della sostituzione degli individui. L'evoluzione viene generalmente eseguita offline, anche se il soggetto dell'ottimizzazione o del design è un agente fisico come un robot. Gli approcci evolutivi tradizionali presentano, quindi, una serie di carenze nell'evoluzione di controller robotici [Silva et al., 2012]. I controller vengono trasferiti ai robot post-evoluzione e, una volta sviluppati, sono quindi specializzati in un determinato compito e condizioni ambientali. Sono soluzioni fisse e presentano una capacità limitata di adattamento agli ambienti e ai compiti non visti durante l'evoluzione. Affinché un algoritmo evolutivo (EA) dia ai robot la capacità di adattarsi continuamente, come detto nella sezione 1.1.1, deve essere eseguito sui robot stessi, mentre i robot svolgono i loro task, in altre parole, l'evoluzione deve essere condotta *online*.

La ricerca sull'evoluzione online è iniziata con uno studio di Floreano e Mondada [Floreano and Mondada, 1994], che hanno condotto esperimenti su un robot reale. Gli autori hanno evoluto con successo comportamenti di navigazione e di obstacle avoidance per un robot Khepera. Lo studio è stato un importante passo avanti in quanto ha dimostrato il potenziale dell'evoluzione online dei controller.

1.2.1 Classificazione degli approcci evolutivi

Nell'ultimo decennio sono stati proposti diversi approcci online per *sistemi multi robot*. Gli approcci esistenti di evoluzione online nei sistemi robotici rientrano in tre categorie [Eiben et al., 2010a]:

- **Distributed Evolution**: ogni robot mantiene un singolo genotipo. Il processo evolutivo ha luogo quando i robot si incontrano e scambiano informazioni genetiche. Il principale svantaggio degli approcci ad evoluzione distribuita è che il miglioramento delle soluzioni esistenti si basa solo sullo scambio di informazioni genetiche tra robot. In ambienti ampi e aperti, dove gli incontri possono essere rari o la trasmissione frequente di informazioni può essere impossibile, il processo evolutivo può essere soggetto a stagnazione.;

- **Encapsulated Evolution**: ogni robot mantiene una popolazione di genotipi immagazzinata internamente e gestisce localmente un'istanza dell'algoritmo evolutivo. Ogni robot esegue un sottoinsieme di controller e non è presente nessuna comunicazione tra i robot che possa influenzare il processo evolutivo, quindi questo approccio non ricade nel problema della stagnazione. Controller alternativi vengono eseguiti in sequenza e viene misurata la loro idoneità. Con questo paradigma, i robot si adattano individualmente senza scambiare materiale genetico con altri robot. Il principale svantaggio dell'approccio è quindi la mancanza di trasferimento di conoscenze o scambio di informazioni genetiche tra robot, che può accelerare l'evoluzione online [Huijsman et al., 2011].;

- **Hybrid Evolution**: le due precedenti metodologie possono essere combinate, ottenendo un approccio in cui ogni robot trasporta più genomi e li condivide con i suoi pari. Il processo evolutivo in esecuzione su ogni singolo robot è quindi autosufficiente per quanto riguarda l'ottimizzazione della popolazione interna ed è potenzialmente accelerato a causa del parallelismo intrinseco e dello scambio di informazioni tra robot. Nell'evoluzione online, il fatto che i robot operano in parallelo implica che possono anche imparare in parallelo [Silva et al., 2017].

Prospettiva spaziale: On-board vs Off-board

Dal punto di vista spaziale, vengono distinti due casi [Silva et al., 2017, Eiben et al., 2010a]:

- **on-board**: tutti i calcoli necessari e gli operatori evolutivi come la selezione, il crossover e la mutazione vengono eseguiti dai robot stessi, considerando la loro potenza di elaborazione e la capacità di archiviazione potenzialmente limitate;

- **off-board**: i calcoli richiesti e gli operatori evolutivi vengono eseguiti con l'ausilio di apparecchiature esterne al di fuori dei robot. Tale apparecchiatura esterna potrebbe essere un computer, interfacciato con i robot, ovvero, un "master" che gestisce il processo evolutivo online sulle basi delle informazioni di fitness che raccoglie dai robot, ad esempio gestendo gli operatori evolutivi di selezione e variazione e iniettando nuovi controller nei robot.

1.2.2 Background sui meccanismi di evoluzione online per sistemi Multi Robot

Di seguito, verranno descritti i principali approcci studiati in letteratura per l'evoluzione online di sistemi multi robot, sulla base dello schema di classificazione descritto nella sezione 1.2.1.

Distributed Evolution Il primo tentativo di evoluzione online veramente autonoma in sistemi multi robot è stato effettuato da Watson et al. [Watson et al., 2002], i quali hanno introdotto un approccio chiamato *Embodied Evolution* (EE) in cui un EA online è distribuito su un gruppo di robot. La motivazione principale alla base dell'uso dei sistemi multi robot è stata quella di sfruttare la potenziale accelerazione dell'evoluzione dovuta ai robot che evolvono i controller in parallelo e che scambiano soluzioni candidate del task.

Ogni robot mantiene un livello di energia virtuale e un punteggio di fitness che riflette la prestazione individuale. Ad ogni ciclo di controllo, i robot aggiornano il loro punteggio di fitness individuale in base alle performance del task, e quindi trasmettono probabilisticamente una parte del loro genoma stocasticamente mutato con una probabilità proporzionale alla loro fitness. Se un robot trasmette una stringa genetica, il punteggio di fitness viene diminuito di un ammontare costante, come una penalità analoga ai costi di riproduzione. I robot che ricevono le trasmissioni incorporano questo materiale genetico nel loro genoma con una probabilità inversamente proporzionale alla loro forma fisica. In questo modo, gli operatori di selezione e variazione vengono implementati in maniera distribuita attraverso le interazioni tra robot.

Successivamente, diversi miglioramenti ed estensioni della EE sono state proposte. Bianco e Nolfi hanno proposto una metodologia che coinvolge un gruppo di unità robot simulate con la capacità di autoassemblare un altro robot [Bianco and Nolfi, 2004]. Durante l'esecuzione, ogni robot simulato è stato in grado di riprodursi e crescere usando il corpo di altri robot. La riproduzione è stata ottenuta iniettando il genoma in un altro robot nelle immediate vicinanze. La metodologia è stata valutata in tre serie di esperimenti effettuati in simulazione. I risultati sperimentali hanno mostrato l'emergere di morfologie con forme specifiche ben adattate a diversi scenari. Tuttavia, un aspetto limitante dell'approccio è la difficoltà nello svolgimento di esperimenti simili usando gruppi di robot reali [Trianni, 2006].

Karafotias et al. [Karafotias et al., 2011] hanno proposto Embodied Distributed Evolutionary Algorithm (EDEA) valutandone le prestazioni in simulazione. Uno degli obiettivi alla base di EDEA è quello di cercare di superare la coincidenza spaziale necessaria per lo scambio di materiale genetico tra robot. L'algoritmo comprende una fase di identificazione dei partner che tenta

di fornire ai robot una visione globale della popolazione attraverso canali di comunicazione a lungo raggio. Ogni robot contatta tutti i robot nel raggio d'azione, identifica i potenziali partner con cui scambiare materiale genetico e quindi seleziona una soluzione candidata. Dopo aver scelto la soluzione candidata, il robot confronta la propria fitness con quella del potenziale partner, e probabilisticamente applica gli operatori di ricombinazione e mutazione. La prole viene utilizzata come nuovo controller.

EDEA è concettualmente simile ad un approccio distribuito introdotto in [Simoes and Barone, 2002]. In tale approccio, i robot eseguono una procedura ciclica composta da una fase di esecuzione, in cui tentano di eseguire il task, e una fase di accoppiamento sincronizzata che viene attivata da un timer interno. Nella fase di accoppiamento, ciascun robot utilizza un collegamento radio per comunicare il punteggio di fitness e il genoma ad ogni altro robot, mentre riceve le stesse informazioni dai robot rimanenti.

Encapsulated Evolution In letteratura, questo tipo di approcci sono meno studiati rispetto ad approcci di evoluzione distribuita o di evoluzione ibrida [Silva et al., 2017]. Bredeche et al. [Bredeche et al., 2009] hanno proposto l'algoritmo (1 + 1)-online, dove ogni robot mantiene un solo individuo per robot e la variazione degli individui è limitata alla mutazione. Il nuovo individuo sostituisce il migliore solamente se ottiene una fitness maggiore. Successivamente, Haasdijk et al. [Haasdijk et al., 2010] hanno proposto l'algoritmo (μ + 1)-online, che mantiene una popolazione di μ individui è all'interno di ciascun robot e viene utilizzata anche l'operazione di crossover tra individui, oltre alla mutazione. In entrambi questi approcci gli individui vengono valutati sequenzialmente durante un periodo di valutazione predefinito.

Hybrid Evolution Un esempio di questi approcci è dato da Huijsman et al. [Huijsman et al., 2011] che hanno introdotto un approccio basato sull'algoritmo incapsulato (μ + 1)-online e su un algoritmo evolutivo peer-to-peer chiamato EvAg [Laredo et al., 2010] . In una serie di diversi esperimenti basati sulla simulazione, il metodo ibrido ha costantemente prodotto prestazioni più elevate, sia dell'evoluzione distribuita, che dell'evoluzione incapsulata.

In [Bredeche et al., 2012] è stato introdotta una variante dell'algoritmo EDEA [Karafotias et al., 2011] intitolata minimal EDEA (mEDEA), in cui ogni robot mantiene un livello di energia che riflette le prestazioni dell'attuale controller, in più ai controller viene assegnata una durata. Durante l'esecuzione del task, i robot trasmettono continuamente i loro genomi ad altri robot nel raggio di comunicazione, fintanto che il livello di energia è superiore allo zero e la durata non è scaduta. Se il livello di energia raggiunge lo zero durante la vita del controller, il robot non funzionerà e rimarrà fermo. Quando la durata

di un controller scade, l'algoritmo seleziona casualmente uno dei genomi ricevuti, lo muta e le operazioni di esecuzione e trasmissione vengono riavviate. All'interno di questa configurazione, i genomi che permettono al robot di muoversi maggiormente nell'ambiente hanno maggiori probabilità di diffondersi a un ritmo più elevato rispetto ai genomi che causano un minor spostamento del robot.

Silva et al.[Silva et al., 2015b] hanno introdotto odNEAT, una variante del real-time NEAT [Stanley and Miikkulainen, 2002] per l'ottimizzazione online e decentralizzata del software di controllo dei robot. odNEAT è distribuito su più robot che si evolvono in parallelo e scambiano soluzioni candidate del task quando si incontrano. Ogni robot ottimizza una popolazione interna di genomi e vengono scambiate informazioni genetiche tra due o più robot. Durante l'esecuzione del task, ciascun robot è controllato da un ANN che rappresenta una soluzione candidata per un determinato task. I controller mantengono un livello di energia virtuale che riflette le loro prestazioni individuali. Il valore di fitness è definito come livello di energia medio. Quando il livello di energia virtuale di un robot raggiunge una soglia minima, il controller corrente viene considerato inadatto per l'attività e viene creato un nuovo controller tramite la selezione, crossover e mutazione.

Nonostante i progressi, da ciò che traspare dallo stato dell'arte, risulta evidente che i ricercatori hanno valutato gli algoritmi proposti principalmente attraverso la simulazione, infatti, ancora pochi studi sull'evoluzione online sono stati condotti su robot reali [Nelson et al., 2009, Koos et al., 2012]. Di conseguenza, anche se nuovi algoritmi sono stati introdotti e gli approcci riguardante l'evoluzione online sono maturati, la classe di attività affrontato non è aumentato in complessità [Silva et al., 2015a].

1.3 Reti Booleane

Le reti booleane (BN) sono un modello di rete di regolazione genetica (GNR) introdotto da Kauffman [Kauffman, 1969]. Queste reti hanno dimostrato di catturare efficacemente importanti fenomeni nella regolazione genica e sono di interesse dal punto di vista ingegneristico, grazie alla loro espressività comportamentale ricca e complessa, nonostante la compattezza del modello [Roli et al., 2011b].

I primi studi sull'evoluzione delle reti booleane sono stati effettuati in [Kauffman and Smith, 1986, Lemke et al., 2001]. Questi studi sono principalmente un'indagine sull'evoluzione di reti booleane e sollevano domande interes-

santi e fondamentali sulle dinamiche evolutive a seconda dalle caratteristiche strutturali della rete.

Una BN è un sistema dinamico stato-discreto e tempo-discreto la cui struttura è definita da un grafo orientato di N nodi, in cui ogni nodo è associato ad una variabile booleana $x_i, i = 1, ...N$, e ad una funzione booleana $f_i(x_{i_1}, ..., x_{i_{K_i}})$, dove K_i è il numero di input del nodo i. Gli argomenti della funzione booleana f_i sono i valori dei nodi i cui archi uscenti sono collegati al nodo i. Per cui ogni nodo può assumere, ad ogni istante di tempo, un valore booleano che dipende dai valori dei nodi entranti e dalla funzione booleana associata al nodo stesso. Lo stato del sistema al tempo t, $t \in \mathbb{N}$, è definito dall'array delle N variabili booleane al tempo t: $s(t) \equiv (x_1(t), ..., x_N(t))$.

I modelli di reti booleane più studiati sono caratterizzati da dinamica *sincrona*, in cui i nodi aggiornano i loro stati in parallelo, e *funzioni deterministiche*. Esistono, tuttavia diverse varianti, con regole di aggiornamento asincrone e probabilistiche [Gershenson, 2004].

In figura 1.2 viene mostrato un esempio di rete booleana.

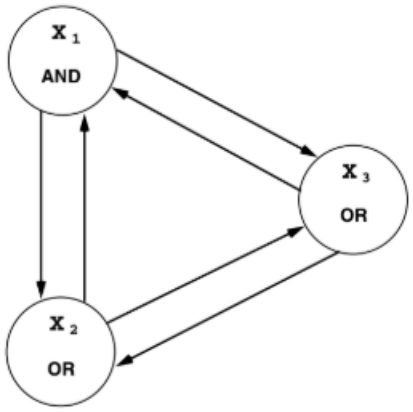

Figura 1.2: Esempio di rete booleana con $N = 3$ e $K = 2$

1.3.1 Random Boolean Network

Una categoria speciale di BN che ha ricevuto particolari attenzioni per la progettazione di software di controllo in robotica è quella delle Random BN (RBN). Queste reti sono in grado di catturare fenomeni rilevanti nei meccanismi genetici e cellulari, ma anche nei sistemi complessi in generale [Roli et al., 2011a].

Una RBN è una BN in cui, sia la topologia, che le funzioni booleane sono generate casualmente scegliendo K input per nodo in maniera casuale tra i restanti $N - 1$ nodi e definendo le funzioni booleane tramite l'assegnamento

a ciascuna entry delle tabelle di verità un 1 con probabilità p e uno 0 con probabilità $1 - p$. Il parametro p è chiamato *bias*.

1.3.2 Reti Booleane per la Progettazione di Sistemi Robotici

Le reti booleane possono essere utilizzate all'interno del software di controllo dei robot.

La progettazione di reti booleane è di per se un compito difficile e l'utilizzo di questi sistemi come software di controllo per robot può essere ancor più complicato, in quanto la rete è tenuta ad interagire con l'ambiente, che possiede una propria dinamica. La complessità di questo compito, tuttavia, può essere domata mediante l'uso di procedure di progettazione automatica, che possono rendere il processo più solido e generale rispetto ad una procedura personalizzata [Roli et al., 2011b].

Le BN sono solitamente considerate sistemi isolati, il che significa che lo stato della rete non è influenzato da fattori esterni [Roli et al., 2011b]. La rete, quindi, tenderà a raggiungere uno stato di stabilità chiamato *attrattore*, ovvero, una sequenza ciclica o un modello di stati nello spazio degli stati della rete che rappresenta uno stato stabile del sistema dinamico. Per cui, a partire da qualsiasi stato di un BN, dopo una serie di stati transitori, la rete raggiungerà un attrattore di qualche tipo. L'insieme di stati che porta allo stesso attrattore è chiamato *Bacino di Attrazione*.

Nei sistemi robotici, queste reti vengono utilizzate assegnando al valore di un insieme di nodi (nodi di input della rete booleana) le letture dei sensori e utilizzando il valore di un altro insieme di nodi (nodi di output della rete) per codificare i segnali utili a controllare gli attuatori dei robot. Una volta che sono stati definiti i mapping di input e output, la rete che forma il programma del robot deve essere progettata. Una possibilità consiste nel modellare il processo di progettazione della rete come un problema di ricerca, nel quale il goal è massimizzare le performance del robot [Roli et al., 2011b].

In figura 1.3 viene mostrato un esempio di schema di accoppiamento tra rete booleana e robot.

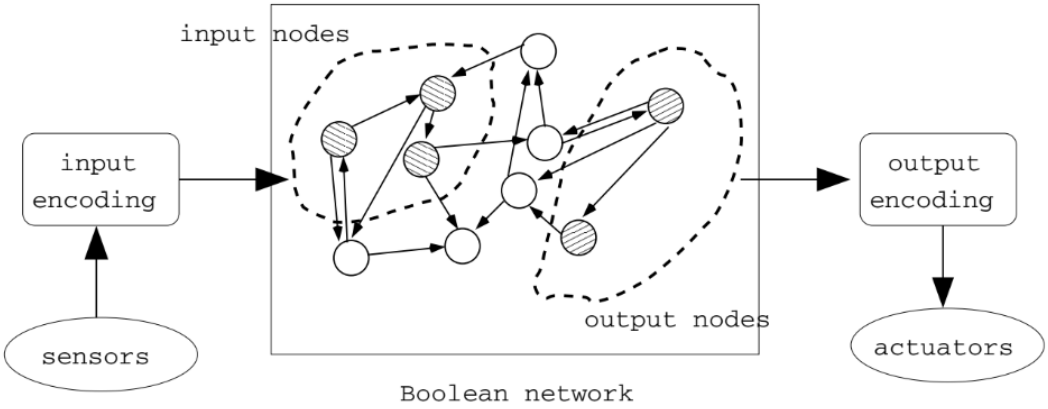

Figura 1.3: Esempio di schema di accoppiamento tra BN e robot, tratto da [Roli et al., 2011b] e riprodotta con il permesso degli autori.

Collegando, quindi, alcuni nodi ai sensori dei robot si aggiungono alla rete booleana delle perturbazioni esterne, grazie alle quali, sarà possibile passare da un attrattore ad un altro. Questo passaggio potrà permettere al robot di eseguire comportamenti diversi sulla base della percezione dei sensori.

In un processo evolutivo, le reti booleane potranno essere valutate variando le entry delle tabelle delle verità dei nodi, ovvero manipolando le funzioni booleane dei nodi, e/o la topologia della rete. Per ogni rete ottenuta da queste variazioni verranno valutate la performance utilizzando una funzione di fitness/obiettivo.

Capitolo 2

Meccanismi di Aggiornamento per l'Evoluzione Online di Reti Booleane

In questo capitolo, introduciamo il problema affrontato durante la tesi, presentando e descrivendo i meccanismi applicati e le soluzioni adottate per affrontare il problema.

Verranno descritti i task che dovranno compiere i robot e la configurazione dei software di controllo e delle reti booleane che controllano i robot. Infine, verranno presentate le funzioni obiettivo utilizzate per valutare la fitness delle reti evolute dai singoli robot e i meccanismi utilizzati per l'evoluzione.

I risultati ottenuti verranno descritti all'interno del capitolo 3.

2.1 Caso di studio

All'interno di questa tesi esploreremo diversi meccanismi evolutivi di progettazione automatica online, in grado di permettere ad uno sciame di robot di imparare autonomamente a svolgere determinati task, descritti di seguito, attraverso l'utilizzo di controller basati su Random Boolean Network.

In questo capitolo verranno presentati gli algoritmi sviluppati per il problema in esame. L'obiettivo principale della tesi non è quello di trovare il miglior meccanismo evolutivo online, ma si vuole avviare un processo di ricerca in cui i meccanismi proposti vengono testati e confrontati, in modo tale da valutare quali vantaggi possa portare una sincronizzazione globale delle reti evolute all'interno dello sciame dei robot.

In particolare, verranno valutati tre diversi meccanismi evolutivi: senza sincronizzazione, con sincronizzazione e con sincronizzazione reversibile, che verranno descritti all'interno delle Sezioni 2.5.1, 2.5.2 e 2.5.3.

Per la progettazione automatica online di sciami di robot utilizziamo Random Boolean Network. Da questo momento identifichiamo come BN-robot qualsiasi tipo di agente robotico dotato di un controller basato su reti booleane [Roli et al., 2011b]. I punti di forza di questo l'approccio derivano dalla ricchezza di possibili dinamiche, insieme all'adattabilità e alla robustezza, che esibiscono le BN. I robot dello sciame saranno quindi dotati di un controller basato su reti booleane, dove alcuni nodi sono impiegati come recettori dei valori recepiti dai sensori, mentre altri nodi si interfacciano con gli attuatori degli agenti robotici.

Effettueremo, quindi, una analisi dei risultati dell'evoluzione online del comportamento di uno sciame di BN-robot, dove i robot dovranno imparare autonomamente a svolgere determinati task. Lo sciame di robot, in particolare, dovrà affrontare due task, il primo più semplice ed il secondo più complesso. Il primo task, *Copertura con arena libera*, verrà descritto all'interno della sezione 2.1.1, mentre il secondo, *Copertura con zone proibite*, sarà la diretta evoluzione del primo a cui verranno aggiunte delle zone proibite non accedibili ai robot, e verrà descritto nella sezione 2.1.2.

L'evoluzione avviene in simulazione e le prestazioni dei robot sono valutate tramite l'utilizzo di funzioni obiettivo, che daranno un punteggio al comportamento del robot e ne guideranno il processo evolutivo. I meccanismi evolutivi si baseranno sui risultati dati da queste funzioni obiettivo riguardo alle reti dei singoli BN-robot.

2.1.1 Task 1: Copertura con arena libera

Il primo task presentato è un task di *copertura*, in cui lo sciame di robot ha lo scopo di coprire l'intera arena massimizzando l'area coperta dai robot. Le RBN sono progettate in modo da fungere da programma per i robot che hanno come obiettivo principale quello di sviluppare un comportamento di coverage, in più devono imparare ad evitare ostacoli e collisioni. In particolare, i robot devono massimizzare la distanza tra di loro all'interno dell'arena, evitando possibili ostacoli, come i muri o i restanti robot presenti, basandosi unicamente sulla propria conoscenza locale dell'arena ottenuta dalle percezioni dei sensori.

L'ambiente, in cui i robot sono simulati, consiste in una arena quadrata delimitata da delle mura. Non sarà presente nulla all'interno dell'arena, se non i robot appartenenti allo sciame.

All'interno del paragrafo 3.1 verrà descritta in maniera più dettagliata la configurazione dell'arena utilizzata.

2.1.2 Task 2: Copertura con zone proibite

Il secondo task è una diretta evoluzione del task precedente, che rende più complicato il processo di addestramento per lo sciame di robot.

L'obiettivo che devono raggiungere i robot dello sciame è quello di coprire l'intera arena, ad eccezione delle aree proibite. Le aree proibite sono aree presenti all'interno dell'arena caratterizzate da un pavimento nero (quadrati neri) e non accessibili ai robot.

Lo sciame deve massimizzare l'area coperta con l'eccezione delle zone nere proibite, senza entrare in collisione con i muri o con gli altri robot. La distanza tra i robot deve pertanto essere massima al di fuori delle aree vietate.

All'interno del paragrafo 3.1 verrà descritta in maniera più dettagliata la configurazione dell'arena utilizzata.

Nonostante i due task possano sembrare compiti semplici, non sono affatto banali. Si tratta, infatti, di processi evolutivi online che richiedono ai robot di affidarsi unicamente alle informazioni fornite dai sensori per imparare ad espandersi all'interno di un ambiente di cui non conoscono le caratteristiche. In aggiunta, all'interno dell'ambiente saranno presenti altri robot che, soprattutto inizialmente, potrebbero avere comportamenti differenti e non per forza coerenti con il comportamento richiesto dal task e questo ne va a complicare l'addestramento.

Applicazioni Il problema della copertura dell'area con sistemi multi robot è un campo di ricerca interessante. I potenziali vantaggi di questi sistemi sono la loro versatilità e robustezza nella realizzazione di molteplici attività. Gli sciami di robot, per questo tipo di task, possono risultare particolarmente utili in situazioni dove sono presenti impedimenti che porterebbero a limitazioni nella possibilità di azione delle persone, oppure in situazioni di pericolo per un essere umano, o in ambienti sconosciuti.

Un ambito applicativo dei task da noi sviluppati potrebbe essere quello delle emergenze dovute ad incendi. In [Penders et al., 2011], utilizzano uno lo sciame di robot per fornire informazioni di guida ai pompieri, supportandoli nella navigazione all'interno di un grande edificio. Il vantaggio nell'utilizzare uno sciame robot, in questo scenario, per coprire una zona di interesse, è dato dal fatto che i robot possono fornire informazioni riguardo alla planimetria dell'edificio tramite i sensori e gli strumenti con cui sono equipaggiati. Questo approccio è particolarmente efficacie in situazioni di scarsa visibilità, durante un incendio, a causa dal fumo all'interno dell'edificio, oppure in presenza di crolli e macerie.

Un ulteriore esempio di campo di interesse per la copertura di un'area con sciami di robot può essere quello della sorveglianza [Li et al., 2008], nel quale possono essere utilizzati strumenti come GPS, videocamera e sensori per catturare periodicamente immagini/video, oppure riconoscere oggetti o intrusi, migliorando in maniera significativa l'accuratezza delle informazioni e la tempestività delle azioni intraprese dalle guardie di sicurezza.

2.2 Funzioni Obiettivo

Ogni rete ottenuta dall'evoluzione dei robot durante il processo evolutivo verrà simulata e valutata da una funzione obiettivo, attraverso la quale ne verrà calcolata la fitness, ovvero l'idoneità della rete rispetto al task assegnato.

In un contesto di robotica evolutiva online, la funzione obiettivo dovrà prendere in considerazione unicamente le informazioni che verranno raccolte dai sensori durante l'esecuzione del robot all'interno dell'ambiente (nel nostro caso simulato).

La funzione obiettivo è definita diversamente a seconda del task in esame, mantenendo comunque la medesima logica, in modo tale da verificare che quest'ultima sia valida in task di coverage con elementi differenti.

In aggiunta al primo task di *copertura con arena libera*, per il secondo task di *copertura con zone proibite* è anche necessario tenere in considerazione le rilevazioni dei sensori del robot relative al colore del pavimento, per identificare

le zone proibite in cui non bisogna entrare. Entrambi i task dovranno tenere in considerazione i robot nelle vicinanze e la presenza di ostacoli.

La funzione obiettivo, utilizzata per guidare l'algoritmo evolutivo nel suo processo di ricerca, è una funzione che assegna punteggio positivo al robot nel caso in cui esegua uno spostamento lungo una determinata direzione che gli viene indicata. La direzione da seguire è data dalla risultante dei vettori, ottenuta come somma vettoriale dei vettori dati dalla percezione dei sensori del robot al tempo t, e ruotata di 180°. I vettori ottenuti dai sensori sono definiti sulla base di campi potenziali artificiali [Khatib, 1986]

La funzione obiettivo descritta si basa sulle informazioni ottenute dai seguenti sensori:

- *Proximity*: i sensori di prossimità del robot vengono utilizzati per rilevare possibili ostacoli (muri o altri robot);
- *Range and bearing*: il sistema RAB è utilizzato per ottenere distanza e direzione di tutti i robot vicini. I robot vicini sono i robot presenti all'interno di un dato range di visibilità. La distanza massima per la comunicazione viene impostata a 150 cm sulla base della dimensione dell'arena e del numero di robot dello sciame.
- *Base ground*: i sensori vengono utilizzati per controllare il colore del pavimento sotto al robot (valido unicamente all'interno del Task 2).
- *Differential steering*: il sensore è utilizzato per controllare la velocità attuale delle ruote, per verificare che il robot sia in movimento volontario;

La funzione obiettivo, da massimizzare, è quindi ottenuta dal contributo di più sensori (RAB, Proximity ed eventualmente Base Ground), da cui si ottengono le forze repulsive che sommate indicano la direzione giusta che dovrebbe seguire il robot. Queste forze repulsive permetto ai robot di allontanandosi dai robot a lui più vicini (entro un dato range) ed evitare collisioni con altri robot e ostacoli. In aggiunta, nel Task 2, sarà portato a non entrare all'intero delle zone proibite e ad allontanarsi da esse.

I vettori ottenuti dalle percezioni dei sensori vengono sommati e il risultato è dato da un vettore che indica la direzione della forza repulsiva risultante, ottenuta ruotando di 180° le percezioni.

Il vettore risultante può assumere una delle 8 direzioni (N, NO, O, SO, S, SE, E, NE) rispetto al sistema di riferimento del robot. Ogni direzione indica un angolo di 45° centrato nella direzione di interesse.

I vettori, prima di essere sommati, vengono normalizzati a un valore compreso tra $[0, 1]$, per fare in modo che tutti diano lo stesso contributo al vettore risultante.

Questa funzione obiettivo valuta il robot durante tutta l'esecuzione del task. Ad ogni step la funzione assegna punteggio 1 nel caso in cui il robot si stia muovendo lungo la direzione corretta, ovvero la direzione indicata dal vettore risultante e considerando un margine di 30° a destra e a sinistra rispetto all'angolo migliore atteso per lo spostamento, 0 altrimenti.

Task 1 La funzione obiettivo inerente a Task 1 è la seguente:

$$S_{F3_{al}} = \sum_{t=1}^{T} s_t \text{ , da massimizzare}$$

$$s_t = \begin{cases} 1, & \text{se } (V_{MOVt}.length > 0 \text{ e } V_{DIRt}.length > 0 \\ & \quad \text{e } (speed_{wlt} > 0 \text{ o } speed_{wrt} > 0) \\ & \quad \text{e } V_{MOVt}.angle - \frac{\pi}{6} < V_{DIRt}.angle < V_{MOVt}.angle + \frac{\pi}{6}) \\ & \text{o } (V_{MOVt}.length < 0 \text{ e } V_{DIRt}.length < 0) \\ 0, & \text{altrimenti} \end{cases}$$

Dove s_t è il punteggio assegnato a seconda del corretto spostamento del robot al tempo t, T è il tempo a disposizione dei robot per completare il task, V_{MOVt} è il vettore che indica lunghezza e direzione del movimento ideale che dovrebbe effettuare il robot al tempo t, V_{DIRt} è il vettore che indica lunghezza e direzione dello spostamento effettuato dal robot al tempo t, $speed_{wlt}$ indica la velocità della ruota sinistra al tempo t, $speed_{wrt}$ indica la velocità della ruota destra al tempo t.

Task 2 Per il Task 2 viene utilizzata la medesima funzione impiegata per il Task 1 con, in aggiunta, le informazioni fornite dai sensori *base ground*.

Tramite quest'ultimo sensore si controlla anche se il robot si trova completamente all'interno di una zona proibita, quando tutti gli 8 sensori *base ground* hanno un valore uguale a 0. In questo caso, verranno assegnati 0 punti al robot, nonostante si stia muovendo nella direzione corretta, in modo da penalizzare il robot che non deve entrare in una area proibita.

Infatti, se un robot si posiziona completamente all'interno di una zona proibita durante l'esplorazione, la risultante dei sensori *base ground* avrà modulo nullo e non andrà ad influire sul vettore risultante che indica la direzione da seguire. In questo caso, il robot, comportandosi "correttamente" seguendo la direzione del vettore risultante, otterrebbe punti anche all'interno della zona

proibita, cosa che non deve assolutamente accadere, per cui è stata aggiunta questa penalizzazione.

La funzione obiettivo inerente a Task 2 è la seguente:

$$S_{F3_{zp}} = \sum_{t=1}^{T} s_t \text{ , da massimizzare}$$

$$s_t = \begin{cases} 1, & \text{se } ((V_{MOVt}.length > 0 \text{ e } V_{DIRt}.length > 0 \\ & \quad \text{e } (speed_{wlt} > 0 \text{ o } speed_{wrt} > 0) \\ & \quad \text{e } V_{MOVt}.angle - \frac{\pi}{6} < V_{DIRt}.angle < V_{MOVt}.angle + \frac{\pi}{6}) \\ & \text{o } (V_{MOVt}.length < 0 \text{ e } V_{DIRt}.length < 0)) \\ & \text{e } \exists gi \in GROUND\_INDEXES \mid ground(gi).value = 1 \\ 0, & \text{altrimenti} \end{cases}$$

Dove s_t è il punteggio assegnato a seconda del corretto spostamento del robot al tempo t, T è il tempo a disposizione dei robot per completare il task, V_{MOVt} è il vettore che indica lunghezza e direzione del movimento ideale che dovrebbe effettuare il robot al tempo t, V_{DIRt} è il vettore che indica lunghezza e direzione dello spostamento effettuato dal robot al tempo t, $speed_{wlt}$ indica la velocità della ruota sinistra al tempo t, $speed_{wrt}$ indica la velocità della ruota destra al tempo t, $GROUND\_INDEXES$ sono gli indici dei sensori *base ground*. Ogni sensore può assumere valore 0 se rileva del pavimento nero (o grigio scuro) o 1 se rileva del pavimento bianco (o grigio chiaro).

La funzione obiettivo presentata (F3) è quella che ha ottenuto i risultati migliori all'interno di tutti gli esperimenti effettuati durante lo sviluppo della tesi.

Sono state realizzate e analizzate cinque ulteriori funzioni obiettivo per la valutazione delle prestazioni dei singoli robot relative ai due task descritti in precedenza, queste ultime sono presentate all'interno della tesi "*Sintesi e comparazione di funzioni obiettivo per l'evoluzione online di sciami di robot controllati da reti booleane*" di Luca Polverelli.

Di seguito verranno velocemente introdotte:

- F1: funzione obiettivo, da massimizzare, che assegna un punteggio positivo al robot ogni volta che si allontana dal robot a lui più vicino, e sulla

base del numero di step in cui non è stato in prossimità di altri robot o dei muri dell'arena.

- F2.1: funzione obiettivo, da massimizzare, che assegna un punteggio al robot sulla base della differenza tra il numero di vicini individuati al tempo t e quelli individuati al tempo $t-1$. La differenza dei vicini viene calcolata ad ogni step di esecuzione.

- F2.2: funzione obiettivo, da minimizzare, che assegna un punteggio sulla base del numero di vicini individuati al tempo T, dove T è il tempo a disposizione dei robot per il completamento del task.

- F2.3: funzione obiettivo, da massimizzare, che assegna un punteggio al robot sulla base della differenza del numero di vicini individuati al tempo T e quelli individuati al tempo 0. Dove per tempo T si intende la fine tempo a disposizione del robot per completare il task, e per tempo 0 si intende l'istante di tempo prima che venga effettuato il primo step.

- F4: funzione obiettivo, da massimizzare, unione delle funzioni F1 e F3, che assegna un punteggio al robot come descritto in F3, ma si basa solamente sulle informazioni relative al robot ad esso più vicino (come in F1).

Funzioni obiettivo globali

Ai fini dell'analisi dell'andamento dell'evoluzione e per valutare l'intero esperimento, utilizziamo delle funzioni obiettivo, definite *globali*, per ognuno dei due task. Queste ultime non incidono sul processo evolutivo dei robot, in quanto i robot non sono a conoscenza della valutazione complessiva che ottiene lo sciame durante l'esperimento. Ogni robot si baserà unicamente sul proprio score, valutato dalla funzione obiettivo *locale* descritta in precedenza.

Le funzioni obiettivo globali utilizzate sono richiamate durante il processo evolutivo dello sciame, ad ogni determinato numero di passi evolutivi, per avere un riferimento che permetta di analizzare le prestazioni dell'evoluzione nel tempo. Tramite queste funzioni globali siamo in grado di valutare le prestazioni globali dell'intero sciame di robot durante la risoluzione dello specifico task. Vengono utilizzate unicamente a scopo di analisi e discussione degli esperimenti. I risultati ottenuti verranno discussi all'interno del Capitolo 3.

Per la creazione delle funzioni obiettivo globali abbiamo preso come riferimento la funzione utilizzata per il task *Coverage with forbidden areas* studiato nell'articolo [Francesca et al., 2015] per la valutazione delle performance di

AutoMoDe-Chocolate, un metodo di progettazione automatica per sciami di robot da loro sviluppato.

Task 1 Per la valutazione del primo task, la funzione obiettivo globale viene calcolata come:

$$F_{G_{al}} = E_{al}[d(T)], \text{ da minimizzare}$$

dove $E_{al}[d(T)]$ indica la distanza, alla fine T del tempo a disposizione per completare il task, tra un insieme di punti sparsi in maniera casuale all'interno dell'arena ed i rispettivi robot più vicini.

Questa funzione verrà chiamata ogni determinato numero di T, come descritto successivamente all'interno della Sezione 2.5.

Lo pseudocodice della funzione obiettivo globale utilizzata per il task *copertura con arena libera* viene descritto dall'Algoritmo 1.

Algorithm 1 Funzione obiettivo globale, copertura con arena libera

```
1: distance ← 0
2: score ← 0
3: for i < 10000 do
4:     randomPoint ← GetRandomPoint()
5:     distance ← distance+GetClosestRobotDistanceFromPoint(randomPoint)
6:     i ← i + 1
7: end for
8: score ← distance/i
9: return score
```

La funzione *GetRandomPoint* ritorna un punto in una posizione casuale all'interno dell'arena, mentre la funzione *GetClosestRobotDistanceFromPoint* ritorna la distanza tra un punto e il robot più vicino ad esso.

Il punteggio globale viene calcolato distribuendo casualmente 10000 punti all'interno dell'arena e per ogni punto viene calcolata la distanza con il robot a lui più vicino. Il punteggio finale è dato dal rapporto tra la somma di tutte le distanze ottenute e il numero di punti utilizzati.

Task 2 Per la valutazione del secondo task, la funzione obiettivo globale viene calcolata come

$$F_{G_{zp}} = E_{zp}[d(T)], \text{ da minimizzare}$$

dove $E[d(T)]$ indica la distanza, alla fine T del tempo a disposizione per completare il task, tra un insieme di punti sparsi in maniera casuale all'interno

dell'arena, ad esclusione delle zone proibite, ed i rispettivi robot più vicini che non si trovano all'interno di un'area proibita. I robot che durante la valutazione si troveranno all'interno di una zona proibita non verranno considerati per il calcolo dello score globale.

Questa funzione verrà chiamata ogni determinato numero di trial, come descritto successivamente all'interno della Sezione 2.5.

Lo pseudocodice della funzione obiettivo globale utilizzata per il task *copertura con zone proibite* viene descritto dall'Algoritmo 2.

Algorithm 2 Funzione obiettivo globale, copertura con zone proibite

```
1: distance ← 0
2: score ← 0
3: for i < 10000 do
4:     randomPoint ← GetRandomPoint()
5:     while IsInAForbiddenArea(randomPoint) do
6:         randomPoint ← GetRandomPoint()
7:     end while
8:     distance ← distance +
        GetClosestRobotNotInAForbAreaDistFromPoint(randomPoint)
9:     i ← i + 1
10: end for
11: score ← distance/i
12: return score
```

La funzione *IsInAForbiddenArea* ritorna *true* se il punto dato in ingresso si trova all'interno di una zona proibita, *false* altrimenti. La funzione *GetClosestRobotNotInAForbAreaDistFromPoint* ritorna la distanza tra un punto casuale all'interno dell'arena ed il robot ad esso più vicino che non sia all'interno di una zona proibita. Un robot sarà identificato come essere all'interno di una zona proibita solamente se il suo baricentro si trova in una zona proibita.

Il punteggio globale sarà calcolato come per Task 1, con la differenza che i punti casuali non potranno essere posizionati all'interno delle zone proibite e i robot che si trovano all'interno di un'area proibita durante la valutazione non verranno considerati durante il calcolo.

2.3 Configurazione dei Controller dello Sciame di Robot

Il processo evolutivo online dello sciame di robot è eseguito e valutato tramite esperimenti simulati. I robot simulati utilizzati sono *foot-bot*, piccoli

robot di 17 cm di diametro sviluppati all'interno del progetto Swarmanoid [1]. Il foot-bot, mostrato in Figura 2.1, è una particolare configurazione di moduli basata sulla piattaforma robotica marXbot [Bonani et al., 2010].

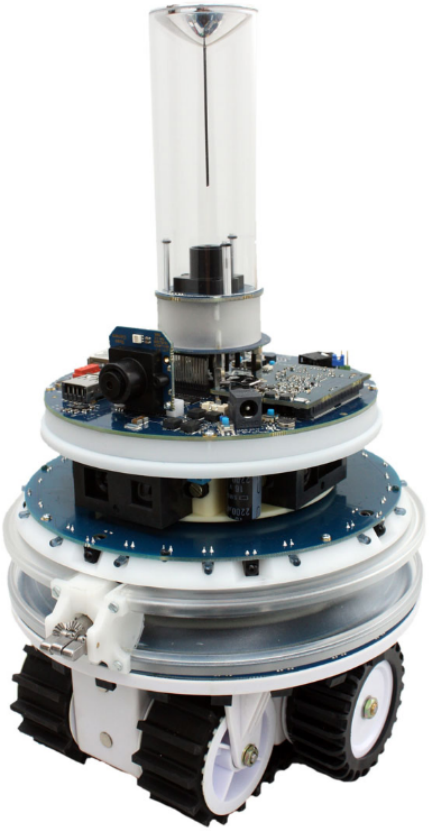

Figura 2.1: Foot-bot: robot utilizzato all'interno di questo lavoro durante la simulazione degli esperimenti

Sensori e Attuatori Esamineremo brevemente solo i sensori e gli attuatori equipaggiati sui robot che sono utili ai fini degli esperimenti, specificandone le caratteristiche:

- *differential steering*: comanda le 2 ruote del robot, posizionate una a destra ed una a sinistra rispetto al centro del robot. Tramite esso è quindi possibile impostare e leggere la velocità per ognuna delle stesse;

  Negli esperimenti relativi a questo lavoro, le velocità delle ruote sono impostate su zero o su un valore predefinito, per consentire di associare un valore binario al loro controllo. In questo modo, è possibile che il robot si

[1] http://www.swarmanoid.org/

muova in qualsiasi direzione semplicemente impostando la combinazione corretta di due valori binari.

- *range-and-bearing* (RAB): questo sistema permette al robot di effettuare una comunicazione localizzata, ovvero che un robot, oltre a poter ricevere dati da un altro robot, può individuare la posizione del mittente rispetto al proprio sistema di riferimento.

  I robot possono scambiare informazioni unicamente se vengono rilevati dal RAB, questo non avviene se la loro linea visiva è ostruita da un ostacolo. Inoltre, i robot hanno un range massimo limitato con cui poter scambiare dati, questo significa, che un robot sarà in grado di localizzare solamente gli altri robot all'interno di questo range.

  Il range and bearing, in questo caso, è utilizzato per ottenere la distanza e la direzione dei robot localizzati, rispetto al sistema di riferimento del robot.

  Ad ogni step, un robot riceve un numero variabile di messaggi dai robot vicini. Ai fini degli esperimenti effettuati per questo lavoro sarà importante identificare la distanza dell'origine del messaggio (indicata in cm) e l'angolo tra l'asse x locale del robot e la posizione della sorgente del messaggio.

- *proximity*: i sensori di prossimità sono 24, posizionati in maniera equamente distribuita sulla circonferenza attorno al corpo del robot. I sensori sono posizionati in ordine crescente in senso antiorario ad una distanza di $\frac{\pi}{12}$ l'uno dall'altro. Ogni sensore ha un range di visibilità di 10cm e restituisce una lettura composta da un angolo in radianti e un valore nell'intervallo [0,1] inversamente proporzionale alla distanza dall'oggetto rilevato. Il valore 0, relativo alla percezione di un sensore, corrisponde a nessun oggetto rilevato ed aumenterà all'avvicinarsi del robot ad un oggetto.

  Gli angoli dei sensori di prossimità non adottano il sistema di riferimento dell'arena, ma quello del robot.

- *positioning*: utilizziamo questo sensore per ottenere la posizione e l'orientazione del robot rispetto al sistema di riferimento dell'arena.

Per il secondo task è necessario equipaggiare un ulteriore tipo di sensori ai robot:

- *base_ground*: per rilevare le aree nere che indicano le zone proibite viene utilizzato questo sensore che è in grado di rilevare il colore del pavimento.

E' composto da 8 sensori, ad una distanza di $\frac{\pi}{4}$ l'uno dall'altro, che restituiscono la posizione espressa come vettore 2D derivante dal centro del robot e il valore 0 o 1, dove 0 significa nero (o grigio scuro) e 1 significa bianco (o grigio chiaro).

Il controller del robot è composto da un ciclo che, ad ogni step, alterna le fasi di *sensing* dell'ambiente, di *aggiornamento* e di *azione*. Più nel dettaglio, ad ogni istante di tempo vengono eseguite tre operazioni:

1. La lettura dei sensori viene codificata nei valori di input;
2. Viene aggiornato lo stato della rete;
3. Il valore dei nodi di output viene letto, codificato e utilizzato per operare sugli attuatori.

Binarizzazione dei valori dei nodi di input e output della rete Le letture dei sensori devono essere convertite in valori binari prima di essere applicate ai nodi di input e di output della BN. Per mappare i sensori negli input della BN utilizziamo quindi queste due metodologie:

- **Task 1**: per reti relative al primo task vengono utilizzati 4 nodi di input. Le informazioni vengono mappate ai nodi di input utilizzando un *codice Gray*[2] di 4 variabili booleane, dove ogni bit sarà dato in ingresso ai quattro nodi di input della rete. Il codice Gray è un sistema numerico binario dove due parole del codice successive differiscono di un solo bit.

  Ai nodi di input viene assegnato l'angolo indicato dal vettore risultante dato dalla somma vettoriale dei vettori ottenuti dai sensori *RAB* e *Proximity*. Tramite codice Gray indichiamo il valore relativo ad una delle 8 direzioni (N, NO, O, SO, S, SE, E, NE) del vettore risultante dei due sensori. Ogni direzione ha un range di 45°, centrato nella direzione di riferimento, rispetto al sistema di riferimento del robot. Il codice 0000 viene assegnato quando non viene rilevato niente dai sensori.

- **Task 2**: per reti relative al secondo task vengono utilizzati 5 nodi di input. In questo caso, la metodologia è la stessa del Task 1, ma al vettore risultante viene sommato, in aggiunta, il vettore ottenuto dalle rilevazioni del sensore *Base Ground*. In più, viene utilizzato un ulteriore nodo di input, collegato a quest'ultimo sensore, a cui verrà assegnato il valore 1 nel caso in cui venga rilevato un pavimento nero (zona proibita), 0 altrimenti.

[2]https://en.wikipedia.org/wiki/Gray_code

Per quanto riguarda i nodi di output delle reti booleane, in entrambi i task ne vengono utilizzati 2 connessi agli attuatori, ai quali viene associato un valore binario come segue:

$$wv = \begin{cases} 1 & \rightarrow \text{ ruota in movimento (v = 15 cm/s)} \\ 0 & \rightarrow \text{ ruota ferma (v = 0 cm/s)} \end{cases}$$

In questo modo, la forza degli stimoli percepiti non andrà ad influenzare la velocità di movimento del robot, in quanto ciascun attuatore è associato ad un solo nodo di uscita e convertito in un singolo valore binario.

2.4 Modello della Rete Booleana adottata

I robot dello sciame utilizzati negli esperimenti, saranno equipaggiati con controller basati su reti booleane (BN-robot). Le reti booleane utilizzate in questa tesi sono Random Boolean Network (RBN) dove verranno scelti casualmente alcuni nodi come nodi di input ed altri come nodi di output.

Durante la progettazione di una rete booleana l'accoppiamento della BN con il robot è fondamentale, ovvero la definizione del mapping tra sensori e input della rete, e tra output della rete e attuatori.

Le reti utilizzate sono descritte dai seguenti parametri principali:

- N : numero di nodi della rete;
- K : numero di connessioni entranti per ogni nodo. Da cui il numero di ingressi per ciascuna funzione booleana associata.
- I : numero di nodi di input della rete;
- O : numero di nodi di output della rete;
- $P \in [0, 1]$: probabilità di assegnare a 1 o 0 una entry della tabella della verità di un nodo durante la creazione della RBN;
- $Q \in [0, 1]$: probabilità di assegnare 1 o 0 al valore di un nodo come stato iniziale durante la creazione della RBN;

Inoltre, per la creazione delle reti dei robot abbiamo definito i seguenti vincoli:

- un nodo non può essere scelto sia come nodo di input che come nodo di output;

- i nodi di input non hanno nessun altro nodo in ingresso, il loro valore dipenderà solamente dai sensori dei robot;
- i nodi di input non hanno una tabella della verità dal momento che non hanno altri nodi in ingresso;
- Le connessioni iniziali tra i nodi sono generate random e senza auto-connessioni;
- le tabelle della verità hanno cardinalità pari a 2^K;
- i nodi sono inizializzati a 1 durante la creazione della rete con probabilità P;
- le entry della tabella della verità delle funzioni booleane di ogni nodo sono inizializzate con valore 1 con probabilità Q.

Le RBN sono quindi create scegliendo casualmente, tra tutti i nodi della rete, I nodi di input e O nodi di output. Verranno poi determinati i K nodi collegati in ingresso ad ogni nodo, ovvero le connessioni entranti (invarianti durante l'esperimento), i valori iniziali dei nodi e le relative funzioni booleane, attraverso l'inizializzazione casuale delle tabelle delle verità.

Le reti booleane che implementano il controller dei robot sono soggette ad un *aggiornamento sincrono* e *deterministico*. Queste reti sono progettate con tecniche di ricerca locali, ovvero una semplice discesa stocastica in cui una mossa può cambiare un valore nella tabella di verità di una funzione di un nodo; viene scelta una entry casuale nella tabella della verità di un nodo scelto casualmente e se la rete corrispondente ha una valutazione non peggiore di quella attuale viene memorizzata dal robot come migliore rete fino ad ora trovata.

2.5 Meccanismi di Aggiornamento per Sciami di Robot

Nella Sezione 1.2 abbiamo introdotto diversi approcci per lo sviluppo di meccanismi evolutivi, nel seguito di questo lavoro ci concentreremo su meccanismi evolutivi online.

Un meccanismo di aggiornamento online, o meccanismo evolutivo online, è un algoritmo che guida l'evoluzione delle istanze dei software di controllo dei robot, nel nostro caso di uno sciame di robot. I robot dello sciame sono inseriti in un ambiente sconosciuto e nel quale devono apprendere autonomamente ad eseguire un task, basandosi unicamente su informazioni locali.

Questi meccanismi sfruttano le informazioni date dalla valutazione, da parte della funzione obiettivo, del comportamento delle RBN dei robot, per ottenere uno sciame di robot autonomi in grado di adattare i propri software di controllo in maniera autonoma e autosufficiente per imparare a far fronte a diverse situazioni.

Lo scopo di questa tesi è, quindi, quello di progettare e analizzare diversi meccanismi di aggiornamento delle reti booleane che controllano i robot di uno sciame durante la loro evoluzione. I BN controller si dovranno adattare senza alcuna supervisione esterna, mentre i robot svolgono i loro compiti, e saranno valutati attraverso apposite funzioni obiettivo.

L'evoluzione, all'interno del nostri esperimenti, è dettata dai seguenti parametri:

- **Trial T**: si riferisce ad un passo evolutivo ed è composto da un determinato numero di step S del robot, in ognuno dei quali verrà eseguita una iterazione del ciclo di controllo del robot. Durante il Trial viene valutato il comportamento di ogni robot tramite la funzione obiettivo, e al termine del Trial ne fornirà lo score complessivo. Durante ogni T, i robot verranno valutati per la rete che stanno eseguendo, che sarà differente ad ogni Trial.

- **Evaluation Trial ET**: si riferisce ad un Trial di valutazione, eseguito ogni determinato numero di T. Durante un ET vengono valutati i robot eseguendo, per ogni robot, la migliore rete da esso trovata fino a quel momento, in più, viene valutato il comportamento dell'intero sciame di robot tramite la funzione obiettivo globale.

  E' importante precisare che, essendo questo studio basato su un approccio di robotica evolutiva online, il punteggio globale calcolato durante un ET non viene mai comunicato ai robot, ma viene utilizzato unicamente a fini dell'analisi dell'evoluzione e dei confronti.

- **Synchronization Trial ST**: si riferisce ad un Trial specifico per i meccanismi evolutivi che effettuano la sincronizzazione e che sostituisce l'ET. Durante un ST vengono valutati i robot tramite l'esecuzione su ogni robot della migliore rete trovata da tutti i robot fino a quel momento, ovvero tutti i robot eseguiranno il ST con la medesima rete.

  In più, viene valutato il comportamento dell'intero sciame di robot tramite la funzione obiettivo globale. Come per l'ET il punteggio globale non incide sull'evoluzione dei robot.

Il processo di ricerca termina una volta raggiunto un numero di ET o ST definito a priori. All'interno della Capitolo 3 verranno indicati i parametri utilizzati per gli esperimenti.

In questo lavoro sono stati sviluppati e analizzati 3 diversi meccanismi evolutivi chiamati: *Evoluzione senza sincronizzazione*, *Evoluzione con sincronizzazione* e *Evoluzione con sincronizzazione reversibile*.

2.5.1 Evoluzione senza Sincronizzazione

Questo meccanismo evolutivo consiste in una semplice *ricerca locale* della migliore configurazione della rete booleana che controlla i singoli robot dello sciame. Gli algoritmi di ricerca locale sono metodi che partono da una soluzione iniziale e tentano iterativamente di sostituire questa soluzione con una migliore che fa parte del vicinato della soluzione attuale [Blum and Roli, 2003].

In particolare, viene implementata la ricerca locale come una semplice discesa stocastica, dove ogni azione prevede un cambiamento di un valore della tabella delle verità della funzione booleana del nodo, a partire da una rete booleana generata casualmente (RBN). Ogni azione consiste in un flip da 0 a 1, o viceversa, di una entry della tabella della verità. La soluzione risultante viene accettata e mantenuta come nuova rete solamente se non è peggiore della soluzione attuale.

In questo studio, durante il processo evolutivo, la topologia della rete viene mantenuta inalterata.

I singoli robot, quindi, mantengono in memoria la rete che è stata valutata come migliore dalla funzione obiettivo, ovvero la rete con la fitness più alta, fino a quel momento. Ad ogni Trial, viene generata una nuova rete attraverso un flip di una entry della tabella della verità di un nodo scelto casualmente e questa rete viene mantenuta solamente se ottiene un punteggio uguale o maggiore della rete migliore attuale.

Questo approccio viene classificato come un approccio di *encapsuled evolution* [Eiben et al., 2010a]. Ogni robot esegue localmente un algoritmo evolutivo, valutando autonomamente una sottopopolazione delle configurazioni della rete sulla base della funzione obiettivo. Il miglioramento iterativo (ottimizzazione) dei controller è il risultato degli algoritmi evolutivi eseguiti sui singoli robot.

Da questo approccio, i software di controllo risultanti dai singoli robot saranno tipicamente eterogenei, nonostante possa accadere che, durante l'evoluzione, alcune reti convergano alla stessa configurazione.

In Figura 2.2 viene graficato l'approccio di encapsuled evolution tratto da [Eiben et al., 2010a].

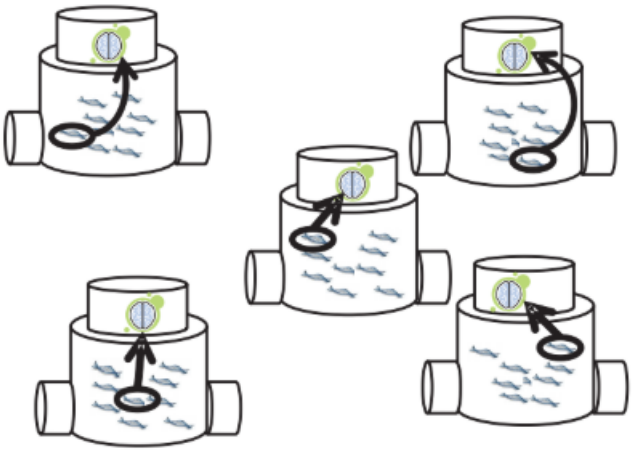

Figura 2.2: Evoluzione senza sincronizzazione: evoluzione online incapsulata in un gruppo di robot, in cui ogni robot esegue un algoritmo evolutivo onboard. Il processo evolutivo non richiede comunicazione e interazione tra robot. Immagine tratta da [Eiben et al., 2010a] e riprodotta con il permesso degli autori.

Questo semplice schema definisce il nucleo del meccanismo evolutivo, ad esso, come anticipato in precedenza, sono aggiunti dei trial per la rivalutazione delle reti migliori. Più precisamente, all'interno del processo evolutivo dello sciame di robot, ci saranno dei trial di valutazione ET, durante i quali viene effettuato un processo di rivalutazione delle reti migliori di ogni robot. Durante questi ET ogni robot esegue la sua migliore rete ottenuta fino a quel momento. *Il valore di fitness/il punteggio risultante da questa nuova valutazione sovrascrive il valore precedente della medesima rete.*

Questo è utile perché una rete potrebbe aver ottenuto un buon punteggio in determinate condizioni favorevoli, ma non essere adatta al task in questione. Ogni valutazione, infatti, inizia nella posizione finale della valutazione precedente, portando a condizioni di valutazione anche molto diverse da una rete all'altra di uno stesso robot. Questo implica che la valutazione di una rete può essere fuorviante, semplicemente a causa delle condizioni di partenza fortunate / sfortunate.

Effettuare una rivalutazione ci permette di verificare se la valutazione della rete ritenuta migliore è una condizione stabile e corretta, poiché preferiamo controller con comportamenti solidi, ovvero che valutati più volte ottengano tipicamente buoni risultati. Tuttavia, può comportare un inconveniente intrinseco, in quanto reti buone possono venire sostituite da reti con punteggi inferiori, ma che sono stati fortunati in specifiche condizioni favorevoli. Questo inconveniente dovrebbe risolversi in quanto una rete fortunata, in media non

da buoni risultati, di conseguenza non sopravviverà alla rivalutazione, evitando che il processo adattivo si blocchi su una rete non buona.

L'Algoritmo 3 mostra lo pseudocodice del meccanismo fino ad ora descritto.

Algorithm 3 Meccanismo evolutivo: evoluzione senza sincronizzazione

Require: $EXPERIMENT\_LENGTH$ durata dell'esperimento, F funzione obiettivo, $EVAL\_TIME$ numero di trial tra le più rivalutazioni e valutazioni globali

```
1: bn ← GenerateRandomNetwork()
2: bestBn ← bn
3: bestScore ← 0
4: trial ← 0
5: while trial < EXPERIMENT_LENGTH do
6:     score ← EvaluateBN(F, bn)
7:     if score >= bestScore then
8:         bestScore ← score
9:         bestBn ← bn
10:    end if
11:    if trial%EVAL_TIME = 0 then                ▷ Evaluation Trial
12:        bestScore ← EvaluateBN(F, bestBn)
13:    end if
14:    bn ← DoARandomFlip(bestBn)
15:    trial ← trial + 1
16: end while
17: return bestBn
```

La funzione *GenerateRandomNetwork* genera una nuova rete booleana inizializzata in maniera casuale. La funzione *EvaluateBn* ottiene il punteggio effettuato da una rete in un trial, a seconda della funzione obiettivo data in ingresso. La funzione *DoARandomFlip* applica un flip ad una entry casuale della tabella della verità di un nodo casuale della rete data in ingresso, e ritorna la nuova rete.

Un approccio simile all'*Evoluzione senza sincronizzazione* è l'algoritmo evolutivo (1+1)-online basato sulla classica (1+1) Evolution Strategy (ES), studiato in [Bredeche et al., 2009], e che rientra negli approcci incapsulati. Nel loro caso, l'approccio è basato su ANN e l'evoluzione riguarda unicamente i pesi della rete. Come nel nostro studio, ogni robot mantiene un solo individuo per robot e la variazione degli individui è limitata alla mutazione. Il nuovo in-

dividuo sostituisce il migliore solamente se ottiene una fitness maggiore. Viene utilizzata, anche in questo algoritmo, una rivalutazione delle reti migliori, in questo caso con una certa probabilità, in cui il punteggio ottenuto da questa rivalutazione viene sostituito al precedente. In aggiunta al nostro meccanismo, nel loro approccio viene utilizzata un valore σ per la gestione della convergenza prematura a un ottimo locale di cui risentono questo tipo di approcci. In particolare, σ è impostato su un minimo predefinito per promuovere la ricerca locale ogni volta che viene memorizzata una nuova rete (cioè quando lo sfidante supera il campione). Quindi, σ aumenta gradualmente fino ad un valore massimo (cioè la ricerca si sposta verso la ricerca globale) mentre il campione supera i suoi figli. Se la ricerca locale porta a miglioramenti, σ rimane basso, favorendo così la ricerca locale. Se non viene apportato alcun miglioramento su base locale, a causa di un paesaggio neutro o di un ottimale locale, i valori crescenti di σ assicurano che la ricerca si sposterà in nuove regioni nello spazio di ricerca.

In più, introducono un tempo di recupero dal periodo di valutazione precedente, durante il quale il comportamento del robot non viene preso in considerazione per il calcolo del valore di fitness. Ciò favorisce i genomi che sono efficaci sia per uscire dai guai durante la fase di recupero, a causa delle cattive situazioni in cui si erano trovati in partenza dovute al comportamento della rete precedente, sia per mostrare comportamenti efficienti durante la fase di valutazione.

2.5.2 Evoluzione con Sincronizzazione

La struttura alla base di questo meccanismo è la stessa del meccanismo di aggiornamento "*Evoluzione senza sincronizzazione*". Ogni robot mantiene la rete migliore fino ad ora valutata e per ogni nuova valutazione viene generata una nuova rete, tramite un flip di una entry della tabella della verità di un nodo della rete scelto in maniera casuale, e se la nuova rete è migliore della migliore viene sostituita ad essa.

In aggiunta a questo comportamento di base, viene effettuata una sincronizzazione della rete migliore dello sciame ogni determinato numero di Trial. Questo specifico trial di sincronizzazione è chiamato Synchronization Trial ST, introdotto all'interno della Sezione 2.5, durante il quale ad ogni robot dello sciame viene assegnata la rete migliore (con fitness più alta) fino ad ora trovata tra tutte quelle valutate da ogni robot e viene effettuata la valutazione dei robot con tale rete.

Al termine della sincronizzazione ogni robot sostituirà la propria migliore rete con quella che gli è stata passata durante la sincronizzazione e, dal Trial

successivo, ogni robot cercherà di evolvere questa nuova rete, attraverso i flip, fino alla successiva sincronizzazione.

Questo meccanismo evolutivo può essere classificato come un approccio ibrido, in quanto combina una metodologia di evoluzione incapsulata ad una distribuita. Infatti, mentre ogni robot esplora possibili soluzioni on-board, durante il ST la rete migliore di tutte viene condivisa a tutti i robot.

Solitamente approcci evolutivi distribuiti o ibridi utilizzano una sincronizzazione locale [Bianco and Nolfi, 2004, Karafotias et al., 2011, Silva et al., 2015b], in questo caso invece la sincronizzazione avviene globalmente in quanto lo sciame di robot si sincronizza nello stesso momento sulla stessa rete, ritenuta la migliore trovata fino a quel momento.

Ciò che si vuole ottenere da una sincronizzazione globale consiste nel riuscire ad accelerare la convergenza ad una soluzione ottima, dal momento che tutti i robot si dedicheranno all'esplorazione di una stessa sottopopolazione di soluzioni a partire dalla soluzione con maggiore fitness trovata fino al momento della sincronizzazione.

Per la gestione della sincronizzazione globale viene utilizzato un *master*, similmente all'approccio descritto in [Eiben et al., 2010a]. Nel suo approccio ogni robot trasporta un genotipo e invia informazioni riguardanti la propria fitness al master. L'evoluzione è gestita da un computer master esterno che esegue tutti gli operatori evolutivi per la selezione e la riproduzione. E' il master, quindi, che decide, utilizzando le informazioni globali che possiede, quali controller dei robot devono essere ricombinati e / o mutati e quali dovrebbero essere sostituiti con nuovi controller.

Il nostro approccio è leggermente diverso, ogni robot mantiene la propria rete migliore in memoria ed esegue gli operatori evolutivi (mutazione della rete tramite flip), il master viene utilizzato solamente durante la sincronizzazione per distribuire a tutti i robot il controller con fitness maggiore, dopo aver ottenuto dai robot le loro reti migliori.

Viene utilizzato un master perché, a differenza di altre soluzioni completamente decentralizzate, permette in maniera semplice di effettuare una sincronizzazione globale simultanea della rete migliore su tutti i robot dello sciame, senza ricadere in possibili problemi dati da uno scambio tra robot. Ad esempio, si potrebbero avere problemi nel caso in cui uno o più robot si ritrovino (in particolare in arene vaste) in aree dell'arena "isolate", ovvero in aree in cui i robot non sarebbero visibili da nessun altro robot dello sciame. In questo caso non sarebbero in grado di trasmettere la loro rete e/o ricevere la rete migliore.

In Figura 2.3 viene graficata l'interazione tra robot e master tramite una immagine tratta da [Eiben et al., 2010a], ma con descrizione adattata al caso di

studio in esame.

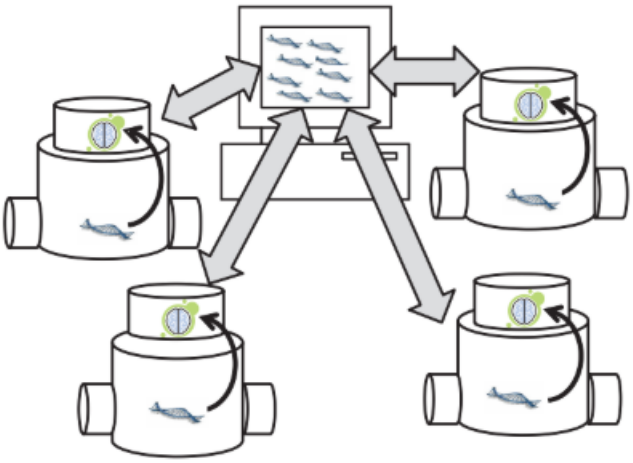

Figura 2.3: Evoluzione con sincronizzazione: evoluzione online ibrida di un gruppo di robot, in cui ogni robot esegue un algoritmo evolutivo on-board, e un master gestisce, durante la sincronizzazione, lo scambio tra i robot della migliore rete trovata dallo sciame. La comunicazione tra i robot e il master è necessaria per la trasmissione delle reti booleane e le relative valutazioni. Queste interazioni sono indicate dalle frecce grigie nella figura. Immagine tratta da [Eiben et al., 2010a] e riprodotta con il permesso degli autori.

L'Algoritmo 4 mostra lo pseudocodice, relativo ad un robot, del meccanismo fino ad ora descritto.

La funzione *GenerateRandomNetwork* genera una nuova rete booleana inizializzata in maniera casuale. La funzione *EvaluateBn* ottiene il punteggio effettuato da una rete in un trial, a seconda della funzione obiettivo data in ingresso. La funzione *GetBestSwarmBN* ottiene dal master la rete migliore trovata fino ad ora dai robot dello sciame. La funzione *DoARandomFlip* applica un flip ad una entry casuale della tabella della verità di un nodo casuale della rete data in ingresso, e ritorna la nuova rete.

Una meccanismo evolutivo simile al nostro, ma completamente decentralizzato, viene presentato in [Simoes and Barone, 2002], in cui i robot eseguono

Algorithm 4 Meccanismo evolutivo: Evoluzione con sincronizzazione

Require: $EXPERIMENT\_LENGTH$ durata dell'esperimento, F funzione obiettivo, $SINC\_TIME$ numero di trial tra una sincronizzazione e l'altra

1: $bn \leftarrow GenerateRandomNetwork()$
2: $bestBn \leftarrow bn$
3: $bestScore \leftarrow 0$
4: $trial \leftarrow 0$
5: **while** $trial < EXPERIMENT\_LENGTH$ **do**
6: $\quad score \leftarrow EvaluateBN(F, bn)$
7: $\quad$ **if** $score >= bestScore$ **then**
8: $\quad\quad bestScore \leftarrow score$
9: $\quad\quad bestBn \leftarrow bn$
10: $\quad$ **end if**
11: $\quad$ **if** $trial\%SINC\_TIME = 0$ **then** ▷ Synchronization Trial
12: $\quad\quad bestBn \leftarrow GetBestSwarmBN()$
13: $\quad\quad bestScore \leftarrow EvaluateBN(F, bestBn)$
14: $\quad$ **end if**
15: $\quad bn \leftarrow DoARandomFlip(bestBn)$
16: $\quad trial \leftarrow trial + 1$
17: **end while**
18: **return** $bestBn$

una procedura ciclica composta da una fase di esecuzione, in cui tentano di eseguire il task, e una fase di accoppiamento sincronizzata che viene attivata da un timer interno. Durante la fase di accoppiamento il robot con la fitness maggiore invia il proprio genoma a tutti i restanti robot via radio. Quindi ciascuno dei restanti robot inizia una fase di crossover, in cui combinano il proprio genoma con quello ricevuto dal miglior robot. Successivamente, i bit di questo genoma risultante vengono selezionati casualmente per essere mutati (invertiti).

2.5.3 Evoluzione con Sincronizzazione Reversibile

Questo meccanismo evolutivo è una variazione del meccanismo *Evoluzione con Sincronizzazione*, con la differenza che la sincronizzazione può essere reversibile.

In questo approccio, infatti, ogni robot durante il Synchronization Trial ottiene dal master la migliore rete trovata fino a quel momento dallo sciame di robot e, dopo l'esecuzione e la relativa valutazione della rete per ogni robot, questa rete sostituirà la migliore di un robot solamente se, durante il trial di sincronizzazione, questa nuova rete ha ottenuto un punteggio maggiore della rete migliore memorizzata dal robot stesso.

In questo modo, un robot non viene vincolato a cambiare rete se quest'ultima ha ottenuto risultati peggiori di quella per lui migliore, come succedeva con il meccanismo *Evoluzione con Sincronizzazione*. Con questo approccio, quindi, si vuole evitare di sostituire le BN a robot in condizioni diverse dal robot che ha passato la rete al resto dello sciame e per i quali questa BN non è la soluzione migliore. Infatti, i robot potrebbero trovarsi in condizioni differenti all'interno dell'arena, per cui la rete passata potrebbe essere valida per un robot, ma non per altri. In più, in questo modo, si cerca di evitare che una BN fortunata venga memorizzata da tutto lo sciame, in quanto una rete potrebbe aver ottenuto un buon punteggio solamente in determinate condizioni favorevoli, nonostante non sia adatta al task in questione.

L'Algoritmo 5 mostra lo pseudocodice del meccanismo fino ad ora descritto.

Algorithm 5 Meccanismo evolutivo: evoluzione con sincronizzazione reversibile

Require: $EXPERIMENT\_LENGTH$ durata dell'esperimento, F funzione obiettivo, $SINC\_TIME$ numero di trial tra una sincronizzazione e l'altra

```
1: bn ← GenerateRandomNetwork()
2: bestBn ← bn
3: bestScore ← 0
4: trial ← 0
5: while trial < EXPERIMENT_LENGTH do
6:     score ← EvaluateBN(F, bn)
7:     if score >= bestScore then
8:         bestScore ← score
9:         bestBn ← bn
10:    end if
11:    if trial%SINC_TIME = 0 then                ▷ Synchronization Trial
12:        bn ← GetBestSwarmBN()
13:        score ← EvaluateBN(F, bn)
14:        if score >= bestScore then
15:            bestScore ← score
16:            bestBn ← bn
17:        end if
18:    end if
19:    bn ← DoARandomFlip(bestBn)
20:    trial ← trial + 1
21: end while
22: return bestBn
```

La funzione *GenerateRandomNetwork* genera una nuova rete booleana inizializzata in maniera casuale. La funzione *EvaluateBn* ottiene il punteggio effettuato da una rete in un trial, a seconda della funzione obiettivo data in ingresso. La funzione *GetBestSwarmBN* ottiene dal master la rete migliore trovata fino ad ora dai robot dello sciame. La funzione *DoARandomFlip* applica un flip ad una entry casuale della tabella della verità di un nodo casuale della rete data in ingresso, e ritorna la nuova rete.

Capitolo 3

Analisi dei Meccanismi

In questo capitolo verrà descritta la configurazione degli esperimenti effettuati. In particolare, verranno illustrate le arene, relative ai due task, all'interno delle quali i robot opereranno, in più, verranno indicati i parametri utilizzati per la realizzazione delle reti booleane e i parametri riguardanti la durata degli esperimenti.

Infine, verranno illustrati, testati, analizzati e discussi i risultati ottenuti dagli esperimenti sulle reti sviluppate utilizzando le metodologie evolutive descritte all'interno del Capitolo 2.

3.1 Configurazione degli Esperimenti

Gli esperimenti svolti sono stati eseguiti in simulazione tramite l'utilizzo del simulatore ARGoS[1], progettato per simulare esperimenti che coinvolgono un grande numero di robot di diversi tipi [Pinciroli et al., 2012]. ARGoS è il primo simulatore multi-robot che è allo stesso tempo efficiente (prestazioni veloci con molti robot) e flessibile (altamente personalizzabile per esperimenti specifici). Per questo motivo è particolarmente adatto per l'utilizzo in simulazione di sciami di robot.

Per l'implementazione dei controller dei robot è stato utilizzato il linguaggio Lua[2], mentre la simulazione viene controllata attraverso *loop function* utilizzando il linguaggio C++.

Una *loop function* è una porzione di codice in grado di controllare e modificare lo stato della simulazione, ad esempio, all'interno del nostro lavoro vengono utilizzate per controllare la durata degli esperimenti, gestire le tem-

[1] https://www.argos-sim.info/
[2] https://www.lua.org/start.html

pistiche delle valutazioni globali e delle sincronizzazioni, e per il calcolo della funzione obiettivo globale e la stampa dei risultati.

All'interno dei nostri esperimenti, come descritto all'interno della sezione 2.3, i robot simulati sono modellati come foot-bot ed ogni robot è dotato di sensori per il rilevamento degli ostacoli e del pavimento e per la comunicazione tra i robot.

Descrizione delle arene Durante i nostri esperimenti, un gruppo di trenta robot opera all'interno di un'arena quadrata circondata da mura. La dimensione dell'arena è stata scelta, sulla base della quantità di robot utilizzati, di una dimensione di 5 x 5 metri.

Le arene utilizzate presentano le medesime caratteristiche ad eccezione della pavimentazione utilizzata.

Arena libera L'arena utilizzata per l'esecuzione del Task 1 è chiamata 'arena libera' ed è composta da un'area quadrata con una pavimentazione grigia e uniforme.

In figura 3.1 è mostrata un'immagine dell'arena.

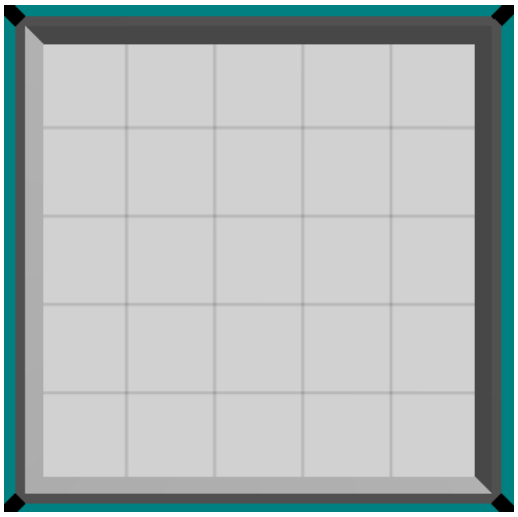

Figura 3.1: Arena libera utilizzata per il Task 1

Arena con zone proibite L'arena utilizzata per l'esecuzione del Task 2 è invece chiamata 'arena con zone proibite' in quanto all'interno sono presenti tre zone proibite rappresentate da quadrati di colore nero. Queste zone sono posizionate su tre angoli dell'arena, come mostrato in figura 3.2, escludendo l'angolo dal quale partono i robot durante la simulazione, ed hanno una dimensione pari a 1.5x1.5m. La dimensione delle aree proibite ai robot sono state

scelte tenendo in considerazione la dimensione dell'arena e il numero di robot utilizzati per gli esperimenti, in modo tale da dare la possibilità ai robot di espandersi correttamente all'interno dell'arena imparando ad evitare le zone proibite.

In figura 3.2 è mostrata un'immagine dell'arena.

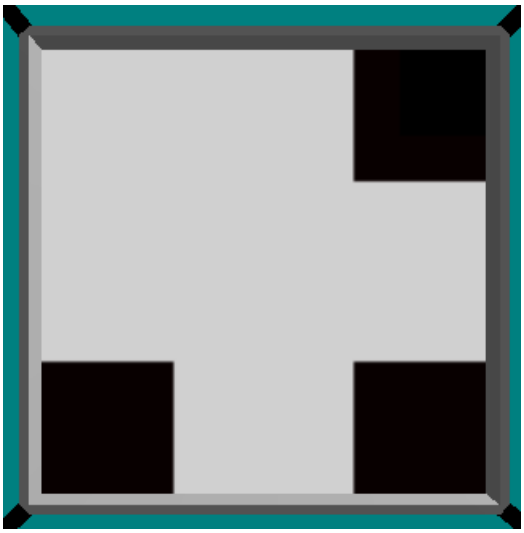

Figura 3.2: Arena con zone proibite utilizzata per il Task 2

Tutte le simulazioni iniziano con i robot dello sciame posizionati in maniera casuale nell'angolo in alto a sinistra dell'arena, come mostrato dalle figure 3.3 e 3.4, in modo tale da spronare i robot ad espandendosi, allontanandosi tra di loro il più possibile all'interno dell'arena.

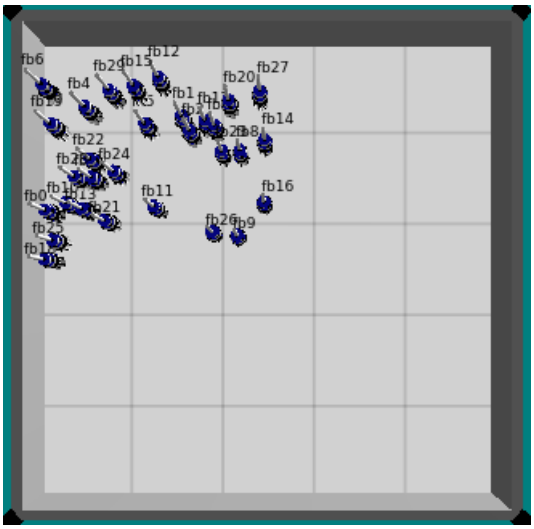

Figura 3.3: Stato iniziale dei robot all'inizio della simulazione con il Task 1

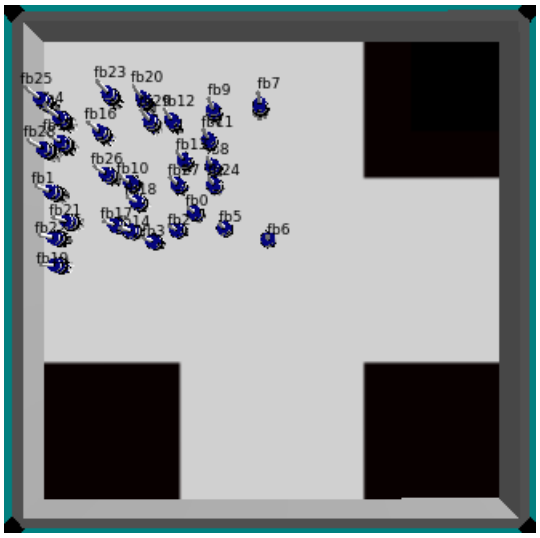

Figura 3.4: Stato iniziale dei robot all'inizio della simulazione con il Task 2

Parametri delle simulazioni Prima di tutto, è stato ritenuto opportuno regolare, tramite diversi test, i parametri e la durata dell'esperimento, in modo tale da consentire allo sciame di robot di evolversi sufficientemente prima del termine dell'esperimento permettendogli di imparare il compito assegnato.

Gli esperimenti descritti sono stati effettuati utilizzando la seguente configurazione di parametri:

- Numero di S per ogni T: 1200

- Numero di Trial tra due ET o ST: 40
- Numero di ET o ST per simulazione: 40
- Numero di robot: 30
- Simulazioni: 5 per esperimento

Ogni *Step* ha una durata di 10ms, in questo modo, con 1200 Step si ottiene un *Trial* della durata di 120 secondi. Considerando le dimensioni dell'arena e la velocità delle ruote dei robot, abbiamo stimato che questo tempo è sufficientemente lungo per permettere allo sciame di svolgere entrambi i task studiati. Ogni robot valuta 40 reti tra due ET o ST, memorizzando tra esse la migliore. I 40 ET o ST permettono di analizzare l'andamento dello sciame nel tempo e di valutare l'effettiva efficacia della funzione obiettivo utilizzata, in più, durante i 40 ST viene effettuata le sincronizzazione globale della migliore rete.
Il numero di soluzioni esplorate è dato dalla combinazione dei parametri indicati e si riferisce a circa $40 \cdot 40 \cdot 30 = 48000$ reti esplorate dallo sciame di robot. Non è, tuttavia, garantito che una stessa rete non venga valutata più volte.
Con l'utilizzo di questi parametri, ogni esperimento avrà una durata di $120 \times 40 \times 41 = 196.800$ secondi $\approx$ 2 giorni, 6 ore e 40 minuti. Questo tempo viene ridotto eseguendo la simulazione in modalità batch (senza visualizzazione). In questo modo i tempi dell'esperimento si riducono ad una durata che varia dalle 4 alle 8 ore, a seconda della funzione obiettivo e del dispositivo utilizzato.
Le simulazioni sono state eseguite su tre dispositivi con le seguenti caratteristiche:

- processore Intel Core i5-2400 CPU @ 3.10GHz, RAM 8GB DDR3, archiviazione Crucial SSD Mx500 500GB, Western Digital Blue Desktop HDD 500GB;
- processore Intel Core i5-6198DU CPU @ 2.30GHz, RAM 12GB DDR3, archiviazione Seagate ST1000LM024 HDD 1TB;
- processore Intel Core i7-4710HQ CPU @ 2.50GHz, RAM 16GB DDR3, archiviazione Western Digital Blue HDD 1TB.

Tutte le simulazioni sono state effettuate su sistema operativo Windows 10 utilizzando il layer di compatibilità Windows Subsystem for Linux.

Parametri delle RBN Il controller dei robot è composto da una RBN. Queste reti sono definite da alcuni parametri che devono essere definiti a priori all'interno dei nostri esperimenti.

I parametri utilizzati per le reti booleane che controllano i robot dello sciame sono i seguenti:

- N: 20;
- K: 2;
- I - Task 1: 4;
- I - Task 2: 5;
- O: 2;
- P: 0.5;
- Q: 0.5.

Ogni rete è composta da un numero di nodi pari a 20, poiché questa dimensione fornisce un compromesso tra il costo di calcolo nella simulazione della rete e la dimensione dello spazio degli stati. Come descritto nella sezione 2.3, i nodi di input della rete booleana sono connessi ai sensori del robot e variano di numero a seconda del task da compiere, mentre i nodi di output sono connessi agli attuatori che comandano le 2 ruote del robot. Ogni nodo, ad eccezione dei nodi di input, avrà 2 nodi collegati in ingresso. Le connessioni iniziali tra i nodi vengono generate casualmente (nessuna auto-connessione) e vengono mantenute fisse durante la ricerca. Le funzioni booleane iniziali dei nodi vengono generate impostando i valori 0/1 nelle tabelle di verità in maniera casuale. I valori iniziali e le entry della tabella della verità di ogni nodo sono inizializzati ad 1 con una probabilità del 50%. Una tabella della verità è composta da $2^2 = 4$ entry.

3.2 Esperimenti

In questa tesi siamo principalmente interessati a determinare se i meccanismi evolutivi studiati permettono una evoluzione dei controller in grado di risolvere il compito specificato, verificandone la qualità della soluzione ottenuta e valutando quale meccanismo evolutivo possa essere maggiormente di aiuto per un processo di apprendimento automatico online.

Per ogni meccanismo evolutivo analizzato, conduciamo 5 esperimenti evolutivi indipendenti, ognuno corrispondente ad una posizione e ad una BN iniziali differenti per i robot dello sciame. In ogni esperimento si addestrano i robot dello sciame mediante la ricerca locale in un ambiente simulato. Gli esperimenti continuano fino a quando non viene raggiunto il numero massimo di valutazioni globali, ovvero 40. Ogni esperimento dura 4/8 ore di tempo simulato.

Per ogni task e per ogni meccanismo di aggiornamento viene mostrato l'andamento degli score globali relativi alle 40 valutazioni effettuate durante l'esperimento. Vengono poi recuperati il valore massimo, il minimo e la media dei punteggi effettuati dai singoli robot durante una valutazione globale. Questi valori vengono utilizzati per la creazione di tre diversi grafici che indicano rispettivamente lo score medio, minimo e massimo effettuato dai robot durante il calcolo di ogni score globale. In questo modo si è in grado di valutare se il punteggio ottenuto dai singoli robot è proporzionale al punteggio globale ottenuto dallo sciame, per poter analizzare l'effettiva efficacia del meccanismo evolutivo e della funzione obiettivo utilizzati durante l'addestramento per la risoluzione di un task.

Dopo un processo di *trial and error* effettuato durante le fasi iniziali di sviluppo delle reti, è stata individuata una stima del punteggio ottimale globale dello sciame di robot per entrambi i task, che equivale a:

- $\sigma_1 = 0,38$ per Task 1;

- $\sigma_2 = 0,35$ per Task 2;

Questi valori sono strettamente correlati ai nostri parametri, ovvero dimensione dell'arena, numero di robot e numero di punti posizionati random e utilizzati per le valutazioni globali dello sciame. Per quanto riguarda σ_2 dipenderà anche dalla dimensione e dal numero di zone proibite.

Quindi σ_1 e σ_2 possono ritenersi ottimali unicamente per il nostro scenario.

All'interno delle figure 3.5 e 3.6 vengono mostrati lo stato finale di una simulazione che ottiene un punteggio ottimo e lo stato finale di una simulazione con punteggio buono, ma non ottimale, relativi al Task 1.

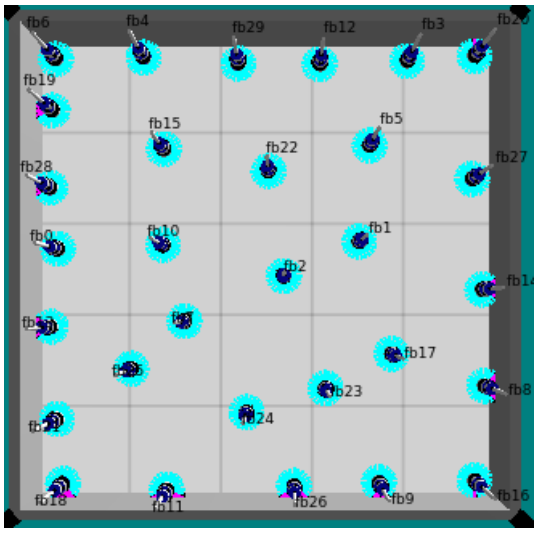

Figura 3.5: Stato finale dei robot al termine dell'addestramento del Task 1, con score globale finale pari a 0.382772

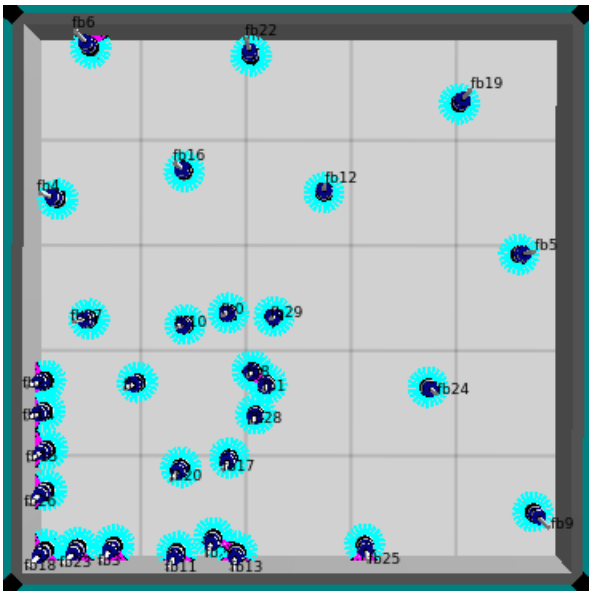

Figura 3.6: Stato finale dei robot al termine dell'addestramento del Task 1, con score globale finale pari a 0.475408

In figura 3.7 e 3.8 sono, invece, mostrati rispettivamente lo stato finale dello sciame di robot di una simulazione con punteggio ottimo e lo stato finale di una simulazione con punteggio finale non ottimale, relativi al Task 2.

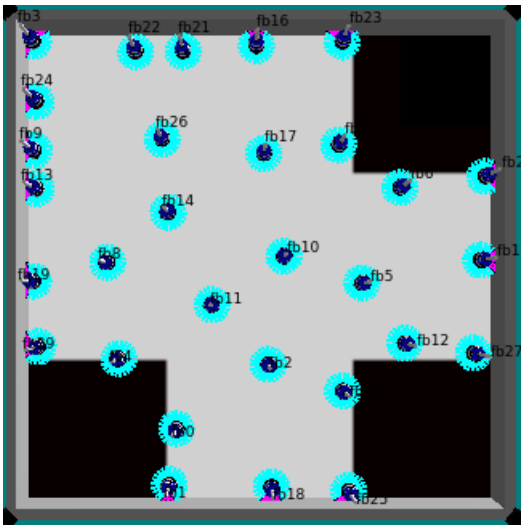

Figura 3.7: Stato finale dei robot al termine dell'addestramento del Task 2, con score globale finale pari a 0.350467

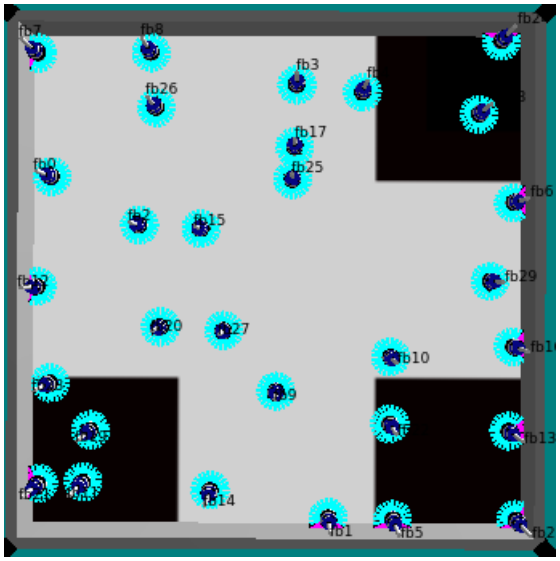

Figura 3.8: Stato finale dei robot al termine dell'addestramento del Task 2, con score globale finale pari a 0.439434

Negli esperimenti condotti in questa tesi verrà utilizzato un approccio di evoluzione online orchestrata da un master. Come già introdotto all'interno della sezione 2.5.2, questo approccio si basa sull'esistenza di una entità centrale che gestirà l'evoluzione dei robot, mentre questi ultimi sono in esecuzione in maniera online.

I robot avranno il compito di imparare autonomamente a risolvere due task che implicano un insieme di azioni in ambienti di distinta complessità. I diversi task richiedono azioni come l'espansione all'interno dell'arena, la capacità di evitare ostacoli e la capacità di non navigare in zone proibite. Questo implica che i robot dovranno imparare ad equilibrare le diverse azioni dipendenti dal task richiesto.

Di seguito verranno mostrati i risultati ottenuti dagli esperimenti effettuati, ordinati a partire dal migliore al peggiore score globale ottenuto dallo sciame al termine dell'addestramento. Per ogni meccanismo evolutivo analizzeremo gli andamenti dei risultati relativi alle esecuzioni degli esperimenti per ogni task assegnato.

3.2.1 Evoluzione senza Sincronizzazione

Task 1

Le figure 3.9, 3.10, 3.11, 3.12 e 3.13 mostrano i risultati ottenuti dagli esperimenti relativi al Task 1 utilizzando il meccanismo evolutivo senza sincronizzazione.

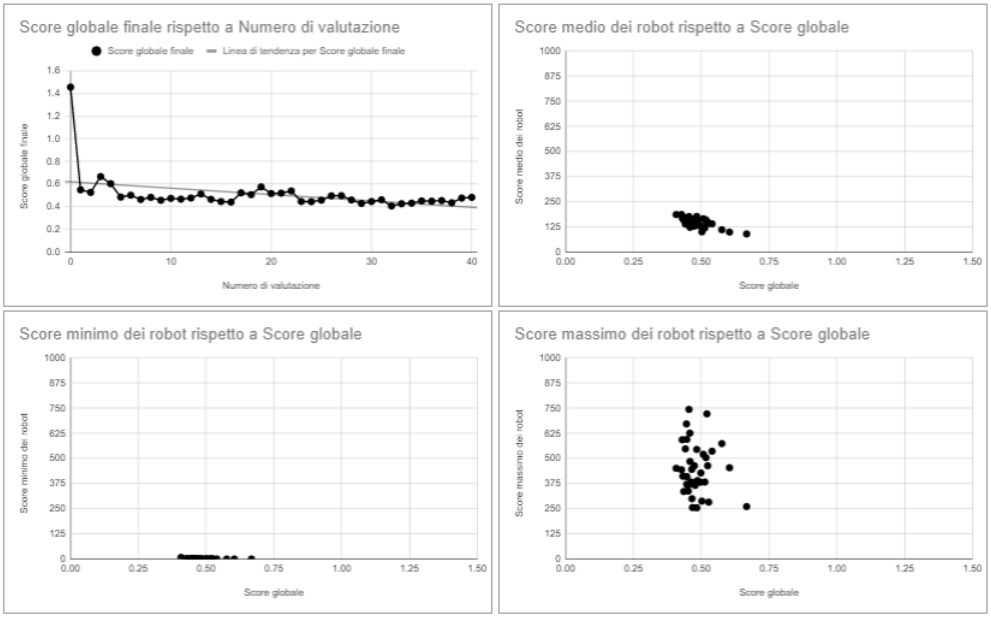

Figura 3.9: Andamento dei risultati della prima esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo senza sincronizzazione

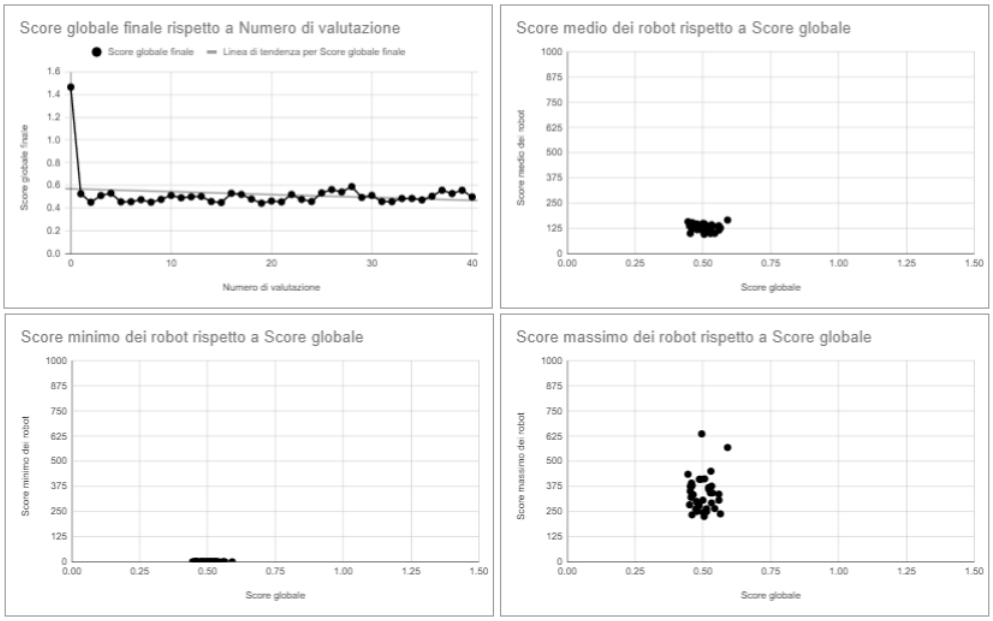

Figura 3.10: Andamento dei risultati della seconda esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo senza sincronizzazione

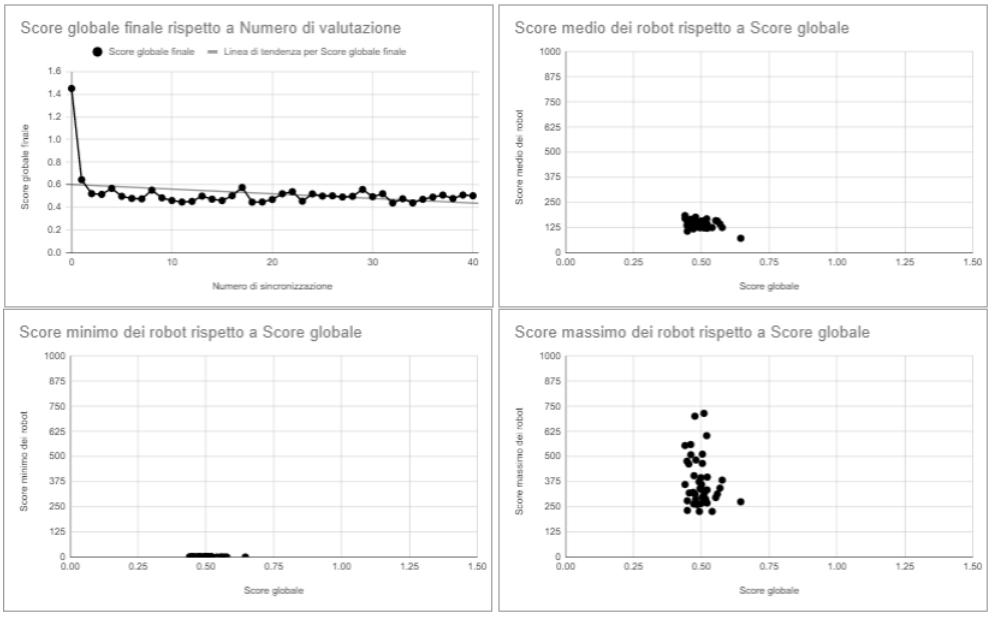

Figura 3.11: Andamento dei risultati della terza esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo senza sincronizzazione

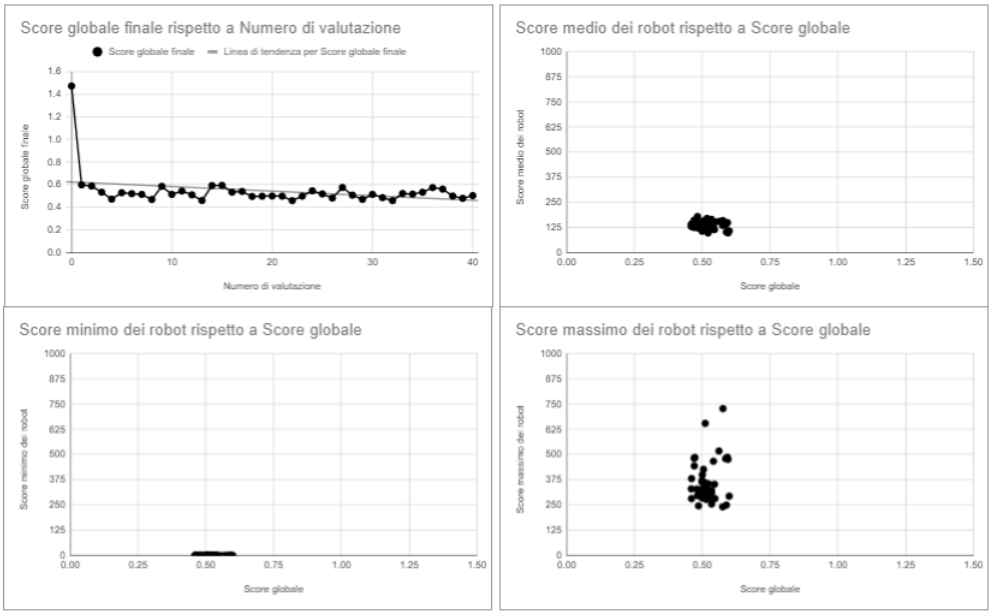

Figura 3.12: Andamento dei risultati della quarta esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo senza sincronizzazione

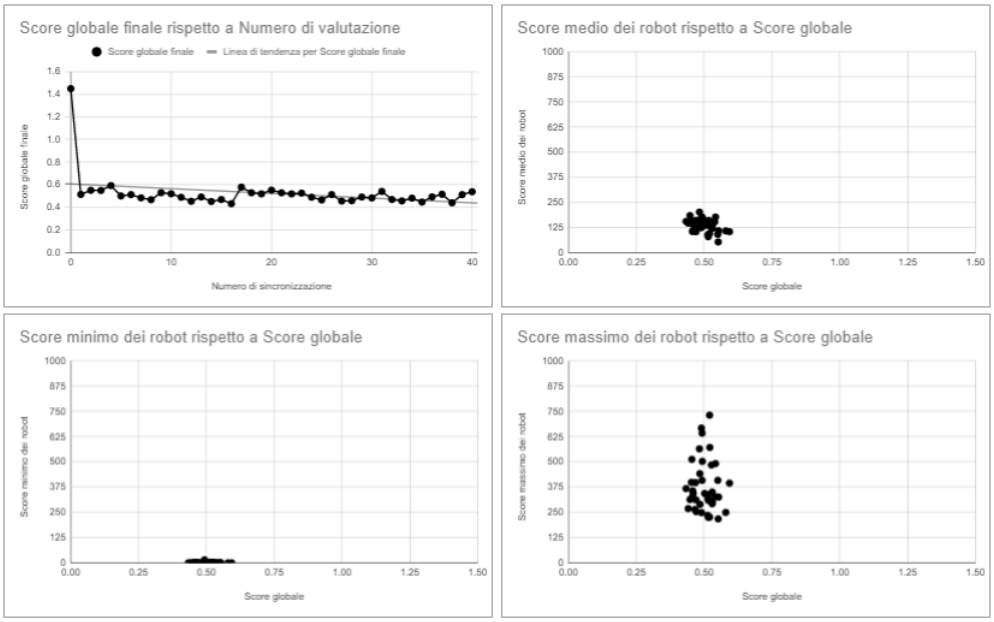

Figura 3.13: Andamento dei risultati della quinta esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo senza sincronizzazione

Task 2

Le figure 3.14, 3.15, 3.16, 3.17 e 3.18 mostrano i risultati ottenuti dagli esperimenti relativi al Task 2 utilizzando il meccanismo evolutivo senza sincronizzazione.

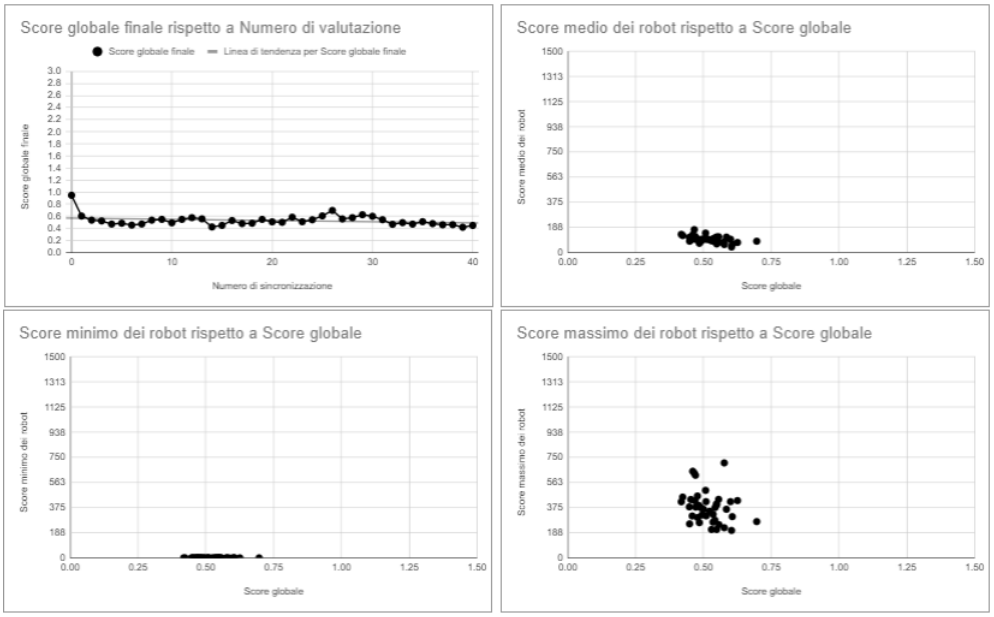

Figura 3.14: Andamento dei risultati della prima esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo senza sincronizzazione

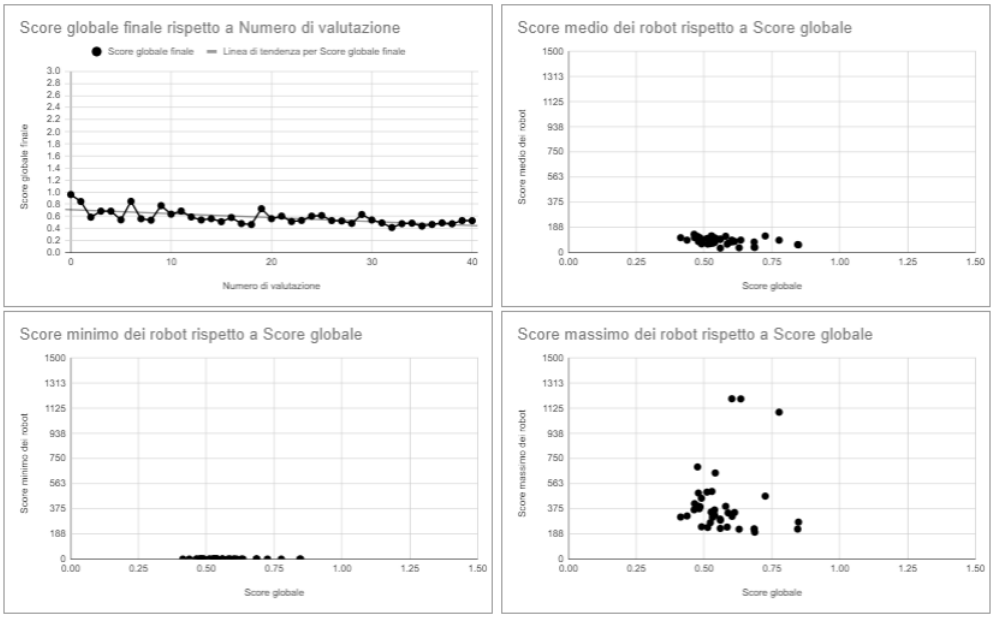

Figura 3.15: Andamento dei risultati della seconda esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo senza sincronizzazione

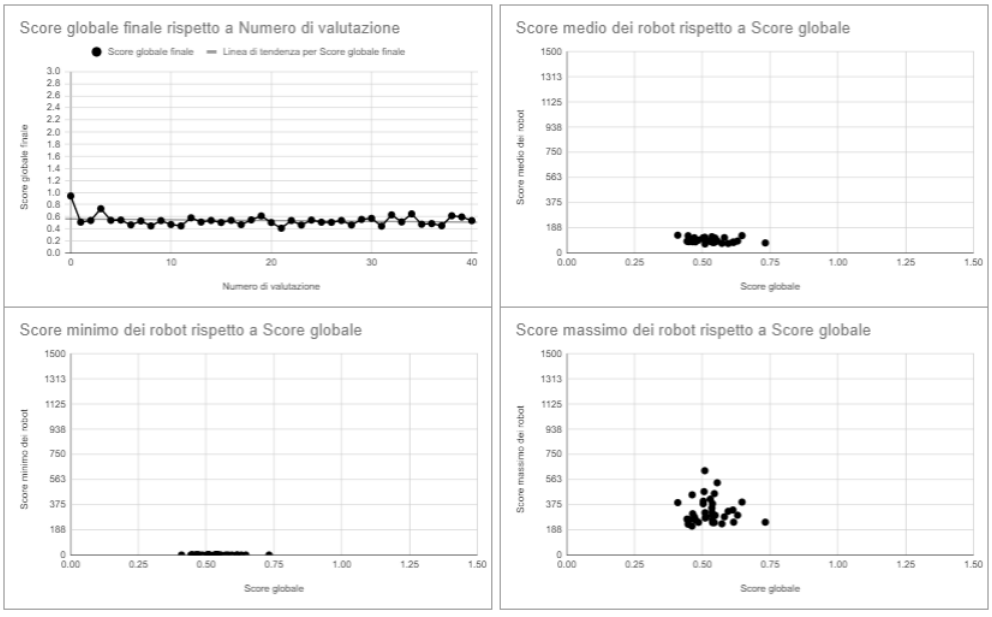

Figura 3.16: Andamento dei risultati della terza esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo senza sincronizzazione

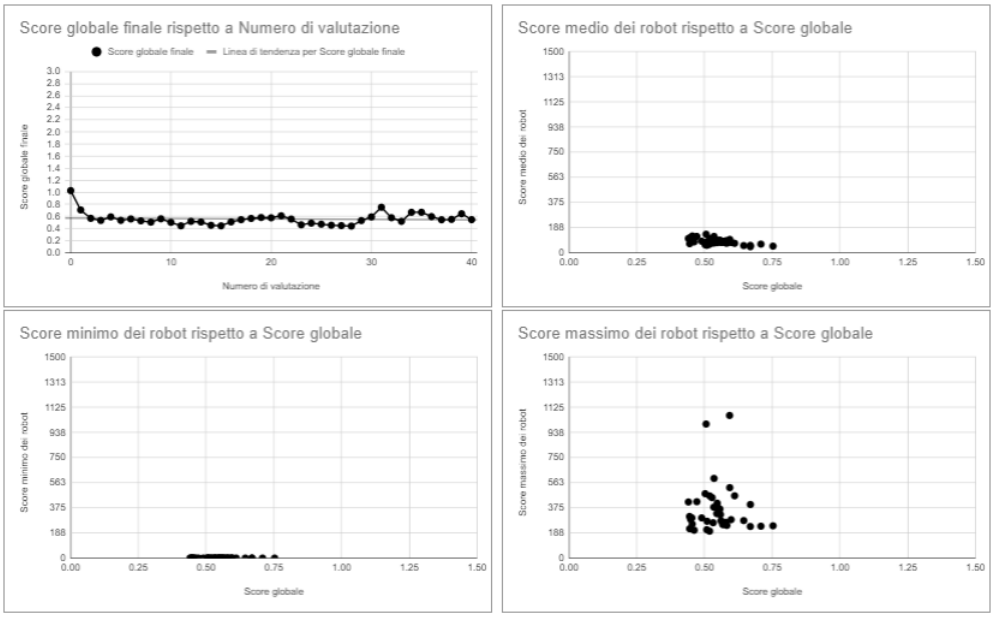

Figura 3.17: Andamento dei risultati della quarta esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo senza sincronizzazione

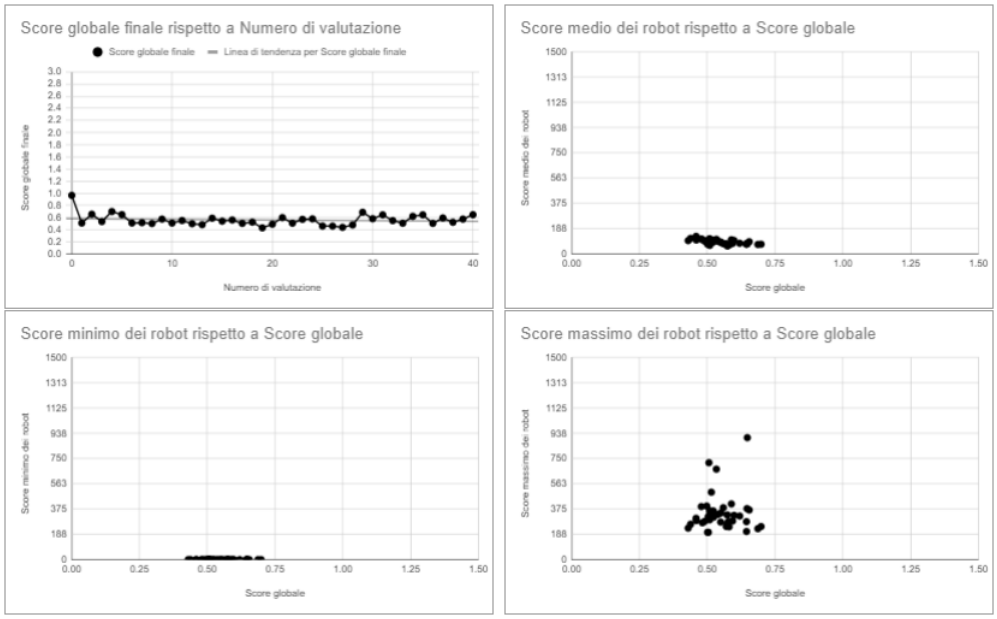

Figura 3.18: Andamento dei risultati della quinta esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo senza sincronizzazione

Dai grafici di entrambi i task, che mostrano l'andamento dello score globale

dello sciame, risulta evidente un miglioramento immediato dello score già dalla prima valutazione globale, nelle successive valutazioni invece l'andamento risulta pressoché costante, salvo leggere variazioni. Questo indica che la ricerca di una rete migliore dalle rete causali di partenza è veloce, dal momento che dopo poche valutazioni lo sciame riesce ad ottenere un comportamento tale da ottenere un punteggio globale prossimo a 0.50. In questo caso però, giunti a determinate prestazioni, non si ottiene un ulteriore miglioramento da parte dell'intero sciame. Tale comportamento lo si può individuare in tutte e cinque le simulazioni di entrambi i task.

Analizzando i restanti grafici, che mostrano gli score dei robot rispetto allo score globale, si nota che il punteggio medio rimane sempre su un range di valori bassi per qualsiasi score globale, migliorando leggermente al migliorare dello score globale.

Dai grafici che mostrano i punteggi massimi è possibile osservare anche punteggi buoni, ma nel complesso il punteggio medio è molto basso, indice del fatto che molti robot non si comportano correttamente.

Inoltre, è evidente che i punteggi minimi di ogni valutazione globale equivalgono a zero, ovvero, esiste sempre qualche robot che non esegue correttamente il task, non prendendo punti. Questo può essere dovuto ad alcuni robot che non sono ancora riusciti a trovare una rete adatta al compito assegnato.

Tutte le simulazioni per entrambi i task ottengono uno score globale intorno a 0.50 a partire dalla prima valutazione e mantengono lo stesso andamento per tutta la durata dell'esperimento. Dai grafici analizzati, si può osservare una coerenza dei valori ottenuti che rimangono piuttosto stabili in tutte le esecuzioni di entrambi i task, dimostrando la correttezza dei punteggi assegnati, nonostante non sia immediatamente evidente a causa della poca variazione dei risultati. In alcuni grafici, come quello della prima esecuzione del Task 1 in figura 3.9, è possibile vedere in maniera più evidente la corretta correlazione tra valutazione globale dello sciame e i punteggi ottenuti singolarmente dai robot.

3.2.2 Evoluzione con Sincronizzazione

Task 1

Le figure 3.19, 3.20, 3.21, 3.22 e 3.23 mostrano i risultati ottenuti dagli esperimenti relativi al Task 1 utilizzando il meccanismo evolutivo con sincronizzazione.

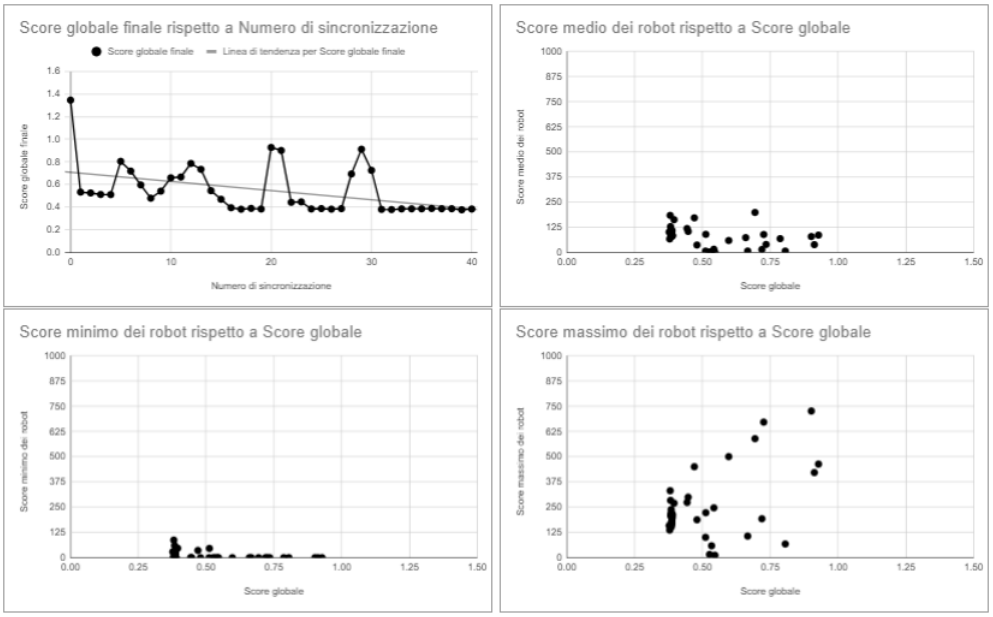

Figura 3.19: Andamento dei risultati della prima esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo con sincronizzazione

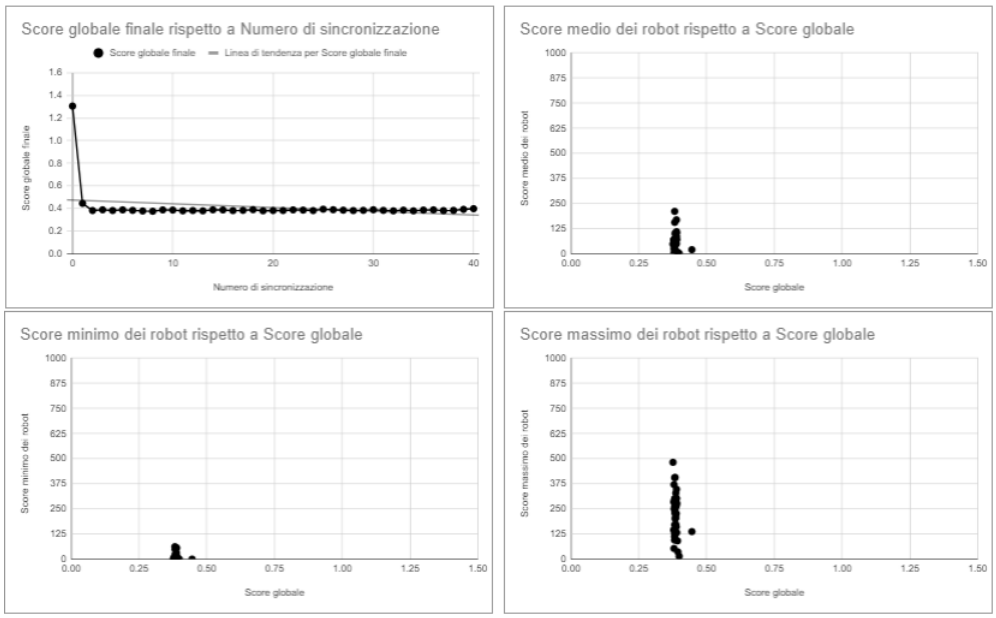

Figura 3.20: Andamento dei risultati della seconda esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo con sincronizzazione

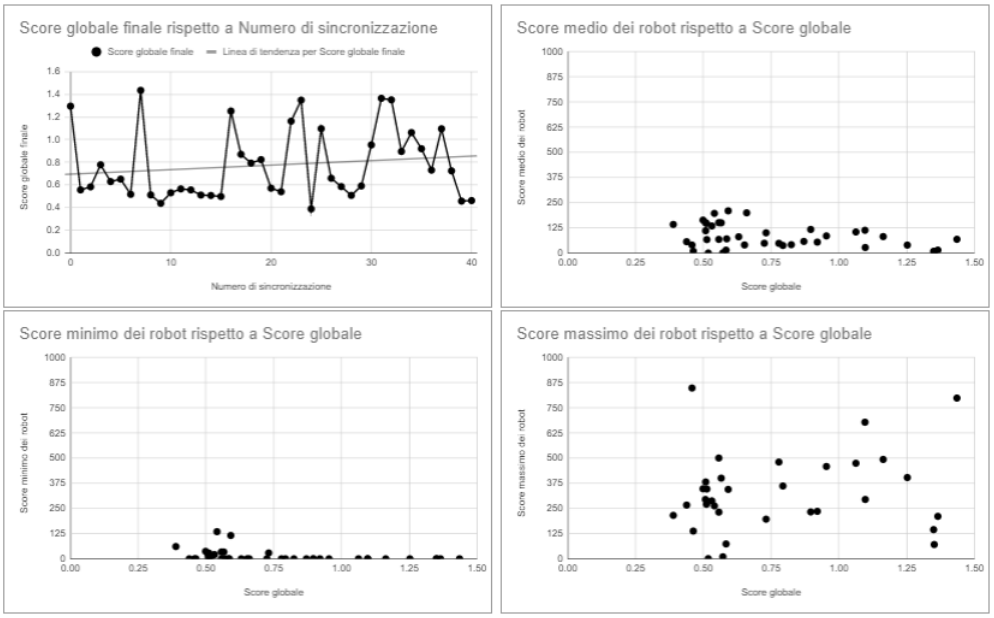

Figura 3.21: Andamento dei risultati della terza esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo con sincronizzazione

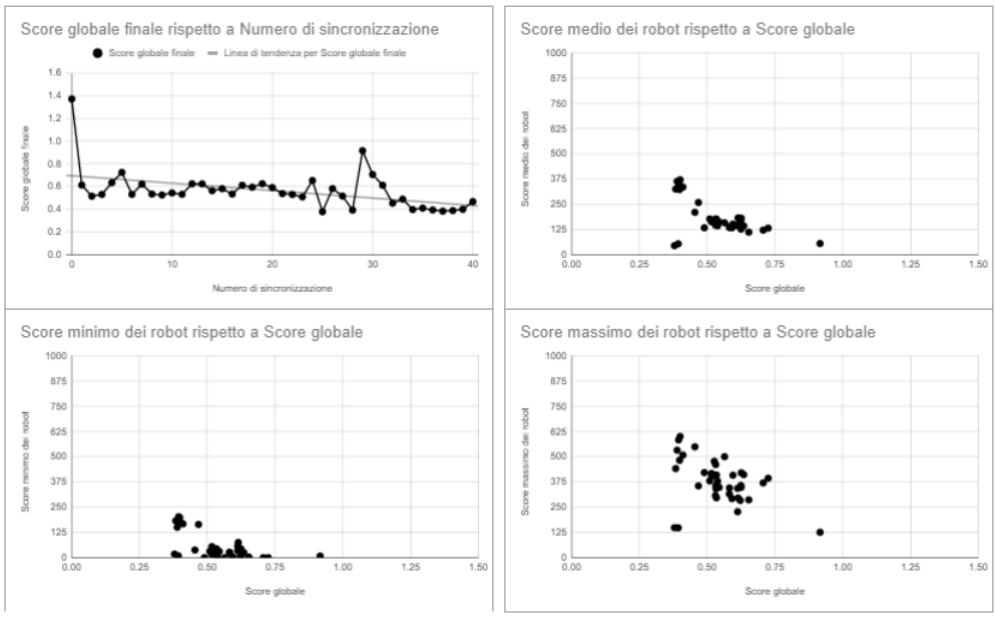

Figura 3.22: Andamento dei risultati della quarta esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo con sincronizzazione

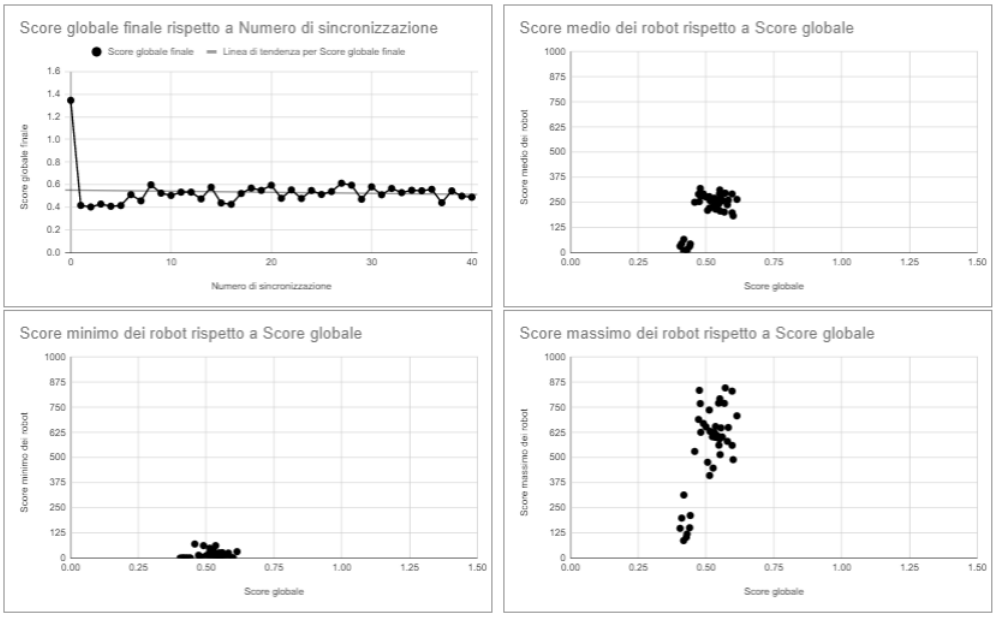

Figura 3.23: Andamento dei risultati della quinta esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo con sincronizzazione

Task 2

Le figure 3.24, 3.25, 3.26, 3.27 e 3.28 mostrano i risultati ottenuti dagli esperimenti relativi al Task 2 utilizzando il meccanismo evolutivo con sincronizzazione.

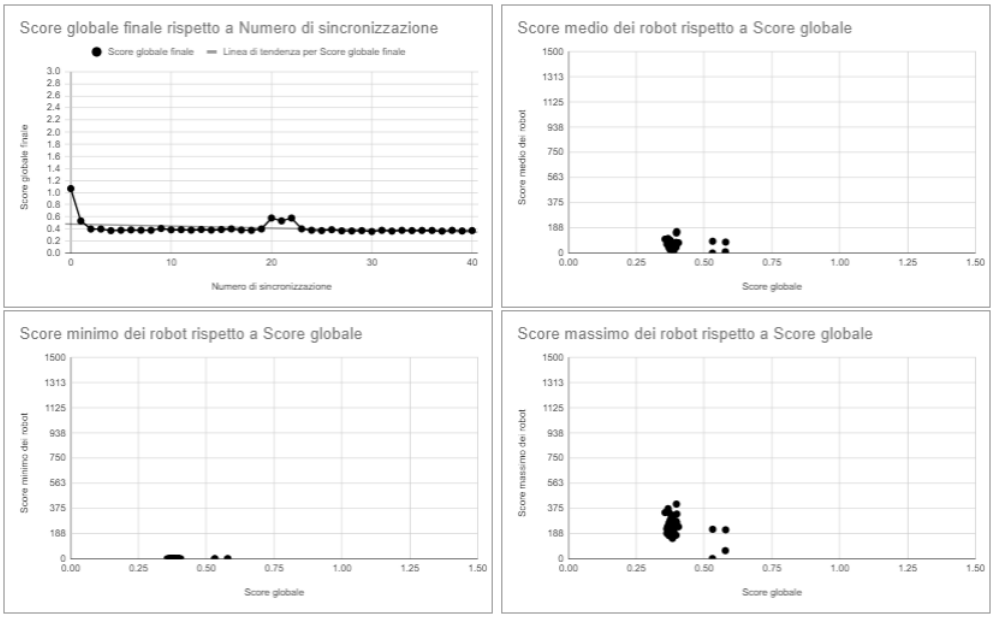

Figura 3.24: Andamento dei risultati della prima esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo con sincronizzazione

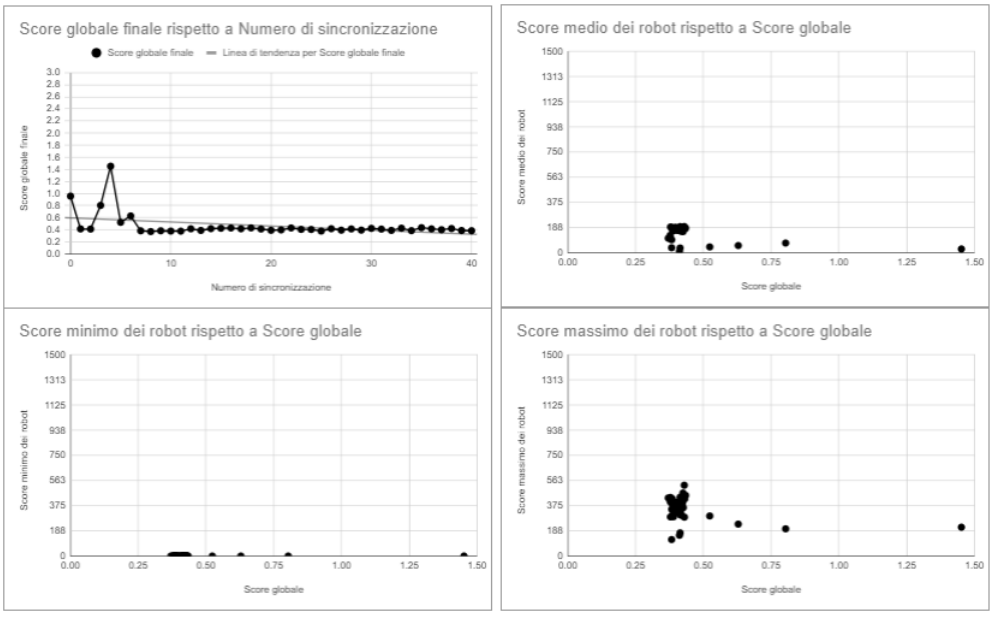

Figura 3.25: Andamento dei risultati della seconda esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo con sincronizzazione

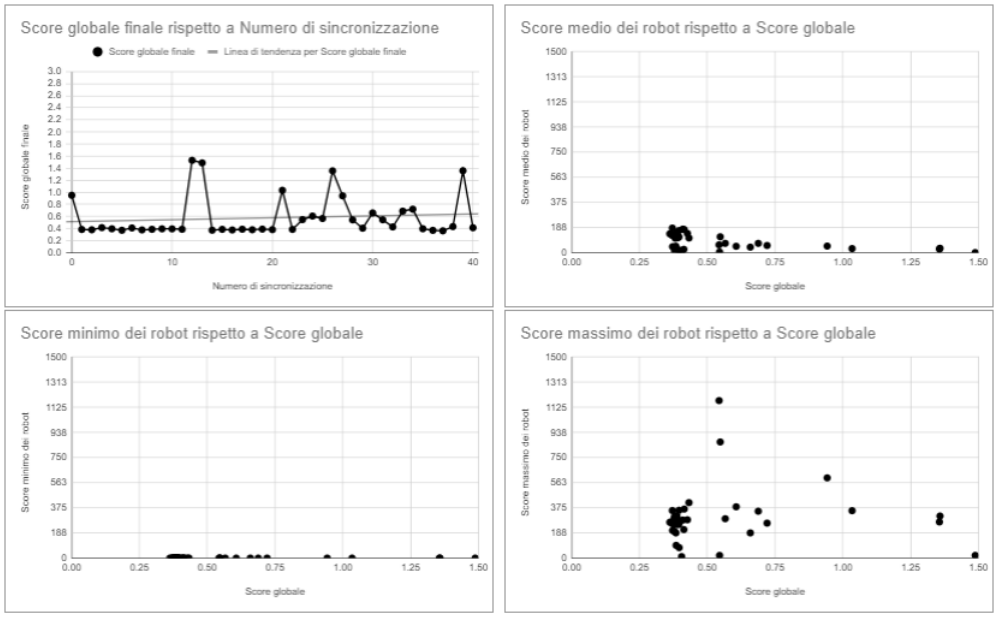

Figura 3.26: Andamento dei risultati della terza esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo con sincronizzazione

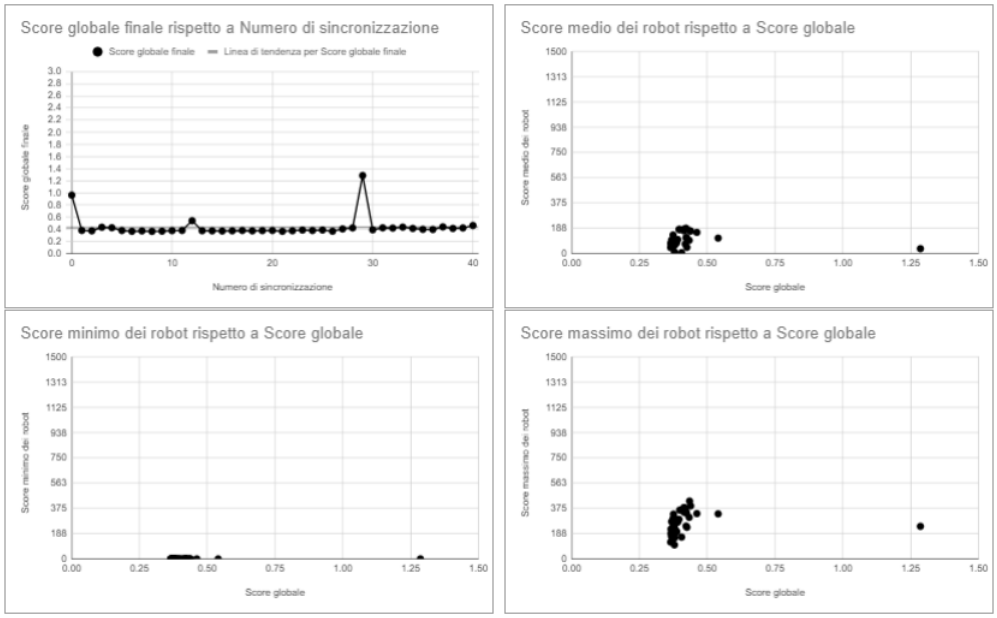

Figura 3.27: Andamento dei risultati della quarta esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo con sincronizzazione

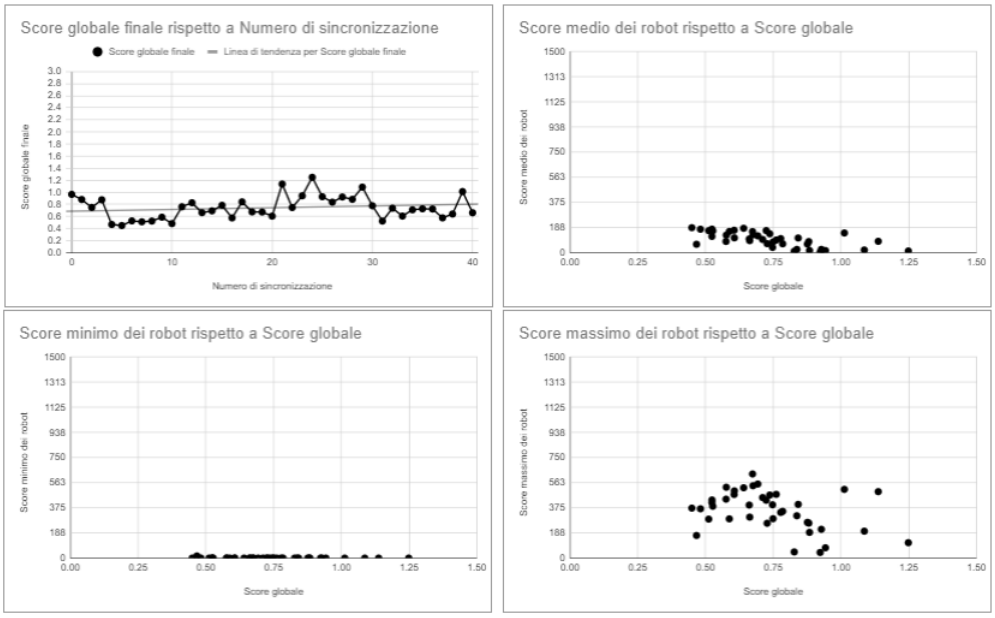

Figura 3.28: Andamento dei risultati della quinta esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo con sincronizzazione

Con il meccanismo in esame, con sincronizzazione, si nota un andamento decisamente più altalenante rispetto al meccanismo senza sincronizzazione. In alcune simulazioni abbiamo, infatti, diversi picchi dovuti al fatto che lo sciame ha ottenuto un punteggio molto alto durante una determinata valutazione globale, che equivale ad un comportamento non corretto per il task da risolvere in questione. Lo si può notare per le esecuzioni di entrambi i task.

Questi picchi possono essere dovuti a robot che ottengono un punteggio singolo molto alto in situazioni fortunate, mantenendo una rete non ottima in memoria e passando questa rete tramite la sincronizzazione ai restanti robot dello sciame. In questo caso il comportamento dello sciame dovuto alla rete passata tramite la sincronizzazione non è idoneo con il task.

Dai grafici che indicano l'andamento dell'evoluzione è possibile osservare a colpo d'occhio che dalle prime sincronizzazioni lo sciame ottiene punteggi immediatamente migliori, dopo di che le valutazioni sono in alcuni casi altalenanti, con picchi frequenti, ma si può osservare anche che c'è una tendenza al miglioramento per alcuni esperimenti.

I risultati di questi esperimenti sono migliori rispetto agli esperimenti effettuati con il meccanismo evolutivo senza sincronizzazione, infatti gli score globali risultano inferiori in quattro simulazioni sulle cinque effettuate.

Dagli altri grafici indicanti la media, si osserva invece che per alcune si-

mulazioni i punteggi assegnati dalla funzione obiettivo ai singoli robot sono coerenti con le valutazioni globali dell'intero sciame, ovvero, a punteggi alti dei robot corrispondono punteggi bassi della valutazione globale, mentre per altre simulazioni si presentano delle incongruenze. In particolare, lo si riesce a distinguere in alcuni esperimenti del Task 1, (figure 3.19 e 3.23) dove si può osservare dai grafici che non sempre gli score medi e massimi risultano migliori con score globale più basso. Questo può essere dovuto a caratteristiche specifiche di questo tipo di meccanismo, più che dalla funzione obiettivo che valuta i robot.

Interessante è il fatto che durante gli esperimenti effettuati per il Task 1 si osservano punteggi minimi spesso maggiori di 0, cosa che non si è verificata con il meccanismo precedente. Questo indica che con la sincronizzazione è possibile ottenere un comportamento dello sciame in cui tutti i robot ottengono punti in qualche modo.

Questo però non si verifica per gli esperimenti riguardanti il Task 2, in quanto i punteggi minimi rimangono costanti a 0. Questo mette in luce il fatto che aumentando la complessità del task, aumenta anche la difficoltà nel trovare delle reti che completino correttamente il task proposto, in quanto i robot che ottengono 0 punti potrebbero non aver ricevuto una rete ottimale per la propria condizione, in particolar modo, inserendo le zone proibite, un robot potrebbe trovarsi all'interno di questa zona e non riuscire più ad uscire, ottenendo 0 punti.

3.2.3 Evoluzione con Sincronizzazione Reversibile

Task 1

Le figure 3.29, 3.30, 3.31, 3.32 e 3.33 mostrano i risultati ottenuti dagli esperimenti relativi al Task 1 utilizzando il meccanismo evolutivo con sincronizzazione reversibile.

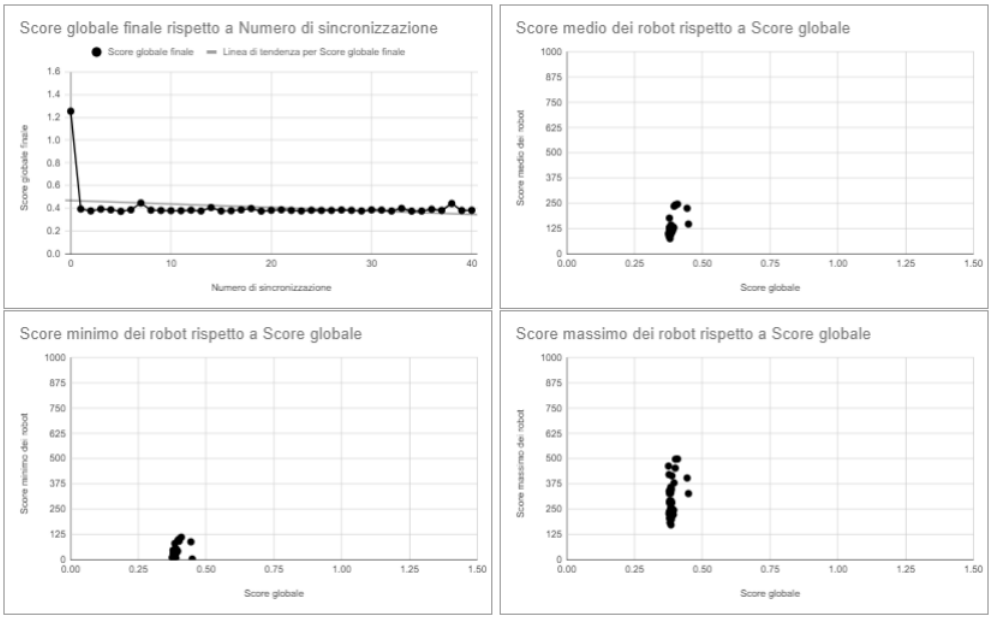

Figura 3.29: Andamento dei risultati della prima esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo con sincronizzazione reversibile

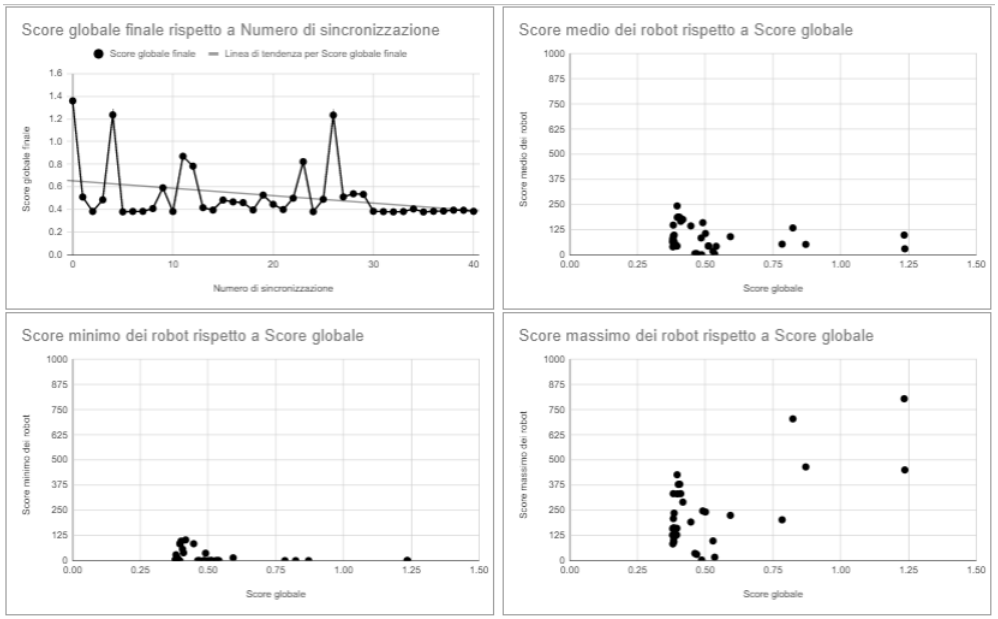

Figura 3.30: Andamento dei risultati della seconda esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo con sincronizzazione reversibile

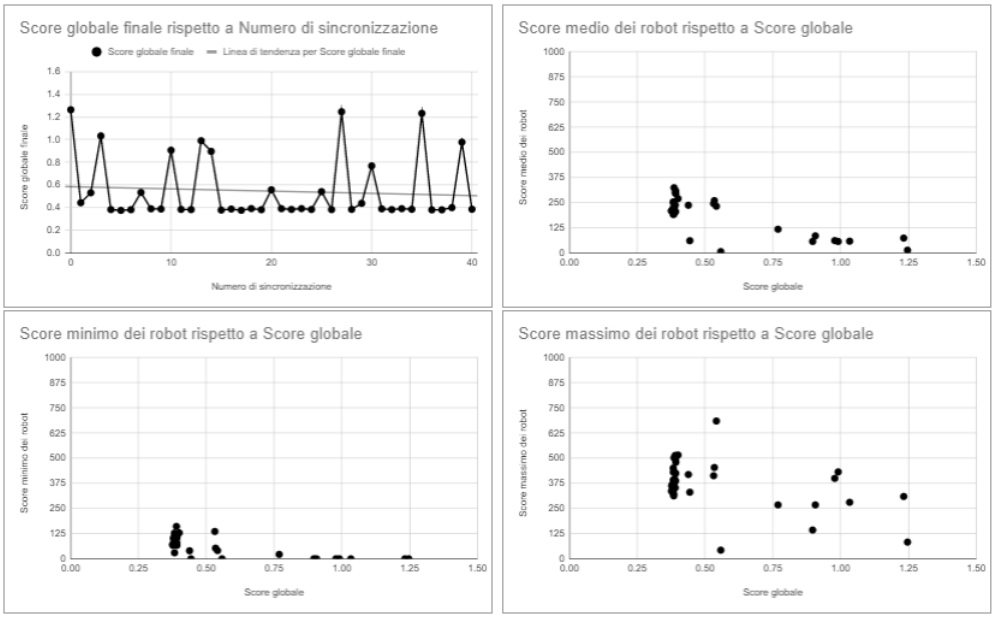

Figura 3.31: Andamento dei risultati della terza esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo con sincronizzazione reversibile

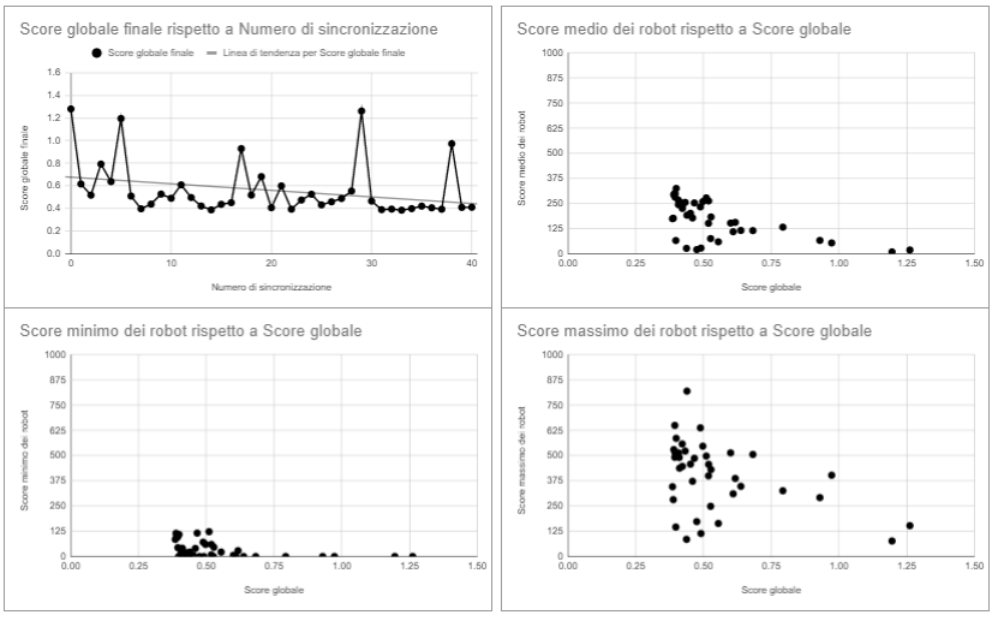

Figura 3.32: Andamento dei risultati della quarta esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo con sincronizzazione reversibile

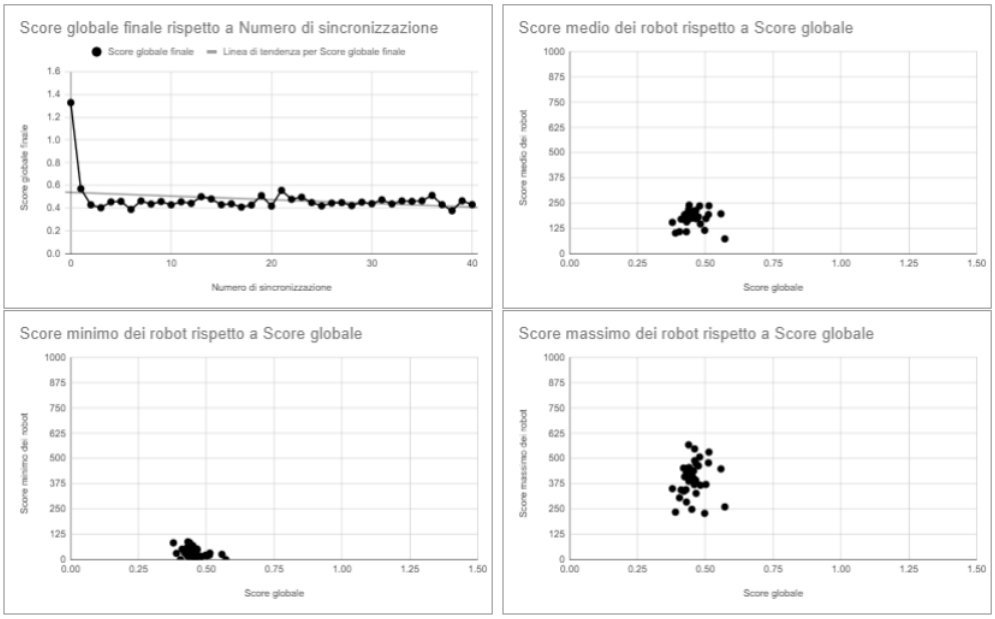

Figura 3.33: Andamento dei risultati della quinta esecuzione del Task 1 con l'utilizzo del meccanismo evolutivo con sincronizzazione reversibile

Task 2

Le figure 3.34, 3.35, 3.36, 3.37 e 3.38 mostrano i risultati ottenuti dagli esperimenti relativi al Task 2 utilizzando il meccanismo evolutivo con sincronizzazione reversibile.

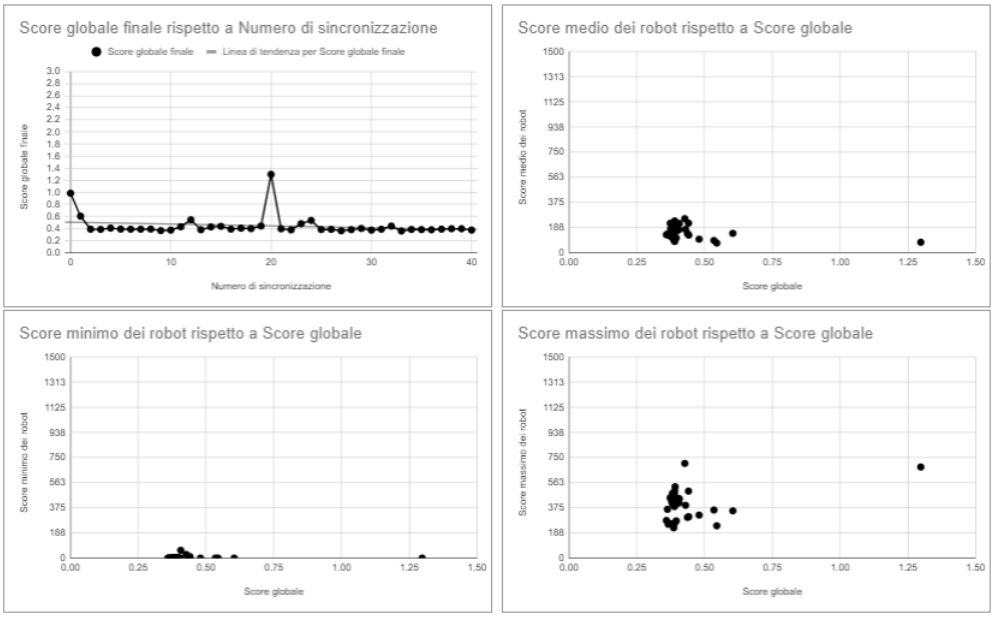

Figura 3.34: Andamento dei risultati della prima esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo con sincronizzazione reversibile

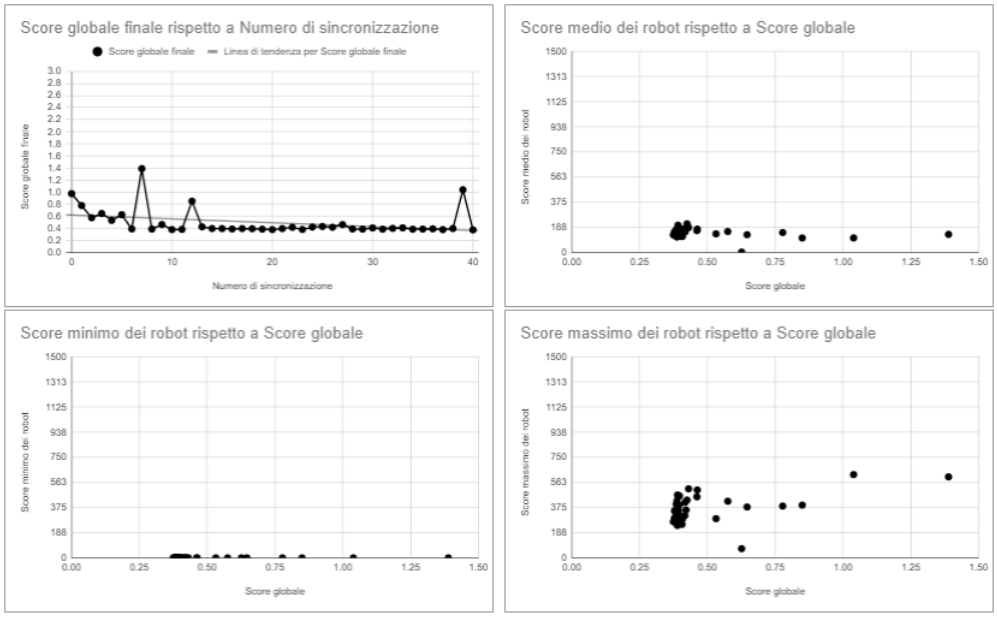

Figura 3.35: Andamento dei risultati della seconda esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo con sincronizzazione reversibile

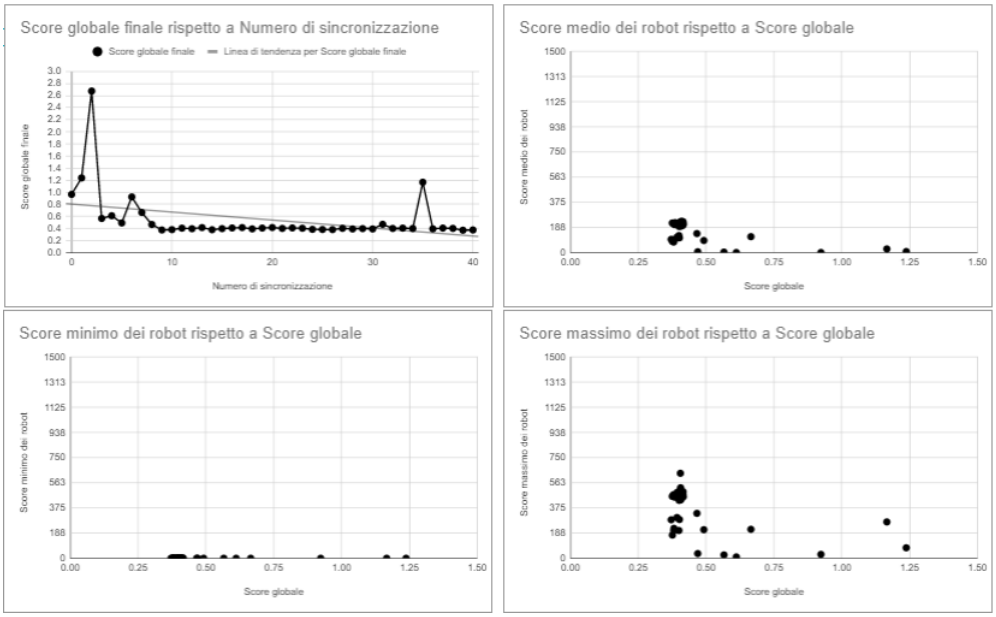

Figura 3.36: Andamento dei risultati della terza esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo con sincronizzazione reversibile

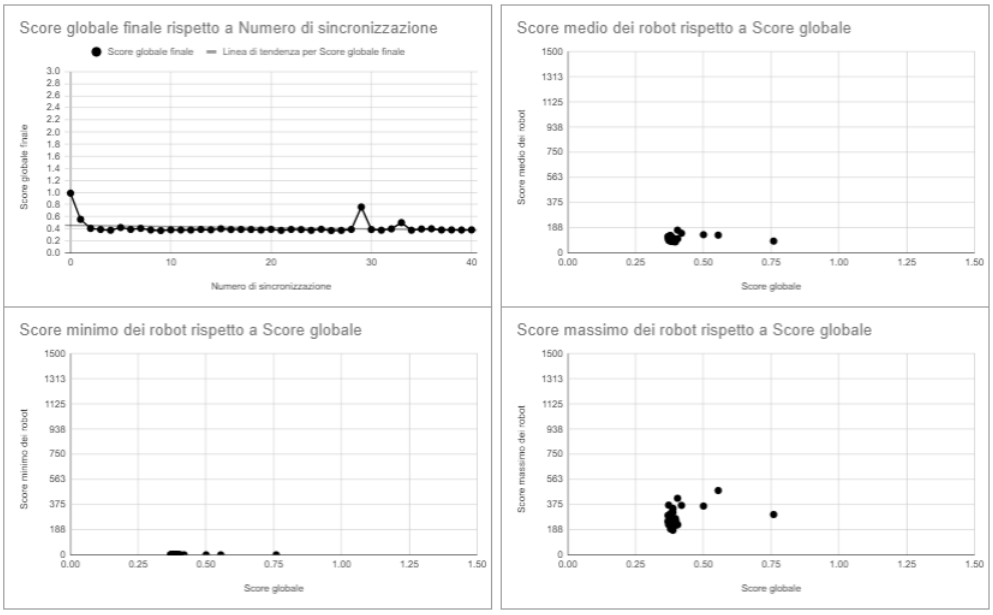

Figura 3.37: Andamento dei risultati della quarta esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo con sincronizzazione reversibile

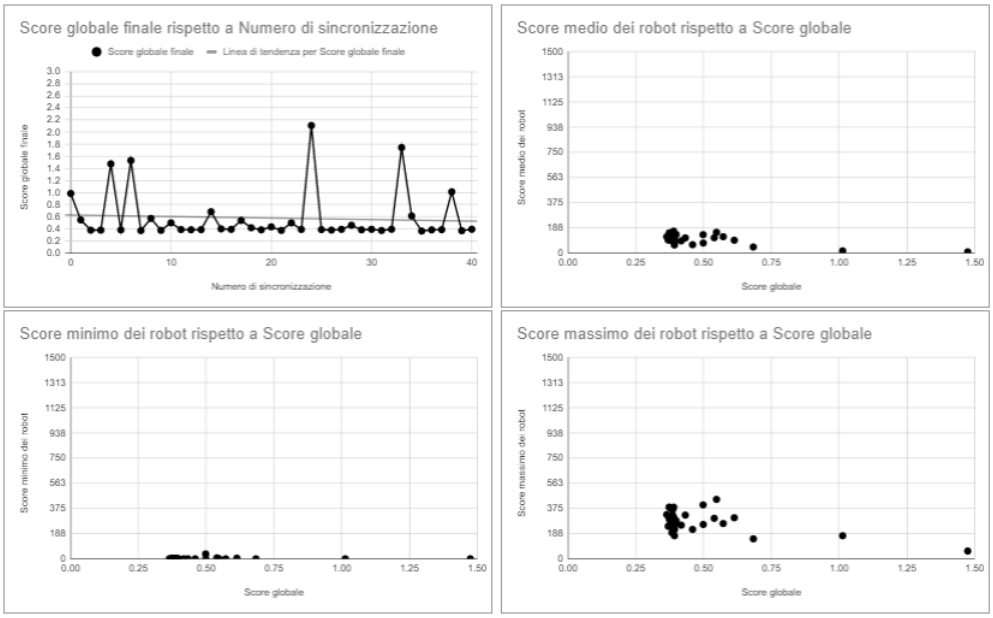

Figura 3.38: Andamento dei risultati della quinta esecuzione del Task 2 con l'utilizzo del meccanismo evolutivo con sincronizzazione reversibile

Analizzando i grafici degli esperimenti svolti con questo meccanismo di sincronizzazione reversibile, i valori risultano assegnati ai robot in maniera coerente, dal momento che a punteggi bassi nella valutazione globale corrispondono punteggi alti nella valutazione dei singoli robot e viceversa, ad esclusione di alcuni score dovuti a reti che hanno ottenuto un buon punteggio anche se non adeguate al task.

Infatti, si osserva su entrambi i task la presenza di picchi all'interno dei grafici in cui è indicato l'andamento globale dei risultati durante ogni sincronizzazione. Questi picchi sono probabilmente dovuti a reti di robot che hanno ottenuto un buon punteggio probabilmente in un caso fortunato, non idonee al compito da svolgere e trasmesse con la sincronizzazione.

Come per il meccanismo con sincronizzazione, analizzato nella sezione 3.2.2, anche in questo caso si osserva la medesima differenza tra i due task sugli score minimi dei singoli robot, che risultano in quasi tutti i casi a zero per il Task 2 e superiori allo zero per il Task 1, in linea con i risultati medi ottenuti dallo sciame.

Mettendo a confronto i risultati ottenuti dall'intero sciame con l'utilizzo del meccanismo con sincronizzazione reversibile rispetto ai meccanismi precedenti è possibile osservare un miglioramento dei risultati ottenuti durante le sin-

cronizzazioni/valutazioni globali. Infatti, quando, tramite la sincronizzazione, viene trasmessa una buona rete all'intero sciame si nota uno score (da minimizzare) inferiore ad entrambi gli altri approcci. In particolare, i risultati sono sempre migliori degli score ottenuti con l'approccio senza sincronizzazione e in alcuni casi sono inferiori anche ai punteggi ottenuti con la sincronizzazione non reversibile.

Riepilogo dei risultati ottenuti

Mostriamo ora un riepilogo dei risultati ottenuti a seguito degli esperimenti svolti in simulazione relativi ai due task eseguiti con i tre diversi meccanismi evolutivi.

I dati ottenuti dalle simulazioni verranno analizzati al fine di affermare se (A) i robot dello sciame siano in grado di imparare a svolgere un determinato compito a loro assegnato in maniera autonoma (online), (B) identificare con quale tipo di meccanismo si riescono ad ottenere i risultati migliori e (C) valutare se la sincronizzazione possa essere d'aiuto per l'apprendimento.

Le statistiche sui dati vengono riassunti dalle tabelle e dai grafici mostrati di seguito.

Nella tabella 3.1 vengono mostrati i risultati ottenuti dagli esperimenti effettuati per risolvere il task con arena libera, mentre nella tabella 3.2 vengono mostrati i risultati ottenuti dagli esperimenti svolti per risolvere il task con arena con zone proibite.

In queste tabelle vengono mostrati, per ogni meccanismo evolutivo:

- gli score finali ottenuti dall'intero sciame di robot alla fine di ogni esperimento, in ordine dalla migliore alla peggiore esecuzione;
- i relativi score iniziali, ovvero il punteggio effettuato dallo sciame prima dell'inizio dell'addestramento e che dipende dalla posizione iniziale dei robot;
- la media dei risultati ottenuti;
- la deviazione standard dei risultati ottenuti;

Meccanismo evolutivo	Score finale 1	Score finale 2	Score finale 3	Score finale 4	Score finale 5	Score iniziale 1	Score iniziale 2	Score iniziale 3	Score iniziale 4	Score iniziale 5	Media	Deviazione standard
Evoluzione senza sincronizzazione	0.483338	0.499929	0.503664	0.504489	0.539165	1.456218	1.465846	1.45088	1.47258	1.449486	0.506117	0.02035934725
Evoluzione con sincronizzazione	0.383954	0.398789	0.462694	0.467846	0.489993	1.34588	1.305468	1.295648	1.3709	1.345899	0.4406552	0.04644101528
Evoluzione con sincronizzazione reversibile	0.384177	0.384495	0.385932	0.411538	0.432655	1.254303	1.359669	1.263667	1.279929	1.328127	0.3997594	0.02172479002

Tabella 3.1: Riassunto dei dati di tutte le esecuzioni effettuate con il Task 1

Meccanismo evolutivo	Score finale 1	Score finale 2	Score finale 3	Score finale 4	Score finale 5	Score iniziale 1	Score iniziale 2	Score iniziale 3	Score iniziale 4	Score iniziale 5	Media	Deviazione standard
Evoluzione senza sincronizzazione	0.447906	0.5262	0.534801	0.547263	0.646523	0.947232	0.959445	0.942678	1.02856	0.96515	0.5405386	0.07088234033
Evoluzione con sincronizzazione	0.371153	0.384174	0.413577	0.461523	0.661328	1.06524	0.956328	0.95096	0.96234	0.965874	0.458351	0.1186593377
Evoluzione con sincronizzazione reversibile	0.374817	0.374883	0.375631	0.38006	0.39279	0.984521	0.974847	0.964203	0.987603	0.984526	0.3796362	0.007665854271

Tabella 3.2: Riassunto dei dati di tutte le esecuzioni effettuate con il Task 2

Nelle figure 3.39 e 3.40 vengono mostrati i *box-plot* dei risultati ottenuti durante il Task 1 ed il Task 2.

I box-plot sono rappresentati da un rettangolo e rappresentano un modo conciso per esprimere le caratteristiche salienti della distribuzione. Questi diagrammi sono ottenuti a partire dai 5 numeri di sintesi: il minimo, il 1° quartile (Q1), la mediana, il 3° quartile (Q3) e il massimo. La scatola del box plot ha come estremi inferiore e superiore rispettivamente Q1 e Q3. La mediana divide la scatola in due parti. I baffi si ottengono congiungendo Q1 al minimo e Q3 al massimo. Il minimo è definito come il dato più basso entro 1,5 IQR del quartile inferiore (Q1) e il massimo come il dato più alto entro 1,5 IQR del quartile superiore (Q3), dove IQR è l'intervallo interquartile, dato dall'intervallo [Q1, Q3]. Nel caso in cui siano presenti dei valori maggiori o minori della lunghezza dei baffi, saranno identificati come outliers (osservazioni eccezionali).

Da questi diagrammi, confrontando tra loro le lunghezze dei due baffi e le altezze dei due rettangoli che costituiscono la scatola (che rappresentano le distanze tra Q1 e mediana e tra mediana e Q3) si ottengono informazioni sulla simmetria della distribuzione.

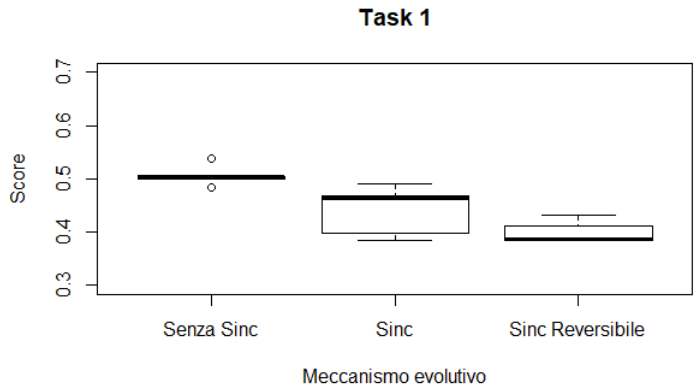

Figura 3.39: Box-plot dei risultati ottenuti utilizzando i tre diversi meccanismi evolutivi durante il Task 1 (score minore è migliore).

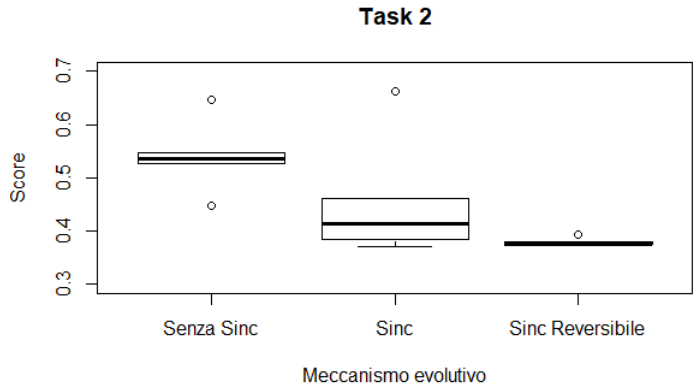

Figura 3.40: Box-plot dei risultati ottenuti utilizzando i tre diversi meccanismi evolutivi durante il Task 2 (score minore è migliore).

3.3 Discussione dei Risultati

In questa sezione verranno analizzati i risultati ottenuti durante la fase di addestramento, verranno poi testate tre delle cinque reti risultanti dagli esperimenti relativi ad ogni meccanismo evolutivo in esame, infine verranno effettuate le considerazioni finali.

Analisi dei risultati

Con i dati ottenuti dai risultati dell'addestramento siamo in grado di effettuare un miglior confronto dei meccanismi evolutivi in esame.

Le tabelle 3.1 e 3.2 mostrano il riepilogo dei dati degli esperimenti per entrambi i task. Da questi risultati si nota fin da subito un miglioramento delle prestazioni dell'intero sciame di robot con l'utilizzo dei meccanismi di sincronizzazione per tutti e due i compiti assegnati.

Questo crescente miglioramento risulta subito evidente anche dai box-plot, mostrati all'interno delle figure 3.39 e 3.40, tramite i quali è possibile analizzare meglio i risultati ottenuti dagli esperimenti fatti per il Task 1 ed il Task 2.

Da questi diagrammi si nota che la differenza tra le rispettive mediane e l'intero box-plot è significativa, in particolare tra il meccanismo senza sincronizzazione e il meccanismo con sincronizzazione reversibile. Quest'ultimo risulta il migliore tra tutti i meccanismi analizzati.

Entrando nel dettaglio dei diagrammi relativi al **Task 1** con arena libera, si può osservare che lo sciame addestrato tramite il *meccanismo con sincronizzazione reversibile* ottiene nella maggior parte dei casi valori migliori di quelli ottenuti dallo sciame addestrato con il meccanismo di sincronizzazione. Inoltre, ottiene sempre risultati nettamente migliori rispetto all'addestramento effettuato senza sincronizzazione. E' anche possibile notare che i risultati sono mediamente gli stessi in tutte le simulazioni, mantenendosi in un range piuttosto ristretto, come indicato infatti dal valore di deviazione standard pari a 0.02172479002, mostrato all'interno della tabella dei risultati. Dai risultati è anche possibile osservare che tutti i risultati e la media indicano valori prossimi e anche migliori del valore σ_1 definito come ottimale.

Per quanto riguarda il *meccanismo con sincronizzazione* semplice, i risultati ottenuti sono maggiormente variabili rispetto ai restanti meccanismi, infatti, possiede una scatola e dei baffi più ampi. In questo contesto, in cui anche un piccolo cambiamento dello score globale pari a 0,05 può influire sul risultato finale dell'intero sciame, questa variazione dei risultati rende il meccanismo meno affidabile nella ricerca di una soluzione ideale al task. E' comunque possibile ottenere buoni punteggi paragonabili al meccanismo con sincronizzazione reversibile, come si può osservare dai valori minimi del box-plot e dalla tabella dei risultati, ma anche punteggi elevati paragonabili al meccanismo senza sincronizzazione e che non producono un comportamento ottimale rispetto al task assegnato.

Infine, analizzando il diagramma ottenuto dalle *simulazioni senza sincronizzazione*, si osservano risultati pressoché identici nella maggior parte delle simulazioni, con un intervallo di valori molto basso, in quanto presenta una

scatola molto sottile. In due casi, però, ottiene risultati leggermente migliori e leggermente peggiori, che si discostano dai restanti tre, rappresentati dai 2 outlier. Questo meccanismo però non ottiene risultati ottimali come i meccanismi con sincronizzazione, infatti, gli score risultano molto superiori al risultato ottimo indicato da σ_1.

Entrando ora nel dettaglio dei dati relativi al **Task 2**, rimane evidente il miglioramento ottenuto dai meccanismi con sincronizzazione. Il *meccanismo di sincronizzazione reversibile* ottiene in quasi tutti gli esperimenti gli stessi valori, se non per un outlier con una valutazione leggermente superiore, ma comunque buona. Questo è ben evidenziato anche dal valore di deviazione standard di 0.007665854271 all'interno della tabella 3.2. La media dei risultati risulta leggermente superiore allo score ottimo σ_2, però si può definire un ottimo punteggio, per cui i robot svolgono in maniera piuttosto corretta il compito assegnato.

Dal box-plot relativo al *meccanismo di sincronizzazione semplice*, invece, come per il diagramma relativo al primo task, i risultati ottenuti risultano in un intervallo più ampio in cui sono presenti sia valori più alti (punteggio peggiore), che valori paragonabili al meccanismo di sincronizzazione reversibile o addirittura migliori. Inoltre, è possibile osservare un outlier che ottiene un punteggio molto alto, che è il peggiore di tutti gli esperimenti effettuati. Questo può significare che al termine della simulazione lo sciame non è ancora riuscito ad individuare una buona rete per il task. Lo si può osservare anche all'interno della figura 3.26 che mostra l'andamento dei punteggi dell'esperimento.

Infine, dal diagrammi a scatola e baffi relativo al *meccanismo senza sincronizzazione* si osservano dei risultati piuttosto superiori a σ_2, per i quali il risultato dello sciame rispetto al task non si può definire buono, ad eccezione dell'outlier inferiore.

In conclusione, dopo aver analizzato dei risultati ottenuti durante l'addestramento online con i diversi meccanismi, si può dire che su entrambi i task si ottengo all'incirca gli stessi risultati, indicando, quindi. che l'approccio utilizzato è indipendente dal task scelto.

I risultati del Task 2 evidenziano, però, una maggiore difficoltà nella ricerca di una buona rete booleana con task più complessi, in quanto in tutti i diagrammi a scatola e baffi sono presenti degli outlier anche piuttosto distanti dalla mediana. Inoltre, gli score ottenuti non sono mai pari al valore ottimale σ_2, nonostante siano molto buoni, a differenza di alcuni risultati del Task 1 che si avvicinano molto a σ_1.

E' immediato in entrambi i casi osservare un miglioramento a scalare dei risultati ottenuti tramite la sincronizzazione, in particolare con la sincronizza-

zione reversibile. Questo significa che, con le configurazioni delle reti booleane ed i parametri da noi scelti, la sincronizzazione è di aiuto durante l'addestramento online di uno sciame di robot, mentre con un meccanismo senza sincronizzazione sembra difficile ottenere reti con risultati soddisfacenti.

Di seguito verranno testate le reti che hanno ottenuto lo score migliore, lo score medio e lo score peggiore, in modo da valutare la reale efficacia dell'addestramento.

Test delle reti risultanti

Per analizzare la capacità di generare reti in grado di risolvere correttamente il task richiesto durante la fase di training, sono stati condotti una serie di test sulle reti booleane risultanti.

Questi test sono eseguiti in un ambiente simulato su condizioni iniziali differenti rispetto a quelle del training, per valutarne l'effettiva efficacia. Nello specifico, per quanto riguarda il Task 1, i robot verranno inizializzati casualmente, anziché essere in un angolo, al centro dell'arena all'interno di un'area di dimensione $1.5m \times 1.5m$. Per quanto riguarda, invece, il Task 2, i robot verranno inizializzati casualmente sempre al centro dell'arena, in più verrà utilizzata una diversa configurazione delle zone proibite all'interno della pavimentazione. Le figure 3.41 e 3.42 mostrano le configurazioni iniziali utilizzate per i test.

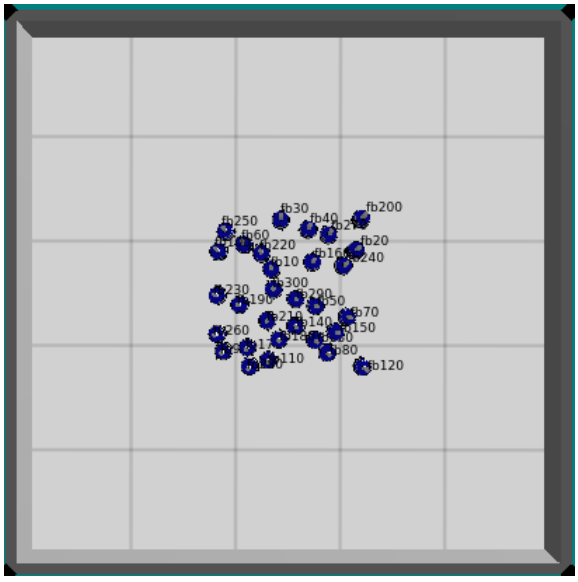

Figura 3.41: Stato iniziale dei robot durante i test per il Task 1

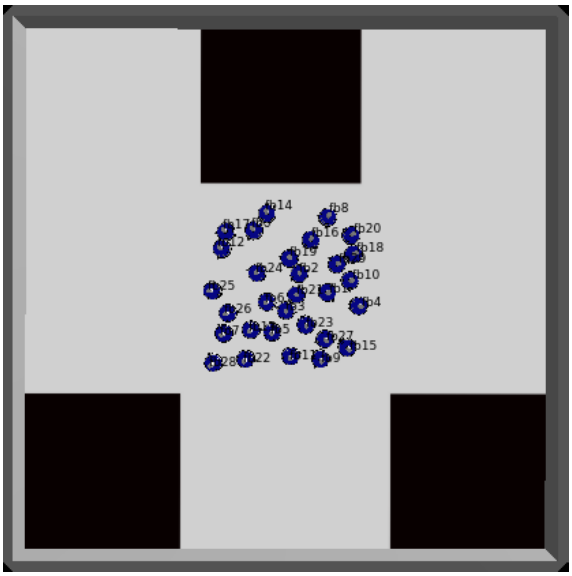

Figura 3.42: Stato iniziale dei robot durante i test per il Task 2

Per ogni meccanismo evolutivo studiato verranno testate tre delle cinque reti risultati, in particolare, ci concentreremo sulle reti appartenenti agli esperimenti numerati con 1, 3 e 5 all'interno delle tabelle riassuntive 3.1 e 3.2, ovvero quelle reti che hanno ottenuto il punteggio migliore, le reti che hanno ottenuto il punteggio globale mediano e le reti che hanno ottenuto il punteggio peggiore, in modo tale da verificarne il corretto funzionamento e analizzare la differenza tra la migliore e la peggiore delle reti ottenute.

In totale, verranno effettuate 10 esecuzioni di test per ogni esperimento. Ogni simulazione avrà una durata di 120 secondi (1200 step da 10ms), un tempo ritenuto sufficiente per consentire allo sciame di robot di soddisfare il task assegnato.

Per quanto riguarda i test relativi alle reti ottenute con il meccanismo evolutivo senza sincronizzazione, verranno testati contemporaneamente i 30 differenti controller risultanti, ognuno inserito all'interno di un robot dello sciame. Mentre, con le reti ottenute dai meccanismi con sincronizzazione, essendo la rete risultante solo una, questa verrà data come controller a tutti i robot dello sciame.

I punteggi vengono assegnati nello stesso modo in cui venivano calcolati i punteggi globali dello sciame durante l'addestramento, attraverso la funzione obiettivo globale indicata all'interno della sezione 2.2.

I risultati vengono mostrati all'interno delle figure 3.43 e 3.44 e sono riportati graficamente mediante box-plot. I box-plot sono suddivisi per meccanismo evolutivo e, in ordine da sinistra a destra, corrispondono alle distribuzioni rela-

tive ai test delle reti che hanno ottenuto punteggio migliore, mediano e peggiore durante la valutazione globale finale del training.

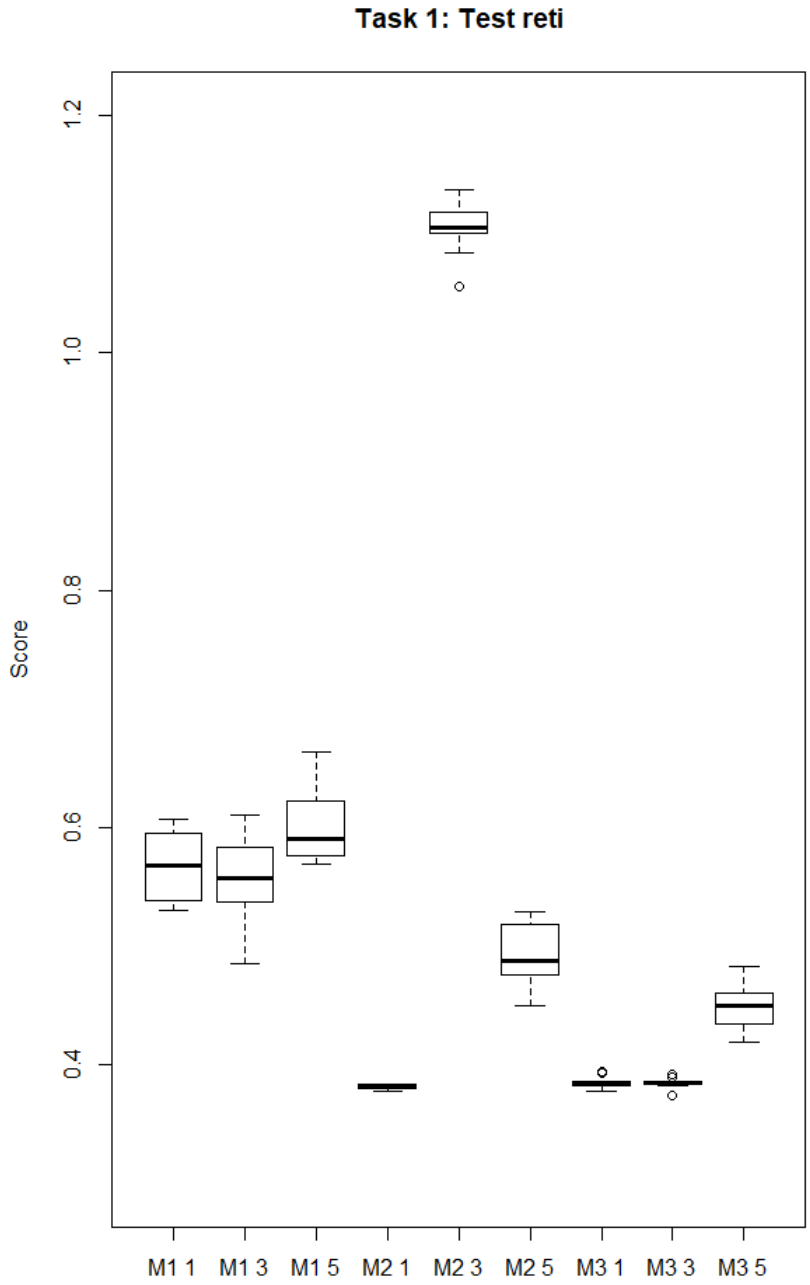

Figura 3.43: Box-plot relativi alle distribuzioni dei punteggi globali ottenuti su 10 esecuzioni delle reti risultanti dall'addestramento per il Task 1. M1, M2 e M3 indicano rispettivamente i meccanismi evolutivi senza sincronizzazione, con sincronizzazione e con sincronizzazione reversibile. Per ogni meccanismo i numeri 1, 3 e 5 indicano che il test viene eseguito rispettivamente con le reti migliori, mediane e peggiori ottenute dall'addestramento.

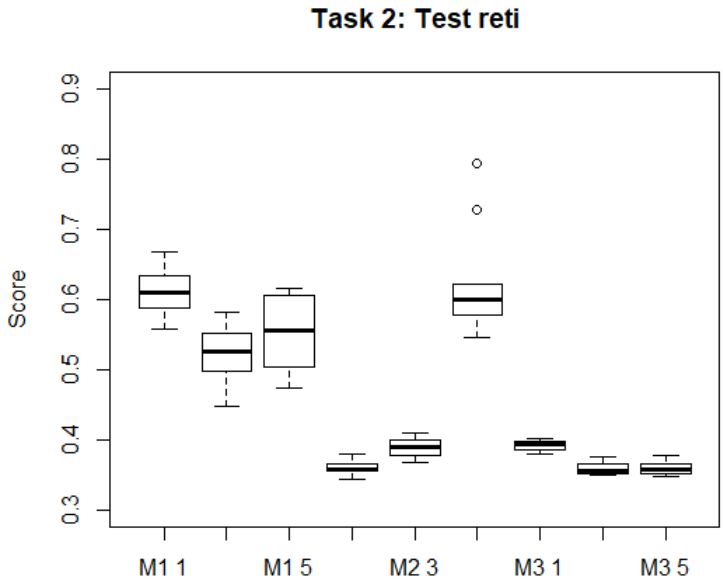

Figura 3.44: Box-plot relativi alle distribuzioni dei punteggi globali ottenuti su 10 esecuzioni delle reti risultanti dall'addestramento per il Task 2. M1, M2 e M3 indicano rispettivamente i meccanismi evolutivi senza sincronizzazione, con sincronizzazione e con sincronizzazione reversibile. Per ogni meccanismo i numeri 1, 3 e 5 indicano che il test viene eseguito rispettivamente con le reti migliori, mediane e peggiori ottenute dall'addestramento.

Di seguito riportiamo, per ogni meccanismo evolutivo, la media e la deviazione standard delle 9 reti testate. Questi risultati verranno poi paragonati con i risultati presenti all'interno della tabella 3.1 e della tabella 3.2, ovvero quelli ottenuti durante la fase di addestramento delle reti dei controller.

Meccanismo Evolutivo	Rete testata	Media	Deviazione Standard
Evoluzione senza sincronizzazione	1	0.5665575	0.02960227065
	3	0.5584095	0.03679492639
	5	0.5989594	0.03121829538
Evoluzione con sincronizzazione	1	0.3815764	0.002074780802
	3	1.105937	0.02356165673
	5	0.4914453	0.02490000487
Evoluzione con sincronizzazione reversibile	1	0.3856522	0.005113181849
	3	0.3851149	0.004721839789
	5	0.4487455	0.02126758351

Tabella 3.3: Riassunto dei dati che indica media e deviazione standard dei test effettuati per il Task 1

Meccanismo Evolutivo	Rete testata	Media	Deviazione Standard
Evoluzione senza sincronizzazione	1	0.6126116	0.0336145007
	3	0.5199693	0.042156437
	5	0.5543314	0.05275420454
Evoluzione con sincronizzazione	1	0.3590597	0.00937857632
	3	0.3886964	0.0132109876
	5	0.6236412	0.07716693495
Evoluzione con sincronizzazione reversibile	1	0.3916462	0.007913136136
	3	0.3583039	0.009530734418
	5	0.359795	0.009355742289

Tabella 3.4: Riassunto dei dati che indica media e deviazione standard dei test effettuati per il Task 2

Task 1 Dalla prima figura 3.43, che visualizza le distribuzioni dei punteggi globali ottenuti dallo sciame, possiamo affermare che 6 reti su 9 testate sono coerenti con i risultati ottenuti dai robot al termine dell'addestramento. Di queste 6 reti, solamente 3 possono essere etichettate come reti risultanti di successo, poiché i loro punteggi, inclusi i valori anomali, sono pari a σ_1, e solamente una rete può essere etichettata con risultati buoni, in quanto inferiori ad un punteggio di 0.47, con esempio all'interno della figura 3.6 che mostra un comportamento non perfetto, ma buono.

E' importante però tenere in considerazione il fatto che le reti scelte per il test sono la migliore, le mediana e la peggiore di quelle risultanti dal training. Questo permette anche di analizzare se, con un determinato meccanismo

evolutivo, è possibile ottenere reti con le stesse prestazioni o con prestazioni altalenanti.

Detto ciò, la *sincronizzazione reversibile* è il meccanismo che ottiene, rispetto agli altri meccanismi, le reti migliori in quasi tutti i casi. Si osserva che la sua rete 5, nonostante sia la rete risultante peggiore, da comunque buoni risultati che ottengono punteggi vicini allo score ottimo σ_1, mentre le reti 1 e 3 hanno un IQR quasi nullo, indicante risultati costanti per ogni istanza, ad eccezione di alcuni outlier.

Questi risultati, inoltre, sono coerenti con quelli ottenuti al termine dell'addestramento, al contrario invece di altre reti, come la rete 3 del *meccanismo di sincronizzazione semplice M2* che da una punteggio di 0.462694, ottenuto al termine dell'addestramento, passa ad un punteggio medio di 1.105937 nei test. In questo ultimo caso, nessuna istanza di test effettua uno score buono, al contrario, i punteggi si aggirano intorno ai punteggi effettuati da tutti i robot stazionari in una piccola area dell'arena. Come si vede all'interno del grafico in figura 3.21, infatti, i picchi ottenuti durante l'addestramento equivalgono al punteggio della configurazione iniziale dei robot all'interno dell'arena. Questo potrebbe essere dovuto al fatto che ai robot è stato assegnato un controller che gli indica di stare fermi o di eseguire piccoli movimenti, senza quindi espandersi, come richiesto dal task. In alcuni casi infatti, come abbiamo accennato precedentemente, può avvenire che un robot, durante una run fortunata, ottenga un punteggio molto alto, e che continui a trasmettere quella rete non ottima ai restanti robot dello sciame.

Nonostante ottengano una rete non funzionante, il punteggio di 0.462694 dello sciame è dovuto al fatto che, durante l'evoluzione online, i robot iniziano ogni Train a partire dalla posizione in cui avevano terminato il Train precedente, per cui i robot tra una sincronizzazione e l'altra continuano ad esplorare nuove reti tramite un flip di un valore della tabella della verità di un nodo, che possono permettergli di muoversi all'interno dell'arena. Per questo motivo, al termine dell'addestramento le loro posizioni gli permettevano di effettuare un punteggio globale abbastanza basso, ma le reti che li guidavano erano quelle di robot stazionari, avendo al termine con un comportamento scorretto. Questo è confermato all'interno dei grafici degli score medi e massimo in figura 3.21, in cui si può osservare che nella esecuzione in questione un robot effettua un punteggio molto alto, mentre la media degli score dello sciame è molto bassa, indice di un comportamento non corretto per il task.

E' comunque possibile ottenere reti ottime, come dimostra la rete 1, che ottiene punteggi anche migliori dell'evoluzione con sincronizzazione reversibile.

Infine, i diagrammi dei test effettuati sulle reti ottenute *senza sincronizzazione* risultano tutte con IQR abbastanza ampio e mediane molto superiori alle reti che hanno ottenuto punteggi buoni. Inoltre, le medie ottenute dalle

istanze di test sono tutte superiori ai relativi punteggi ottenuti durante l'addestramento. Questo potrebbe essere dovuto al fatto che lo sciame durante ogni istanza di test inizi l'esecuzione al centro dell'arena, a differenza di ciò che avviene durante l'addestramento, in cui, ad ogni Train, i robot vengono valutati a partire dalla posizione in cui si trovano a seguito dei movimenti effettuati in base alle diverse reti eseguite. La varianza dei risultati ottenuti e i punteggi piuttosto elevati sono indice di un comportamento dello sciame non conforme con il task.

Task 2 Per quanto riguarda il secondo Task, su 9 configurazioni di reti testate, 3 possono ritenersi ottime, con risultati nell'ordine di σ_2, mentre 2 possono ritenersi molto buone. I punteggi buoni, ma non ottimali, di queste due reti potrebbero essere dovuti al fatto che, durante l'evoluzione, alcuni robot entrino all'interno delle zone proibite, mentre i rimanenti robot dello sciame si espandono correttamente all'interno dell'arena, evitando di entrare in queste zone nella maggior parte dei casi.

Dalla figura 3.44 e dai risultati presenti della tabella 3.4, vengono confermate le analisi fatti per il Task 1. Il *meccanismo con sincronizzazione reversibile* si dimostra, anche in questo caso, quello che ottiene i risultati migliori e che rimangono buoni per tutte le reti testate, a differenza del meccanismo con sincronizzazione non reversibile in cui c'è una grande differenza tra i punteggi ottenuti dalla rete 1 alla 5 e del meccanismo senza sincronizzazione che rimane distante dai risultati desiderati. Infatti, la dimensione dell'intervallo interquartile dei test delle reti di M1 rimanere abbastanza costante e con nessun valore anomalo, mentre per M2 si osserva una rete in cui sono presenti due outlier e per M1 l'intervallo interquartile e i baffi risultano piuttosto ampi.

I punteggi medi ottenuti durante il test con il meccanismo M3 risultano, inoltre, migliori rispetto ai punteggi effettuati al termine dell'addestramento. Questo può essere dovuto al fatto che, durante l'evoluzione, alcuni i robot entrati nelle zone proibite non sono riusciti ad uscire, mentre i restanti robot hanno continuato a migliorare la rete ottenendo il risultato richiesto.

Lo stesso vale per il *meccanismo M2* che ottiene, in tutte le simulazioni di test, punteggi medi migliori rispetto ai risultati ottenuti dall'addestramento. In aggiunta, tutti i risultati delle tre reti testate sono in linea con quelli ottenuti al termine dell'evoluzione, differentemente da quando si era verificato per una rete del Task 1.

Per quanto riguarda il *meccanismo senza sincronizzazione*, i risultati delle reti non sono conformi al task, in più, la rete che risultava migliore al termine dell'addestramento, durante i test è risultata la peggiore in quasi tutte le simulazioni, come è possibile osservare dalla differenza delle mediane di M1 1, M1 3 e M1 5. Invece, per quanto riguarda le altre due reti (3 e 5) hanno ottenuto

risultati in media migliori rispetto ai punteggi della fase di addestramento. In ogni caso, queste reti ottengono risultati piuttosto variabili, nessuno dei quali, neanche quelli relativi ai baffi inferiori, può essere ritenuto soddisfacente. La causa di tutto ciò potrebbe essere il fatto che i punteggi ottenuti dalle reti dipendano fortemente dalla posizione di partenza dei robot all'inizio di ogni valutazione.

Considerazioni finali

Al termine degli esperimenti e delle analisi fino ad ora realizzate siamo in grado di effettuare un miglior confronto tra i diversi meccanismi evolutivi studiati all'interno di questa tesi e valutare la capacità di questi approcci di evoluzione online di ottenere dei comportamenti per lo sciame di robot in linea con i task assegnati.

Prima di tutto è possibile osservare che ognuno dei tre meccanismi ottiene performance simili per entrambi i task, infatti, lo sciame di robot, con il modello delle reti scelto e l'aiuto della sincronizzazione, riesce ad ottenere per tutti e due i task reti ottime, anche nel caso di Task 2 in cui è stata aggiunta complessità.

Dall'analisi dei risultati ottenuti, tramite il **meccanismo evolutivo senza sincronizzazione** risulta un compito difficile ottenere buone reti per i task proposti con il modello delle reti da noi utilizzato e una semplice discesa stocastica. I risultati delle simulazioni effettuate con questo meccanismo incapsulato mostrano, infatti, un miglioramento immediato, rispetto al punteggio della configurazione iniziale dello sciame, dopodiché i risultati si stabilizzano su valori piuttosto alti rispetto ai valori globali ottimali σ_1 e σ_2. Questo accade perché solamente pochi robot riescono ad ottenere buoni punteggi dall'evoluzione delle loro reti, mentre la maggior parte dei robot non riesce a trovare una rete adeguata al task, non permettendo alla sciame di convergere ad una soluzione. Infatti, in un processo evolutivo online, siccome i software di controllo di ogni robot evolvono indipendentemente cercando di imparare a svolgere i compiti a loro assegnati, se un robot continua a valutare dei controller poco prestanti, le performance di quel robot non saranno adeguate, indipendentemente da quanto siano buone le prestazioni del resto dello sciame, e questo pregiudica il punteggio finale.

La causa di questa condizione potrebbe essere dovuta al fatto che, con l'approccio da noi utilizzato, i robot non siano in grado di modificare la configurazione della propria rete e potrebbe succedere che, con una determinata topologia di rete, non risulti possibile ottenere il comportamento richiesto. Ad esempio, potrebbe accadere che con la topologia della RBN creata alcuni nodi

di input non vengono connessi in ingresso ad altri nodi, oppure vengano collegati solo a nodi terminali, nodi senza connessioni uscenti o ad una sequenza ciclica, sprecando quindi il contributo di alcune percezioni. Ricordiamo inoltre che, con l'algoritmo da noi utilizzato, l'unica modifica apportata ad una rete di un robot tra un trial e l'altro è un valore di una entry di una tabella delle verità di un nodo scelto casualmente. Con questo approccio, il processo di ricerca, a seconda della RBN di partenza, potrebbe incastrarsi in minimi locali, senza possibilità di sfuggire ad essi, o in aree del panorama di ricerca che non presentano soluzioni di "buona" qualità.

Tramite l'utilizzo di un **approccio con sincronizzazione**, dalle analisi effettuate risulta invece possibile ottenere reti buone, che permettono ai robot di comportarsi come richiesto. Questo avviene per entrambi i task da noi studiati. Con la sincronizzazione globale, si superano quindi alcune limitazioni sopracitate del meccanismo senza sincronizzazione, infatti, se il controller di un robot è guidato da una rete non adatta alla risoluzione del task, quest'ultima verrà sostituita con la migliore trovata dall'intero sciame. Ogni robot inizialmente ha 40 trial per evolvere la propria rete, dopo di che, durante la sincronizzazione, questa rete viene sostituita con la migliore trovata dallo sciame e che ogni robot inizierà ad evolvere tramite ricerca locale fino alla successiva sincronizzazione. I robot, con questo meccanismo, sono quindi vincolati ad evolvere, a seguito della prima sincronizzazione, una rete che non è detto abbia una configurazione ideale per il task scelto e che permetta di ottenere un buon risultato. Probabilmente questo è il motivo per cui i software di controllo ottenuti con questo algoritmo danno risultati variabili, in quanto, è molto dipendente dalla prima rete sincronizzata. Se i robot riescono a trovare una buona configurazione di rete inizialmente, durante tutta la simulazione non faranno altro che migliorarla, mentre se la rete ricevuta non ha la giusta configurazione, perché ad esempio potrebbe essere stata una rete fortunata, oppure i 40 flip potrebbero non essere stati sufficienti per trovare la rete migliore tra tutte, potrebbe risultare difficile ottenere un buon risultato.

Per quanto riguarda il **meccanismo di sincronizzazione reversibile**, gli esperimenti dimostrano che la *sincronizzazione reversibile* fornisce un metodo efficace per l'adattamento on-line di uno sciame di robot nelle condizioni da noi studiate. Risulta infatti il migliore in termini di maggiori risultati ottenuti e stabilità delle prestazioni.

A differenza della sincronizzazione non reversibile, in questo caso i robot hanno la possibilità di non memorizzare una rete passata durante la sincronizzazione se questa non ottiene dei buoni punteggi. In particolar modo, questa tecnica riduce la possibilità di trasmissione di reti fortunate e poco adatte a robot in condizioni diverse dal robot da cui si ottiene la rete. Un valore aggiunto di questo approccio, inoltre, è dato dal fatto che ogni robot ha la possibilità di

valutare ed eventualmente evolvere reti differenti e poi mantenere quella per lui migliore. Infatti, un robot valuta sia la propria rete, generata randomicamente all'inizio dell'esperimento, sia le reti trasmesse tramite sincronizzazione. Questo permette ai robot di avere una visione più ampia delle reti, anziché essere vincolati ad una unica configurazione della rete che li controlla per tutto il processo evolutivo.

Alla fine, gli esperimenti da noi effettuati dimostrano che gli algoritmi implementati con un meccanismo evolutivo ibrido con sincronizzazione globale aiutano il processo evolutivo e sono in grado di trovare reti booleane funzionanti per il compito assegnato. Le conclusioni che abbiamo tratto da questi esperimenti,inoltre, ci danno la possibilità di migliorare gli algoritmi evolutivi presentati o di idearne di nuovi.

Conclusioni

In questa tesi abbiamo introdotto tre meccanismi evolutivi di progettazione automatica online per la generazione di reti booleane in grado di permettere ai robot di uno sciame di imparare a svolgere determinati task. I meccanismi studiati sono tutti basati su una semplice discesa stocastica, eseguendo ad ogni passo un singolo flip nella tabella di verità di un nodo scelto casualmente, e si differenziano per l'approccio evolutivo utilizzato. Il "*Meccanismo Evolutivo Senza Sincronizzazione*", utilizza un approccio evolutivo incapsulato, dove ogni robot evolve unicamente la propria rete, senza comunicare con i restanti robot dello sciame. Invece, il "*Meccanismo Evolutivo con Sincronizzazione*" ed il "*Meccanismo Evolutivo con Sincronizzazione Reversibile*" si basano su un approccio ibrido, in cui i robot ottimizzano i controller in parallelo e scambiano soluzioni candidate per il task. In entrambi i meccanismi con sincronizzazione si esegue una sincronizzazione globale, durante la quale la rete che ha ottenuto il punteggio migliore tra tutte le reti dei robot viene trasmessa a tutto lo sciame. Nel primo caso la sincronizzazione è irreversibile e da quel momento ogni robot inizierà ad evolvere la rete trasmessa, mentre nel secondo caso la sincronizzazione è reversibile, infatti, se con questa rete un robot non ottiene un risultato migliore di quello ottenuto con la propria rete migliore, allora la rete trasmessa non viene mantenuta.

L'algoritmo di ricerca cerca reti che massimizzino i risultati ottenuti dai robot attraverso una funzione obiettivo (vedi Sezione 2.2) che valuta i robot assegnandogli un punteggio positivo nel caso eseguano uno spostamento lungo una determinata direzione ottenuta dalle percezione dei sensori del robot.

Con questo caso di studio si è analizzata la capacità di queste tecniche di evolvere buoni controller comandati da BN e l'efficacia o meno della sincronizzazione per l'evoluzione delle reti booleane che controllano lo sciame di robot.

Per valutare l'efficacia dei meccanismi, questi sono stati applicati su due task, il primo, chiamato "*copertura con arena libera*", consiste nel distribuire i robot uniformemente all'interno di un'arena, mentre il secondo, chiamato "*copertura con zone proibite*", consiste nel distribuire i robot in maniera uniforme all'interno di un'arena dove sono presenti alcune zone proibite segnalate da

aree con una pavimentazione nera.

A seguito dell'evoluzione sono stati analizzati i risultati ottenuti e gli andamenti relativi all'evoluzione dello sciame durante la simulazione, effettuando un confronto prestazionale tra gli algoritmi sopra citati, al fine di trovare le possibilità da essi offerte ed evidenziarne i limiti. Successivamente, le reti booleane ottenute sono state testate per valutarne le effettive capacità. Sono state testate tre delle cinque reti ottenute dalle simulazioni per ogni meccanismo evolutivo e per ognuno dei due task, ovvero, quelle che hanno ottenuto il punteggio migliore, il punteggio mediano e il punteggio peggiore al termine dell'evoluzione.

I risultati indicano che le reti ricavate dalle evoluzioni effettuate tramite simulazioni sono nettamente migliori rispetto a quelle ottenute senza sincronizzazione. La sincronizzazione aiuta infatti a convergere ad una soluzione ottima, dal momento che tutti i robot dello sciame si dedicheranno all'esplorazione di una stessa sottopopolazione di soluzioni a partire dalla soluzione con maggiore fitness trovata fino al momento della sincronizzazione. In particolare, i risultati migliori sono ottenuti dalla sincronizzazione reversibile che ottiene risultati ottimi in 4 reti su 6 testate e buoni in 2 reti, mostrandosi il meccanismo che ottiene risultati più stabili e coerenti con i punteggi effettuati al termine delle simulazioni. Invece, il meccanismo con sincronizzazione non reversibile ottiene risultati ottimi su 2 reti testate su 6 e risultati buoni su 1 rete. Le reti ottenute risultano però meno stabili e in un caso non coerenti con il punteggio globale dello sciame ottenuto al termine dell'evoluzione. Questi risultati possono essere dovuti a reti fortunate trasmesse a tutto lo sciame e non idonee al compito da eseguire. L'algoritmo che non si basa sulla sincronizzazione, invece, non riesce ad ottenere reti con punteggi paragonabili a reti ottime in nessuna delle 10 simulazioni e, conseguentemente, nei test delle reti ottenute. Questo mette in luce un limite di questo meccanismo dato dal fatto che le reti booleane durante l'evoluzione possono arrivare ad una configurazione che è un minimo locale dello spazio delle possibili soluzioni, ma il meccanismo evolutivo utilizzato non permette di passare da un minimo locale ad un altro, non permettendo al robot di esplorare altre possibilità.

In conclusione, è interessante osservare come una semplice discesa stocastica, abbinata ad un meccanismo di sincronizzazione globale, possa ottenere reti booleane in grado di risolvere due task di copertura di un'arena con differenti complessità anche in maniera ottimale.

Sviluppi Futuri

Le possibilità di lavoro futuro aperte da questa tesi consistono nel migliorare degli algoritmi evolutivi presentati e nello svilupparne di nuovi a partire dalle analisi effettuate.

Un primo miglioramento semplice degli algoritmi potrebbe essere quello di aumentare il numero di flip casuali delle entry delle tabelle delle verità dei nodi delle reti, in modo da dare la possibilità ai controller di svincolarsi dal minimo locale in cui possono trovarsi.

Un ulteriore miglioramento potrebbe essere rappresentato da una estensione dello spazio di progettazione della rete anche alla topologia. Invece di modificare unicamente la funzione booleana che caratterizza ogni nodo, si può pensare di modificare casualmente, durante l'evoluzione, anche le connessioni entranti ed i nodi di input/output, in modo tale, in particolare nel meccanismo senza sincronizzazione, da non vincolare i robot a mantenere una sola configurazione di rete (che potrebbe essere non idonea) durante l'evoluzione.

Sarebbe, inoltre, interessante implementare nuovi algoritmi evolutivi a partire dalle considerazioni da noi effettuate durante l'analisi, tenendo in considerazione anche tecniche già proposte nello stato dell'arte e adattandole al nostro caso di studio. Ad esempio, l'algoritmo senza sincronizzazione potrebbe ottenere miglioramenti mantenendo una popolazione di individui all'interno di ogni robot, anziché mantenere una sola configurazione di rete.

Un ulteriore caso di interesse potrebbe essere quello di utilizzare una sincronizzazione locale, in un cui i robot scambiano il proprio materiale genetico solamente con i propri vicini. Questo potrebbe risultare particolarmente di aiuto in situazioni in cui i robot si trovano in condizioni differenti all'interno dell'arena, come nel caso del Task 2 dove alcuni robot potrebbero trovarsi all'interno delle zone proibite e trovarsi nella situazione di dover ricercare una rete differente dai restanti robot per uscirne.

Infine, per ulteriori esperimenti si potrebbero testare l'efficacia di questi meccanismi in compiti robotici più impegnativi, ad esempio in ambienti mutevoli.

Appendice

Appendice A

Risultati completi

In questa appendice sono raccolti i risultati completi ottenuti dagli esperimenti fatti durante la tesi.

I valori visualizzati si riferiscono ai risultati finali ottenuti dalle simulazioni durante l'addestramento delle reti con tutti i meccanismi evolutivi e tutte le funzioni obiettivo studiate per entrambi i task in esame.

Per una corretta visualizzazione e per facilitarne la lettura, le tabelle sono state inserite con un orientamento verticale.

Tabella A.1: Tabella completa dei risultati relativi al Task 1

Meccanismo evolutivo	Funzione obiettivo	Score finale 1	Score finale 2	Score finale 3	Score finale 4	Score finale 5	Score iniziale 1	Score iniziale 2	Score iniziale 3	Score iniziale 4	Score iniziale 5	Media	Deviazione standard
Evoluzione senza sincronizzazione	1	0.463867	0.476096	0.513181	0.554728	0.625787	1.4547	1.36649	1.278241	1.25137	1.270565	0.5267318	0.06575581982
	2.1	0.483946	0.48561	0.513298	0.515195	0.516862	1.285823	1.278698	1.280173	1.265621	1.272133	0.5029822	0.01667623537
	2.2	0.566136	0.620361	0.621121	0.672364	0.674978	1.346055	1.341801	1.335682	1.322676	1.334879	0.6309992	0.04489756708
	2.3	0.436784	0.44486	0.456849	0.470405	0.478675	1.269319	1.266093	1.274909	1.277023	1.301864	0.4575146	0.01735125818
	3	0.483338	0.499929	0.503664	0.504489	0.539165	1.45621	1.465846	1.45088	1.47258	1.449486	0.506117	0.02035934725
	4	0.474358	0.537432	0.538792	0.577425	0.612866	1.340322	1.330613	1.332927	1.326614	1.32834	0.5481746	0.05170321607
Evoluzione con sincronizzazione	1	0.373098	0.44546	0.542249	0.603591	0.737906	1.2874	1.280314	1.28164	1.291729	1.288703	0.5404608	0.1414290331
	2.1	0.505279	0.562447	0.563108	0.631477	1.327584	1.325337	1.324862	1.328722	1.315856	1.298884	0.717979	0.3436994951
	2.2	0.772743	0.890871	1.021634	1.083682	1.164908	1.24348	1.307352	1.253662	1.236551	1.28378	0.9867676	0.1560006059
	2.3	0.479967	0.485489	0.486518	0.567068	0.588177	1.358996	1.339403	1.283015	1.272113	1.258998	0.5214438	0.05188393705
	3	0.383954	0.398789	0.462694	0.467846	0.489993	1.34588	1.305468	1.295648	1.3709	1.345899	0.4406552	0.04644101528
	4	0.40487	0.473993	0.548586	1.074589	1.234565	1.343422	1.298548	1.315888	1.335351	1.341773	0.7473206	0.3794692157
Evoluzione con sincronizzazione reversibile	1	0.395078	0.405693	0.405693	0.443405	0.622749	1.271571	1.276206	1.286127	1.258374	1.279905	0.4545236	0.09582019988
	2.1	0.484472	0.708687	1.118562	1.168771	1.197889	1.324122	1.33592	1.342118	1.308768	1.274135	0.9356762	0.3207979431
	2.2	0.444936	0.451402	0.466541	0.4984	0.506139	1.305487	1.246978	1.239786	1.283026	1.295714	0.4734836	0.02755886589
	2.3	0.46281	0.470829	0.479531	0.482363	0.52543	1.293523	1.368493	1.373734	1.331084	1.347266	0.4841926	0.02429713202
	3	0.384177	0.384495	0.385932	0.411538	0.432655	1.254303	1.359669	1.263667	1.279929	1.328127	0.3997594	0.02172479002
	4	0.399091	0.451558	0.657812	0.732198	1.069382	1.311319	1.327203	1.341795	1.281552	1.296526	0.6620082	0.2666047495

Tabella A.2: Tabella completa dei risultati relativi al Task 2

Meccanismo evolutivo	Funzione obiettivo	Score finale 1	Score finale 2	Score finale 3	Score finale 4	Score finale 5	Score iniziale 1	Score iniziale 2	Score iniziale 3	Score iniziale 4	Score iniziale 5	Media	Deviazione standard
Evoluzione senza sincronizzazione	1	0.508746	0.595434	0.602712	0.679553	0.714152	0.93092	0.818349	0.787622	1.007705	1.001847	0.6201194	0.08014191077
	2.1	0.428599	0.453099	0.515068	0.557547	0.622995	0.990401	0.983389	1.01006	0.987592	0.979543	0.5154616	0.07869080805
	2.2	0.54804	0.587873	0.609558	0.659039	0.84476	1.0211	1.011593	0.98562	1.036072	1.025465	0.649854	0.1160864071
	2.3	0.514126	0.51937	0.547332	0.56061	0.598686	0.997564	1.024061	1.01478	0.992692	1.013383	0.5480248	0.03426240878
	3	0.447906	0.5262	0.534801	0.547263	0.646523	0.947232	0.959445	0.942678	1.02856	0.96515	0.5405386	0.07088234033
	4	0.435834	0.546206	0.555408	0.577669	0.588715	0.98521	0.965784	1.02354	0.973306	0.97457	0.5407664	0.06106481065
Evoluzione con sincronizzazione	1	0.417892	0.463796	0.541524	0.878778	1.04982	0.9623548	0.954543	0.95379	0.9754	0.981168	0.670362	0.2785836707
	2.1	0.480312	0.562698	0.605529	0.732335	0.790934	0.977739	0.985624	0.956815	0.975842	0.984087	0.6343616	0.1263051594
	2.2	0.437914	0.456085	0.63054	0.762813	1.580486	0.999058	1.003507	0.995686	1.012354	0.975412	0.7735676	0.4704116284
	2.3	0.456003	0.474221	0.489973	0.493393	0.507867	0.953213	0.978562	0.940562	0.94429	0.935555	0.4842914	0.01982785381
	3	0.371153	0.384174	0.413577	0.461523	0.661328	1.06524	0.956328	0.95096	0.96234	0.965874	0.458351	0.1186593377
	4	0.421509	0.424028	0.461888	0.524603	2.758249	0.997475	0.985214	0.998652	0.984225	1.00365	0.9180554	1.029542055
Evoluzione con sincronizzazione reversibile	1	0.434387	0.662695	0.780709	0.828457	0.993222	1.005488	1.045616	1.030653	0.999901	1.026687	0.739894	0.2079534548
	2.1	0.473147	0.499995	0.544038	0.612678	1.898455	0.993925	1.012435	1.017207	1.000545	1.026715	0.8056626	0.6131649498
	2.2	0.480801	0.4925	0.509149	0.545566	0.551597	1.018387	1.036313	1.027392	1.027407	1.023711	0.5159226	0.03154132453
	2.3	0.433944	0.459854	0.463217	0.537642	0.56532	1.003111	1.006196	0.991472	1.009084	1.002571	0.4919954	0.0563279238
	3	0.374817	0.374883	0.375631	0.38006	0.39279	0.984521	0.974847	0.964203	0.987603	0.984526	0.3796362	0.00766585427 1
	4	0.365266	0.795751	0.871209	1.357562	1.379125	0.980357	0.997117	0.994029	0.990851	1.007903	0.9537826	0.4248817881

Bibliografia

[Bianco and Nolfi, 2004] Bianco, R. and Nolfi, S. (2004). Toward open-ended evolutionary robotics: evolving elementary robotic units able to self-assemble and self-reproduce. *Connection Science*, 16(4):227–248.

[Blum and Roli, 2003] Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308.

[Bonani et al., 2010] Bonani, M., Longchamp, V., Magnenat, S., Rétornaz, P., Burnier, D., Roulet, G., Vaussard, F., Bleuler, H., and Mondada, F. (2010). The marxbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4187–4193. IEEE.

[Bredeche et al., 2009] Bredeche, N., Haasdijk, E., and Eiben, A. (2009). On-line, on-board evolution of robot controllers. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 110–121. Springer.

[Bredeche et al., 2012] Bredeche, N., Montanier, J.-M., Liu, W., and Winfield, A. F. (2012). Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. *Mathematical and Computer Modelling of Dynamical Systems*, 18(1):101–129.

[Dorigo and Roosevelt, 2004] Dorigo, M. and Roosevelt, A. F. (2004). Swarm robotics. In *Special Issue", Autonomous Robots*. Citeseer.

[Eiben et al., 2010a] Eiben, A., Haasdijk, E., and Bredeche, N. (2010a). Embodied, on-line, on-board evolution for autonomous robotics.

[Eiben et al., 2010b] Eiben, A. E., Karafotias, G., and Haasdijk, E. (2010b). Self-adaptive mutation in on-line, on-board evolutionary robotics. In *2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop*, pages 147–152. IEEE.

[Floreano and Keller, 2010] Floreano, D. and Keller, L. (2010). Evolution of adaptive behaviour in robots by means of darwinian selection. *PLoS Biol*, 8(1):e1000292.

[Floreano and Mondada, 1994] Floreano, D. and Mondada, F. (1994). Automatic creation of an autonomous agent: Genetic evolution of a neural network driven robot. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 421–430. The MIT Press.

[Francesca and Birattari, 2016] Francesca, G. and Birattari, M. (2016). Automatic design of robot swarms: achievements and challenges. *Frontiers in Robotics and AI*, 3:29.

[Francesca et al., 2015] Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., et al. (2015). Automode-chocolate: automatic design of control software for robot swarms. *Swarm Intelligence*, 9(2-3):125–152.

[Gershenson, 2004] Gershenson, C. (2004). Introduction to random boolean networks. *arXiv preprint nlin/0408006*.

[Haasdijk et al., 2010] Haasdijk, E., Eiben, A., and Karafotias, G. (2010). On-line evolution of robot controllers by an encapsulated evolution strategy. In *IEEE Congress on Evolutionary Computation*, pages 1–7. IEEE.

[Holland, 1962] Holland, J. H. (1962). Outline for a logical theory of adaptive systems. *Journal of the ACM (JACM)*, 9(3):297–314.

[Huijsman et al., 2011] Huijsman, R.-J., Haasdijk, E., and Eiben, A. (2011). An on-line on-board distributed algorithm for evolutionary robotics. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 73–84. Springer.

[Karafotias et al., 2011] Karafotias, G., Haasdijk, E., and Eiben, A. E. (2011). An algorithm for distributed on-line, on-board evolutionary robotics. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 171–178.

[Kauffman, 1969] Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of theoretical biology*, 22(3):437–467.

[Kauffman and Smith, 1986] Kauffman, S. A. and Smith, R. G. (1986). Adaptive automata based on darwinian selection. *Physica D: Nonlinear Phenomena*, 22(1-3):68–82.

[Khatib, 1986] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer.

[Koos et al., 2012] Koos, S., Mouret, J.-B., and Doncieux, S. (2012). The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145.

[Laredo et al., 2010] Laredo, J. L. J., Eiben, A. E., van Steen, M., and Merelo, J. J. (2010). Evag: a scalable peer-to-peer evolutionary algorithm. *Genetic Programming and Evolvable Machines*, 11(2):227–246.

[Lemke et al., 2001] Lemke, N., Jose'e, C., and Bodmann, B. E. (2001). A numerical investigation of adaptation in populations of random boolean networks. *Physica A: Statistical Mechanics and its Applications*, 301(1-4):589–600.

[Li et al., 2008] Li, M., Lu, K., Zhu, H., Chen, M., Mao, S., and Prabhakaran, B. (2008). Robot swarm communication networks: architectures, protocols, and applications. In *2008 Third International Conference on Communications and Networking in China*, pages 162–166. IEEE.

[Nelson et al., 2009] Nelson, A. L., Barlow, G. J., and Doitsidis, L. (2009). Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370.

[Nolfi, 1998] Nolfi, S. (1998). Evolutionary robotics: Exploiting the full power of self-organization.

[Nolfi et al., 2000] Nolfi, S., Floreano, D., and Floreano, D. D. (2000). *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press.

[Penders et al., 2011] Penders, J., Alboul, L., Witkowski, U., Naghsh, A., Saez-Pons, J., Herbrechtsmeier, S., and El-Habbal, M. (2011). A robot swarm assisting a human fire-fighter. *Advanced Robotics*, 25(1-2):93–117.

[Pinciroli et al., 2012] Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L. M., and Dorigo, M. (2012). ARGoS: a

modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295.

[Roli et al., 2011a] Roli, A., Arcaroli, C., Lazzarini, M., and Benedettini, S. (2011a). Boolean networks design by genetic algorithms. *arXiv preprint arXiv:1101.6018*.

[Roli et al., 2011b] Roli, A., Manfroni, M., Pinciroli, C., and Birattari, M. (2011b). On the design of boolean network robots. In *European Conference on the Applications of Evolutionary Computation*, pages 43–52. Springer.

[Silva et al., 2015a] Silva, F., Correia, L., and Christensen, A. L. (2015a). A case study on the scalability of online evolution of robotic controllers. In *Portuguese Conference on Artificial Intelligence*, pages 189–200. Springer.

[Silva et al., 2016] Silva, F., Correia, L., and Christensen, A. L. (2016). Evolutionary robotics. In *Evolutionary Robotics*, number 7. Scholarpedia.

[Silva et al., 2014] Silva, F., Urbano, P., and Christensen, A. L. (2014). Online evolution of adaptive robot behaviour. *International Journal of Natural Computing Research (IJNCR)*, 4(2):59–77.

[Silva et al., 2015b] Silva, F., Urbano, P., Correia, L., and Christensen, A. L. (2015b). odneat: An algorithm for decentralised online evolution of robotic controllers. *Evolutionary Computation*, 23(3):421–449.

[Silva et al., 2012] Silva, F., Urbano, P., Oliveira, S., and Christensen, A. L. (2012). odneat: An algorithm for distributed online, onboard evolution of robot behaviours. In *Artificial Life Conference Proceedings 12*, pages 251–258. MIT Press.

[Silva et al., 2017] Silva, F. G. d. et al. (2017). Evolutionary online behaviour learning and adaptation in robotic systems.

[Simoes and Barone, 2002] Simoes, E. and Barone, D. A. C. (2002). Predation: An approach to improving the evolution of real robots with a distributed evolutionary controller. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 1, pages 664–669. IEEE.

[Stanley and Miikkulainen, 2002] Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

[Trianni, 2006] Trianni, V. (2006). *On the evolution of self-organising behaviours in a swarm of autonomous robots*. PhD thesis, Citeseer.

[Watson et al., 2002] Watson, R. A., Ficici, S. G., and Pollack, J. B. (2002). Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and autonomous systems*, 39(1):1–18.