

# UNIVERSITÀ DI PISA

SCUOLA DI INGEGNERIA  
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE  
CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA

## ALGORITMI DI COORDINAMENTO DI SCIAMI DI DRONI BASATI SULL'INTELLIGENZA ALVEARE

ANNO ACCADEMICO 2018-2019

**RELATORI**

PROF.SSA GIGLIOLA VAGLINI  
PROF. MARIO G.C.A. CIMINO  
DOTT. MANILO MONACO

**CANDIDATO**

STEFANO PETROCCHI



## SOMMARIO

Algoritmi Bio-Inspirati .....	3
Introduzione .....	3
Applicazioni .....	3
Approccio .....	3
Modello NetLogo .....	3
Stati dei Robot .....	3
Esplorazione .....	3
Reclutamento ed Esecuzione .....	3
Modelli Matematici .....	4
Strategia di Esplorazione (Algorithm 1) .....	4
Descrizione .....	4
Diffusione del Feromone .....	4
Selezione Cella Successiva .....	5
Strategia di Reclutamento FTS (Algorithm 2) .....	6
Descrizione .....	6
Movimento .....	6
Scelta del Coordinatore .....	6
Strategia di Reclutamento PSO (Algorithm 3) .....	6
Descrizione .....	6
Movimento e Scelta del Coordinatore .....	7
Strategia di Reclutamento ABC (Algorithm 4) .....	7
Descrizione .....	7
Movimento .....	7
Scelta del Coordinatore .....	8
Scelte Effettuate sul Modello .....	9
Diffusione del feromone Quadrata o Circolare (*1) .....	9
Significato di Cella "Visitata" (*2) .....	9
Coefficiente di Randomizzazione (*3) .....	10
Robot in "Riserva" (*4) .....	10
Scelta del Coordinatore (*5) .....	10
Implementazione .....	11
Strategia di Esplorazione .....	11
Algorithm 1 .....	11
Smell Repulsive Pheromone .....	12
Release Repulsive Pheromone .....	12
Compute Repulsive Delta Intensity .....	13
Obstacle Ahead .....	13
Obstacle Avoidance Random .....	14
Check Targets Found .....	14
Find Target .....	15
Update Color Repulsive Patches .....	15
Strategia FTS .....	16
Algorithm 2 (fts_rr) .....	16
Target Choice .....	16
Compute Next Step .....	16
Strategia PSO .....	17
Algorithm 3 (pso_rr) .....	17
Compute Probability PSO .....	17
Compute Position PSO .....	18
Strategia ABC .....	18
Algorithm 4 (abc_rr) .....	18
Compute Probability .....	19
Compute Position .....	20
Strategia di Reclutamento .....	21
Send Request .....	21
Update State .....	21
Strategia di Esecuzione .....	23
Execution Algorithm .....	23
Execution Complete .....	24
Check Targets Complete .....	24
Sciadro con Elaborazione dei Target .....	25
Modello .....	25
Progetto Sciadro .....	25
Meccanismo di Elaborazione dei Target .....	26
Implementazione .....	27
Drone Logic .....	27
Patches Scheduling .....	27
Release Pheromone In Patch .....	27
Cambio di Stato .....	28
Check Possible Execution .....	28
Execution .....	28
Wait Execution .....	29
Testing .....	29
Piano di Collaudo .....	29
Algorithm 1 + Algorithm 2 (FTS) .....	30
"Dump" .....	30
"Rural Mine" .....	31
Algorithm 1 + Algorithm 3 (PSO) .....	32
"Dump" .....	32
"Rural Mine" .....	33
Algorithm 1 + Algorithm 4 (ABC) .....	34
"Dump" .....	34
"Rural Mine" .....	35
Sciadro con Elaborazione .....	36
"Dump" .....	36
"Rural Mine" .....	37
Osservazioni .....	38
Algorithm 1 + Algorithm 2 (FTS) .....	38
Algorithm 1 + Algorithm 3 (PSO) .....	39
Algorithm 1 + Algorithm 4 (ABC) .....	39
Confronto tra Sciadro con Elaborazione e gli Algoritmi Bio-Inspirati .....	40

## ALGORITMI BIO-INSPIRATI

### INTRODUZIONE

#### APPLICAZIONI

Molte applicazioni, relative a robot mobili ed autonomi, richiedono l'esplorazione di un ambiente alla ricerca di obiettivi statici, senza nessuna conoscenza a priori della topologia o della locazione dei target. Questi obiettivi possono essere incendi, mine, vittime di disastri o materiali pericolosi, che i robot devono gestire. In questi scenari una cooperazione tra i robot è necessaria per rilevare ed elaborare efficacemente tutti i target. Gli algoritmi trattati da *"Comparison of bio-inspired algorithms applied to the coordination of mobile robots considering the energy consumption"* permettono il coordinamento di sciami di robot finalizzati allo svolgimento di tali applicazioni.

#### APPROCCIO

Questi algoritmi prevedono un approccio fortemente decentralizzato al problema, vengono quindi utilizzate tecniche di programmazione emergente a tale scopo. I singoli robot non possiedono una visione generale del problema ma si muovono autonomamente in base a semplici *procedure*. L'attenersi contemporaneamente a queste procedure e l'interazione con altri robot e l'ambiente, permette l'emergere di un'organizzazione complessiva di tipo alveare, in cui i robot possono svolgere attività complesse con grande efficienza.

I robot passano attraverso vari *stati* in cui alcune procedure possono essere momentaneamente sospese in favore di altre.

#### MODELLO NETLOGO

L'area di esplorazione è stata modellata attraverso un piano bidimensionale suddiviso in diverse *celle* (equivalenti alle *patch* di NetLogo). I robot sono rappresentati dalle *turtles* che possono muoversi solamente con spostamenti quantizzati da una patch ad un'altra appartenente al proprio *neighborhood* (le otto patch circostanti). Il meccanismo di *obstacle avoidance* prevede che il robot, in presenza di un ostacolo o di un altro robot davanti a sé, scelga casualmente una delle patch del *neighborhood* e si sposti su di essa. Lo scorrere del tempo è suddiviso in istanti quantizzati che equivalgono ai *ticks* di NetLogo.

## STATI DEI ROBOT

#### ESPLORAZIONE

Lo **stato di esplorazione** ha lo scopo di esplorare la regione e individuare alcuni obiettivi sparsi all'interno di essa, viene principalmente implementato attraverso una strategia basata su colonie di formiche. In natura, le formiche depositano un particolare tipo di sostanza chimica nel territorio, il *feromone*, mentre si muovono. Esistono differenti tipi di feromone, ognuno dei quali avente un significato specifico, grazie al quale le formiche sono in grado di prendere delle decisioni. Viene utilizzata una versione virtuale di tale feromone per guidare i robot durante l'esplorazione. Si assume che i robot non conoscano la propria posizione, o la posizione di altri robot, nell'area da esplorare, ma si muovano in accordo a ciò che possono percepire dall'ambiente circostante.

#### RECLUTAMENTO ED ESECUZIONE

Quando un robot individua un obiettivo entra nello **stato di coordinatore** per quel target e inizia un processo di reclutamento per attrarre altri robot a sé. I robot reclutati entrano momentaneamente in uno

**stato di attesa**, fino ad essere in numero sufficiente ad entrare in uno **stato di esecuzione**, in cui svolgono le operazioni specifiche per quel target. A tale scopo il robot coordinatore utilizza una comunicazione wireless a raggio limitato per inviare le richieste di reclutamento ai robot circostanti. I robot che ricevono richieste di reclutamento decidono in maniera autonoma se e da quale coordinatore dirigersi. L'algoritmo di reclutamento si ispira al comportamento di colonie di api.

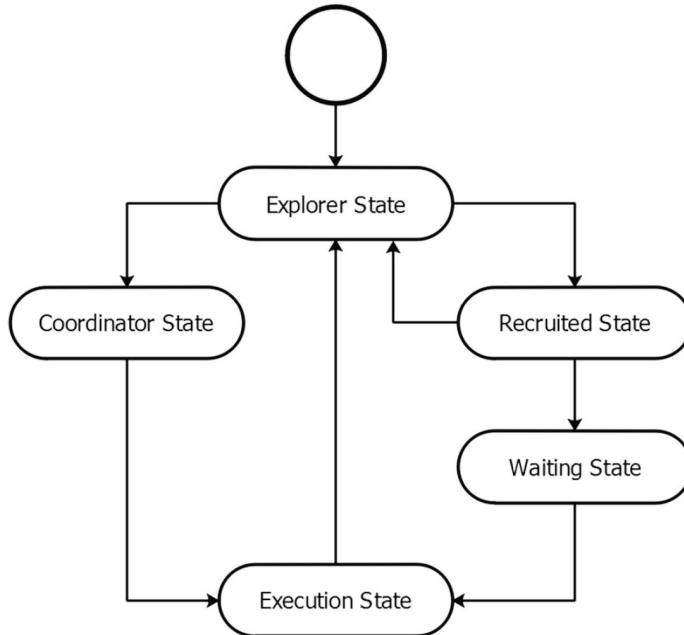


Figura 1: Possibili stati del robot

## MODELLO MATEMATICO

Verranno evidenziati di **azzurro** i coefficienti che possono essere soggetti di un'eventuale ottimizzazione.

## STRATEGIA DI ESPLORAZIONE (ALGORITHM 1)

### DESCRIZIONE

Per minimizzare la possibilità che i robot possano visitare più volte una stessa area della regione d'interesse, è stato introdotto un meccanismo feromonico di repulsione negativa. Durante l'esplorazione, il feromone è depositato immediatamente dopo lo spostamento su una nuova cella. I robot scelgono come cella successiva del loro spostamento quella, appartenente al proprio neighborhood, che presenta la minor quantità di feromone. Il feromone depositato dai robot su una particolare cella si diffonde anche in quelle circostanti fino ad una certa distanza  $R_s$ .

### DIFFUSIONE DEL FEROMONE

Il robot  $k$  all'iterazione  $t$  è localizzato nella cella di coordinate  $(x_k^t, y_k^t)$ , la quantità di feromone che il kappesimo robot deposita sulla cella  $c$  di coordinate  $(x, y)$ , entro un raggio  $R_s$  (\*1), è data da:

$$\Delta\tau_c^{k,t} = \begin{cases} \Delta\tau_0 e^{\frac{-r_{kc}}{a_1}} - \frac{\varepsilon}{a_2} & \text{per } r_{kc} \leq R_s \\ 0 & \text{altrimenti} \end{cases}$$

Dove  $r_{kc}$  è la distanza tra il kappesimo robot e la cella  $c$  ed è definita come:

$$r_{kc} = \sqrt{(x_k^t - x)^2 + (y_k^t - y)^2}$$

Inoltre,  $\Delta\tau_0$  è la quantità di feromone che il drone diffonde sulla cella sopra la quale è posizionato (la massima che può diffondere),  $\varepsilon \in [0,1]$  è una variabile euristica (rumore). Le costanti  $a_1$  e  $a_2$  riducono o aumentano l'effetto del rumore e del feromone.

Più robot possono depositare feromone nell'ambiente allo stesso tempo: la quantità totale di feromone depositato su una cella risulterà essere la somma della quantità di feromone depositato da ogni robot sulla stessa.

La quantità di feromone depositata su una cella non è fissa, il feromone può evaporare al passare del tempo. Il tasso di evaporazione è dato da  $\rho$ . La quantità di feromone evaporata nella cella  $c$  all'istante  $t$  è data da:

$$\xi_t^c = \rho \cdot \tau_c^t$$

Dove  $\tau_c^t$  è la quantità totale di feromone depositato sulla cella  $c$  all'istante  $t$ .

Per il calcolo di  $\rho$  viene utilizzato il coefficiente  $ERTU\%$  (Evaporation Rate Time Unit) che riguarda il tasso di evaporazione per unità di tempo trascorse. Sia  $t_v$  l'ultima volta in cui una cella è stata visitata (\*2) dall'istante corrente  $t$ ,  $(t - t_v)$  è il tempo trascorso dall'ultima visita della cella. Dati questi parametri,  $\rho$  può essere calcolato tramite:

$$\rho = (t - t_v) ERTU\%$$

Considerando l'evaporazione e la diffusione in accordo alla distanza del feromone, l'ammontare totale di feromone sulla cella  $c$  all'istante  $t$  è dato da:

$$\tau_c^t = \tau_c^{t-1} - \xi_c^{t-1} + \sum_{k=1}^{N^R} \Delta\tau_c^{k,t}$$

Dove  $N^R$  è il numero totale di robot.

Il valore feromonico iniziale di ogni cella è settato a zero.

### SELEZIONE CELLA SUCCESSIVA

Ciascun robot  $k$ , ad ogni istante  $t$ , è posizionato in una particolare cella  $c_k^t$  che è circondata da un insieme di celle accessibili  $N(c_k^t)$ , le otto celle del neighborhood. Ogni robot percepisce la quantità di feromone depositato in tali celle e sceglie in quale muoversi allo step successivo. La probabilità, ad ogni istante  $t$ , per un robot  $k$ , di muoversi dalla cella  $c_k^t$  alla cella  $c \in N(c_k^t)$  è calcolata come:

$$p(c|c_k^t) = \frac{(\tau_c^t)^\varphi (\eta_c^t)^\lambda}{\sum_{b \in N(c_k^t)} (\tau_b^t)^\varphi (\eta_b^t)^\lambda} \quad \forall c \in N(c_k^t)$$

Dove  $(\tau_c^t)^\varphi$  è la quantità di feromone nella cella  $c$  all'istante  $t$ , mentre  $(\eta_c^t)^\lambda$  è una variabile euristica che evita che un robot possa rimanere intrappolato in un punto di minimo locale. Inoltre,  $\varphi$  e  $\lambda$  sono due costanti che servono per bilanciare il peso dato al feromone o alla variabile euristica rispettivamente.

Il robot  $k$  si muove nella cella che soddisfa questa condizione:

$$c = \min[p(c|c_k^t)]$$

In questo modo i robot preferiranno le aree meno frequentate e sarà più probabile che si dirigano in zone non ancora esplorate.

## STRATEGIA DI RECLUTAMENTO FTS (ALGORITHM 2)

### DESCRIZIONE

Il *Firefly Algorithm (FA)* è un metodo di ottimizzazione stocastico bio-inspirato che cerca di imitare il comportamento di uno sciame di luciole. Le due regole fondamentali riguardano l'intensità della luce emessa e l'attrazione delle luciole dalla stessa. L'attrazione è proporzionale alla luminosità, le luciole che emettono minor luce saranno attratte da quelle che ne emettono maggiormente. L'intensità luminosa si affievolisce con il quadrato della distanza, perciò le luciole presentano un raggio visivo limitato. L'algoritmo 2 è una versione modificata di FA in modo da poter essere applicata in un ambiente a tempi discreti e bidimensionale.

### MOVIMENTO

Sia  $\beta_0$  l'attrattività iniziale alla distanza  $r_{kz} = 0$  dove  $k, z$  sono due robot distinti;  $\alpha$  un coefficiente di randomizzazione (\*3);  $\sigma$  un fattore di scala che controlla la distanza di visibilità (deve essere generalmente posto ad 1);  $\gamma$  un coefficiente di assorbimento che controlla il decrescere dell'intensità luminosa.

Quando un robot  $k$ , all'iterazione  $t$ , nella cella  $c_k^t$  di coordinate  $(x_k^t, y_k^t)$  riceve un pacchetto da un robot coordinatore  $z$  in prossimità di un target, si muoverà nello step  $t + 1$  alla posizione  $(x_k^{t+1}, y_k^{t+1})$  in accordo alla seguente regola:

$$\begin{cases} x_k^{t+1} = x_k^t + 1 & \text{se } \beta_0 e^{-\gamma r_{kz}^2} (x_z - x_k^t) + \alpha \left( \sigma - \frac{1}{2} \right) > 0 \\ x_k^{t+1} = x_k^t - 1 & \text{se } \beta_0 e^{-\gamma r_{kz}^2} (x_z - x_k^t) + \alpha \left( \sigma - \frac{1}{2} \right) < 0 \\ x_k^{t+1} = x_k^t & \text{se } \beta_0 e^{-\gamma r_{kz}^2} (x_z - x_k^t) + \alpha \left( \sigma - \frac{1}{2} \right) = 0 \end{cases}$$
  

$$\begin{cases} y_k^{t+1} = y_k^t + 1 & \text{se } \beta_0 e^{-\gamma r_{kz}^2} (y_z - y_k^t) + \alpha \left( \sigma - \frac{1}{2} \right) > 0 \\ y_k^{t+1} = y_k^t - 1 & \text{se } \beta_0 e^{-\gamma r_{kz}^2} (y_z - y_k^t) + \alpha \left( \sigma - \frac{1}{2} \right) < 0 \\ y_k^{t+1} = y_k^t & \text{se } \beta_0 e^{-\gamma r_{kz}^2} (y_z - y_k^t) + \alpha \left( \sigma - \frac{1}{2} \right) = 0 \end{cases}$$

Bisogna far notare che i target sono statici e un robot può ricevere più di una richiesta alla volta, il coefficiente di randomizzazione  $\alpha$  viene utilizzato in tale situazione per evitare che i robot si dirigano esclusivamente su un coordinatore.

### SCELTA DEL COORDINATORE

A meno del coefficiente di randomizzazione, il coordinatore scelto è quello più attrattivo, l'attrattività  $\beta$  viene calcolata tramite:

$$\beta = \beta_0 e^{-\gamma r_{kz}^2}$$

## STRATEGIA DI RECLUTAMENTO PSO (ALGORITHM 3)

### DESCRIZIONE

*Particle Swarm Optimization (PSO)* è una tecnica di ottimizzazione che utilizza una popolazione di agenti multipli. Questa tecnica è inspirata al movimento di stormi di uccelli e alle loro interazioni all'interno dello stormo stesso.

---

## MOVIMENTO E SCELTA DEL COORDINATORE

Data la velocità  $v_k^t$  del robot  $k$  all'istante  $t$ , la nuova velocità  $v_k^{t+1}$  può essere calcolata mediante:

$$\begin{cases} v_{xk}^{t+1} = \omega v_{xk}^t + r \cdot c(x_z - x_k^t) \\ v_{yk}^{t+1} = \omega v_{yk}^t + r \cdot c(y_z - y_k^t) \end{cases}$$

Dove  $\omega$  è il coefficiente inerziale,  $c$  il coefficiente di accelerazione e  $r \in [0,1]$  è un numero generato casualmente (\*3). La velocità  $v_k^{t+1}$  non è però utilizzata per il calcolo della velocità effettiva del robot ma come coefficiente per determinarne lo spostamento nella formula successiva.

Quando un robot  $k$ , all'iterazione  $t$ , nella cella  $c_k^t$  di coordinate  $(x_k^t, y_k^t)$  riceve un pacchetto da un robot coordinatore  $z$  che ha trovato un target, si muoverà nello step  $t + 1$  alla posizione  $(x_k^{t+1}, y_k^{t+1})$  in accordo alla seguente regola:

$$\begin{cases} x_k^{t+1} = x_k^t + 1 & \text{se } v_{xk}^{t+1} > 0 \\ x_k^{t+1} = x_k^t - 1 & \text{se } v_{xk}^{t+1} < 0 \\ x_k^{t+1} = x_k^t & \text{se } v_{xk}^{t+1} = 0 \end{cases}$$

$$\begin{cases} y_k^{t+1} = y_k^t + 1 & \text{se } v_{yk}^{t+1} > 0 \\ y_k^{t+1} = y_k^t - 1 & \text{se } v_{yk}^{t+1} < 0 \\ y_k^{t+1} = y_k^t & \text{se } v_{yk}^{t+1} = 0 \end{cases}$$

Quando un robot riceve richieste da più coordinatori simultaneamente si muoverà verso quello più vicino.

## STRATEGIA DI RECLUTAMENTO ABC (ALGORITHM 4)

---

### DESCRIZIONE

Questo approccio di reclutamento si ispira all'algoritmo *Artificial Bee Colony (ABC)* e segue il comportamento delle api in cerca di una fonte di nutrimento di qualità. Nell'algoritmo ABC la distribuzione di fonti di cibo, all'interno del territorio di esplorazione, può essere modificata dalle api stesse nel corso del tempo. L'algoritmo utilizza l'insieme delle api per trovare la soluzione ottima al problema. L'algoritmo 4 è una versione bidimensionale di ABC in cui le fonti di cibo sono state sostituite con l'insieme dei coordinatori che tentano di reclutare concorrentemente uno stesso robot nel loro raggio wireless.

---

### MOVIMENTO

Sia  $\phi \in [-1,1]$  un coefficiente di randomizzazione scelto in maniera casuale, ogni robot  $k$  nel raggio di reclutamento di un coordinatore  $z$ , all'istante  $t$ , si sposta secondo:

$$\begin{cases} x_k^{t+1} = x_k^t + \phi(x_z^t - x_k^t) \\ y_k^{t+1} = y_k^t + \phi(y_z^t - y_k^t) \end{cases}$$

Dove  $(x_z, y_z)$  sono le coordinate del coordinatore sul piano bidimensionale,  $(x_k^t, y_k^t)$  è la posizione corrente del robot  $k$  ed infine  $(x_k^{t+1}, y_k^{t+1})$  è la posizione al passo successivo del robot reclutato.

Spostandosi su un piano bidimensionale, esistono tre possibili aggiornamenti dei valori delle coordinate:  $\{-1, 0, 1\}$ , che portano alle seguenti possibilità di movimento:

$$\begin{cases} x_k^{t+1} = x_k^t + 1 & \text{se } [\phi(x_k^t - x_z) > 0] \\ x_k^{t+1} = x_k^t - 1 & \text{se } [\phi(x_k^t - x_z) < 0] \\ x_k^{t+1} = x_k^t & \text{se } [\phi(x_k^t - x_z) = 0] \end{cases}$$

$$\begin{cases} y_k^{t+1} = y_k^t + 1 & \text{se } [\phi(y_k^t - y_z) > 0] \\ y_k^{t+1} = y_k^t - 1 & \text{se } [\phi(y_k^t - y_z) < 0] \\ y_k^{t+1} = y_k^t & \text{se } [\phi(y_k^t - y_z) = 0] \end{cases}$$

Essenzialmente si vengono a verificare tre casi (\*3): se  $\phi < 0$  il robot che entra nel raggio di reclutamento di un coordinatore si sposterà verso di esso, se  $\phi > 0$  il robot tenderà ad uscire dal raggio di reclutamento, se  $\phi = 0$  il robot si fermerà sul bordo del raggio di reclutamento in “riserva” (\*4).

### SCELTA DEL COORDINATORE

Se un robot riceve più di una richiesta dovrà decidere quale coordinatore prendere come riferimento.

Quando un robot  $k$  nella cella  $c_k^t$  riceve una richiesta di reclutamento da un coordinatore nella cella  $c_z^t$ , il costo del coordinatore  $z$ , per il robot  $k$ , allo step  $t$ , è calcolato come la distanza euclidea tra il robot e il coordinatore:

$$r_{kz} = \sqrt{(x_k^t - x_z)^2 + (y_k^t - y_z)^2}$$

L'utilità del coordinatore  $z$ , per il robot  $k$ , è definita come il reciproco della distanza:

$$\mu_z^k = \frac{1}{r_{kz}}$$

La probabilità che il robot  $k$  scelga il coordinatore  $z$  può essere calcolata come:

$$p_z^k = \frac{\mu_z^k}{\sum_{b=1}^{RR_k} \mu_b^k}$$

Dove  $RR_k$  è l'insieme di richieste di reclutamento ricevute dal robot  $k$  (\*5).

E possibile dimostrare che:

$$\sum_{z=1}^{RR_k} p_z^k = 1$$

Viene pertanto utilizzata la regola della *spinning wheel*: ad ogni coordinatore è associata una probabilità di essere scelto; una volta calcolate tutte le probabilità, il robot  $k$  sceglierà il coordinatore facendo girare la ruota. Dopo aver scelto il coordinatore, il robot si sposterà secondo le regole di movimento descritte precedentemente.

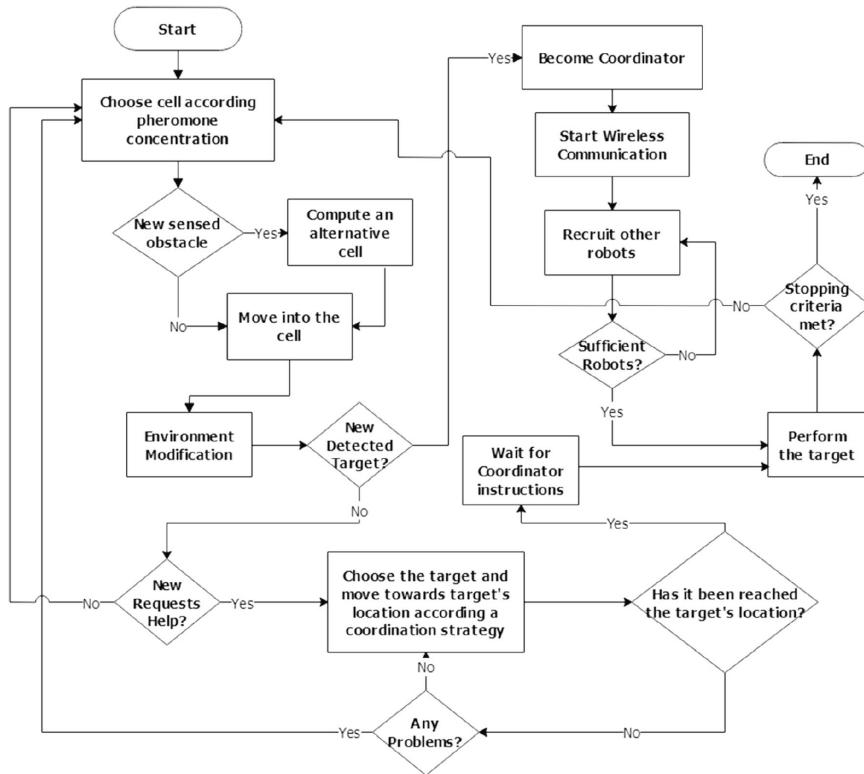


Figura 2: Diagramma di flusso del modello

## SCELTE EFFETTUATE SUL MODELLO

### DIFFUSIONE DEL FEROMONE QUADRATA O CIRCOLARE (\*1)

Data la formula di diffusione del feromone:

$$\Delta\tau_c^{k,t} = \begin{cases} \Delta\tau_0 e^{\frac{-r_{kc}}{a_1}} - \frac{\varepsilon}{a_2} & \text{per } r_{kc} \leq R_s \\ 0 & \text{altrimenti} \end{cases}$$

Nel modello matematico  $R_s$  viene considerato come un raggio circolare, però l'implementazione che si evince dalle immagini fornite nella ricerca utilizza un raggio "quadrato", come mostrato nella figura a fianco. Pertanto, è stato deciso di implementare due versioni della formula che utilizza entrambi i modelli circolare e quadrato. Nel modello quadrato i robot tendono ad avere delle oscillazioni nel loro moto esplorativo a causa dell'uguale quantità di feromone presente nelle tre patch innanzi ad essi.

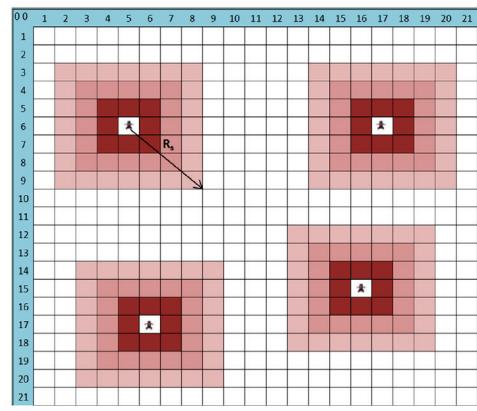


Figura 3: raggio quadrato nella diffusione del feromone

### SIGNIFICATO DI CELLA "VISITATA" (\*2)

Nella versione precedente dell'algoritmo di esplorazione, il valore  $(t - t_v)$  era stata interpretato come il tempo trascorso dall'ultima volta in cui un robot fosse passato *sopra* una data cella.

Nella nuova versione dell'algoritmo, il valore  $(t - t_v)$  viene azzerato ogni volta che una cella *entra nel raggio d'azione feromonico  $R_s$*  del robot.

Date le leggi di aggiornamento del valore feromonico di una cella  $c$ , all'istante  $t$ :

$$\tau_c^t = \tau_c^{t-1} - \xi_c^{t-1} + \sum_{k=1}^{N^R} \Delta\tau_c^{k,t}$$

$$\xi_t^c = \rho \cdot \tau_c^t$$

$$\rho = (t - t_v) ERTU\%$$

Il valore  $\sum_{k=1}^{N^R} \Delta\tau_c^{k,t}$  rappresenta la quantità di feromone diffusa nella cella, se la distanza tra essa e il robot è minore di  $R_s$ .

Per  $(t - t_v) \gg \frac{1}{ERTU\%}$  si ha che  $\xi_c^{t-1} \gg \tau_c^{t-1}$ , in quanto  $\rho \gg 1$ ; in particolare, per  $(t - t_v)$  abbastanza

grande, si ha che  $\xi_c^{t-1} > \sum_{k=1}^{N^R} \Delta\tau_c^{k,t} + \tau_c^{t-1}$ , per cui si avrà  $\tau_c^t = 0$  ( $\tau_c^t \geq 0$ ).

Se il valore di  $(t - t_v)$  non venisse azzerato nel raggio  $R_s$  del robot, ma soltanto quando esso passa sopra una determinata cella, si avrebbe che, per un tempo  $t$  abbastanza grande ( $\gg \frac{1}{ERTU\%}$ ), il valore feromonico delle celle intorno al robot, sopra le quali non è passato, risulterebbe nullo anche nell'istante successivo all'immissione di nuovo feromone da parte dello stesso, in quanto il tasso di evaporazione  $\xi_c^{t-1}$  risulterebbe comunque maggiore della quantità di nuovo feromone emesso dal robot.

In seguito a questi ragionamenti si è deciso di effettuare le sopracitate modifiche all'algoritmo.

#### COEFFICIENTE DI RANDOMIZZAZIONE (\*3)

Nella precedente versione dell'algoritmo di reclutamento, il coefficiente di randomizzazione  $\phi$  veniva ricalcolato ad ogni istante temporale  $t$ .

Nella nuova versione dell'algoritmo, il valore di  $\phi$  viene ora ricalcolato solo se è stato scelto un nuovo coordinatore, o al passaggio dallo stato di esploratore a quello di reclutamento.

Il valore di  $\phi$  porta il robot a muoversi in uno dei seguenti modi: rimanere fermo ( $\phi = 0$ ), avanzare di una cella verso il coordinatore ( $\phi < 0$ ), allontanarsi di una cella dal coordinatore ( $\phi > 0$ ).

Se  $\phi$  viene ricalcolato ad ogni passo è evidente che il robot tenderà a rimanere localizzato su una stessa posizione (può andare in maniera casuale avanti, indietro o rimanere fermo).

Se invece  $\phi$  viene calcolato con il secondo metodo, il robot seguirà un certo tipo di comportamento: rimanendo fermo, avvicinandosi al coordinatore o allontanandosi da esso.

Seguendo un ragionamento analogo si è scelto di ricalcolare il valore di  $\alpha$  e  $r$  solo quando il robot entra in un nuovo raggio di reclutamento.

#### ROBOT IN "RISERVA" (\*4)

Come conseguenza della decisione (\*3), per  $\phi = 0$ , alcuni robot possono rimanere fermi all'interno dell'area di reclutamento del coordinatore. Per evitare eventuali situazioni di deadlock in cui non rimangono sufficienti robot per poter iniziare la fase di esecuzione, il valore di  $\phi$  viene ricalcolato per sicurezza dopo un certo numero di tick (*maxIdleTicks*), se esso è uguale a zero.

#### SCELTA DEL COORDINATORE (\*5)

In *"Comparison of bio-inspired algorithms applied to the coordination of mobile robots considering the energy consumption"* è chiaramente evidenziato come il coordinatore debba essere scelto nuovamente ad ogni iterazione  $t$ .

Questo ricalcolo, sommato agli effetti del coefficiente di randomizzazione  $\phi$ , porta i robot a muoversi da un coordinatore ad un altro alternatamente.

In questa implementazione è stato deciso di ricalcolare il coordinatore scelto solamente nei seguenti casi: 1) appare un nuovo coordinatore il cui raggio di reclutamento ricade sul robot, 2) il robot passa dallo stato di esploratore a quello di reclutamento, 3) il robot esce dal raggio di reclutamento del coordinatore scelto.

Viene comunque fornita anche un'implementazione che esegue il ricalcolo del coordinatore a ogni iterazione.

## IMPLEMENTAZIONE

Di seguito viene riportata l'implementazione NetLogo dei modelli sopra descritti e delle scelte effettuate e descritte sugli stessi.

## STRATEGIA DI ESPLORAZIONE

### ALGORITHM 1

```

to algorithm_1
  ask drones with [drone.state = "EXPLORER"] [
    let nextPatch smellRepulsivePheromone
    if (nextPatch != nobody) [
      face nextPatch
    ]
    obstacleAvoidanceRandom
    findTarget
  ]
  ask drones with [drone.state = "EXPLORER"] [
    ;;releaseRepulsivePheromoneRound
    releaseRepulsivePheromoneSquare
  ]
  ask patches [
    updateRepulsivePheromone
  ]
  ask patches [
    updateRepulsiveVariables
  ]
  checkTargetsFounded
end

```

1. Ogni drone con stato “EXPLORER”:
  - 1.1. Seleziona la prossima patch a cui puntare mediante la procedura *smellRepulsivePheromone*
  - 1.2. Se è stata selezionata una patch, si gira verso di essa
  - 1.3. Si muove secondo la procedura *obstacleAvoidanceRandom*
  - 1.4. Controlla se ci sono target vicini mediante *findTarget*
2. Ogni drone con stato “EXPLORER”:
  - 2.1. Rilascia il feromone repulsivo mediante *releaseRepulsivePheromoneSquare* o *releaseRepulsivePheromoneRound* (\*1)
3. Ogni patch:
  - 3.1. Aggiorna i propri valori di feromone tramite *updateRepulsivePheromone*
4. Ogni patch:
  - 4.1. Aggiorna i valori delle variabili necessarie al calcolo del feromone tramite *updateRepulsiveVariables*
5. Vengono controllati i target localizzati mediante *checkTargetsFouded*

---

## SMELL REPULSIVE PHEROMONE

```

to-report smellRepulsivePheromone

let denominator 0
let firstTerm 0
let secondTerm 0
let tmpDenominator 0
let minIntensityPatch nobody

ask neighbors [
  set firstTerm track.repulsiveIntensity ^ phi
  set secondTerm eta ^ lambda
  set tmpDenominator firstTerm * secondTerm
  set denominator denominator + tmpDenominator
]

let maxIntensity 0
let maxIntensityPatch max-one-of patches [ track.repulsiveIntensity ]
ask maxIntensityPatch [
  set maxIntensity track.repulsiveIntensity
]

let minProb (maxIntensity ^ phi) * (eta ^ lambda)
let numerator 0
let tmpProb 0
ask neighbors [
  if (denominator != 0) [
    set firstTerm track.repulsiveIntensity ^ phi
    set secondTerm eta ^ lambda
    set numerator firstTerm * secondTerm
    set tmpProb numerator / denominator
    if (tmpProb < minProb) [
      set minIntensityPatch self |
      set minProb tmpProb
    ]
  ]
]
report minIntensityPatch
end

```

Date le espressioni per il calcolo della patch successiva ( $c$ ):

$$p(c|c_k^t) = \frac{(\tau_c^t)^\varphi (\eta_c^t)^\lambda}{\sum_{b \in N(c_k^t)} (\tau_b^t)^\varphi (\eta_b^t)^\lambda}$$

$$c = \min[p(c|c_k^t)]$$

1. Un ciclo su ogni patch del neighborhood del drone permette il calcolo del valore del denominatore della prima espressione
2. Viene temporaneamente selezionata la patch con il valore feromonico maggiore
3. Per ogni patch appartenente al neighborhood del drone, il cui denominatore è diverso da zero:
  - 3.1. Viene calcolata la probabilità associata ad essa, secondo la prima espressione
  - 3.2. Se la probabilità calcolata è minore di quella associata alla patch temporaneamente scelta:
    - 3.2.1. Diventa la nuova patch temporaneamente scelta
4. Viene restituita la patch con probabilità minore tra quelle appartenenti al neighborhood

---

## RELEASE REPULSIVE PHEROMONE

```

to releaseRepulsivePheromoneRound
  let dist 0
  let xcoord xcor
  let ycoord ycor
  ask patches with [ distance myself <= mark.sensingRange ][
    set track.sumRepulsiveDeltaIntensity
      track.sumRepulsiveDeltaIntensity
      + computeRepulsiveDeltaIntensity (distance myself - 1)
    set ticksFromLastVisit 0
  ]

```

```

to releaseRepulsivePheromoneSquare
  let dist 0
  let xcoord xcor
  let ycoord ycor
  ask patches with [ abs(pxcor - xcoord) <= mark.sensingRange
                      and abs(pycor - ycoord) <= mark.sensingRange] [
    ifelse( abs(pxcor - xcoord) > abs(pycor - ycoord)) [
      set dist abs(pxcor - xcoord)
    ] [
      set dist abs(pycor - ycoord)
    ]
    set track.sumRepulsiveDeltaIntensity
      track.sumRepulsiveDeltaIntensity
        + computeRepulsiveDeltaIntensity (dist)
    set ticksFromLastVisit 0
  ]
end

```

1. Entro un raggio circolare o quadrato (\*1) dal robot:

- 1.1. Viene aggiornato il valore feromonico associato a queste patch secondo l'espressione:

$$\tau_c^t = \tau_c^{t-1} - \xi_c^{t-1} + \sum_{k=1}^{N^R} \Delta\tau_c^{k,t}$$

- 1.2. Viene azzerato il valore di *ticksFromLastVisit* in seguito alla decisione adottata al (\*2)

## COMPUTE REPULSIVE DELTA INTENSITY

```

to-report computeRepulsiveDeltaIntensity [ dist ]
  set epsilon random-float 1
  ifelse (dist <= mark.sensingRange) [
    let exponent (- dist) / a1
    let firstTerm mark.repulsiveStartIntensity * (e ^ exponent)
    let secondTerm epsilon / a2

    report (firstTerm + secondTerm)
  ] [
    report 0
  ]
end

```

- 1) Viene calcolato il valore del rumore  $\varepsilon \in (0,1)$
- 2) Se la distanza è minore del raggio di diffusione  $R_s$  del feromone:
  - a) Viene calcolato e restituito il valore  $\Delta\tau_c^{k,t}$  seguendo la formula:

$$\Delta\tau_c^{k,t} = \Delta\tau_0 e^{\frac{-r_{kc}}{a_1}} - \frac{\varepsilon}{a_2} \quad per \ r_{kc} \leq R_s$$

- b) Altrimenti viene restituito il valore 0

## OBSTACLE AHEAD

```

to-report obstacleAhead
  let isObstacle false
  ifelse(patch-ahead 1 != nobody)[
    if any? turtles-on patch-ahead 1 [
      set isObstacle true
    ]
    ask patch-ahead 1[
      if(obstacle = true)[
        set isObstacle true
      ]
    ]
  ]
  set isObstacle true
]
report isObstacle
end

```

- 1) Se la patch davanti al drone:
  - a) È fuori dal campo di simulazione
  - b) È occupata da un altro drone
  - c) È un ostacolo
    - 1.1. Restituisce *true*
    - 1.2. Restituisce *false*
  - d) Altrimenti

### OBSTACLE AVOIDANCE RANDOM

```

to obstacleAvoidanceRandom
  let isObstacle? obstacleAhead
  ifelse (isObstacle? = false) [
    move-to patch-ahead 1
  ]
  [
    let clearNeighbor nobody
    let patchFinded false
    ask neighbors [
      if(not patchFinded = true)[
        set clearNeighbor self
        ask myself[
          face clearNeighbor
          set isObstacle? obstacleAhead
          if(isObstacle? = false) [
            move-to patch-ahead 1
            set patchFinded true
          ]
        ]
      ]
    ]
  ]
end

```

- a. Se davanti a sé il drone non ha ostacoli:
  - a.1. Si muove sulla patch di fronte
- b. Se davanti a sé il drone ha un ostacolo:
  - b.1. Per ogni patch appartenente al neighborhood (NetLogo le sceglie in ordine casuale), se non è stata ancora trovata una patch libera:
    - b.1.1. Il drone si gira verso la patch
    - b.1.2. Se la patch davanti al drone stesso è libera:
      - b.1.2.1. Si muove su tale patch

### CHECK TARGETS FOUND

```

to checkTargetsFound
  ask patches [
    if(target = true and status = "found" and execution.complete = false)[
      set pcolor pink
    ]
  ]
end

```

1. Per ogni patch, se:
  - 1.1. È un target
  - 1.2. È stato trovato
  - 1.3. L'esecuzione su di esso non è conclusa
    - 1.3.1. Per debug assume la colorazione rosa

---

## FIND TARGET

```

to findTarget
  ask patches in-radius drone.executionRange [
    if(target = true and not(status = "found"))[
      set status "found"
      ask myself[
        set drone.state "COORDINATOR"
        set color yellow
        ask drones in-radius drone.wirelessRange [
          set drone.targetSelected []
        ]
      ]
    ]
  ]
end

```

1. Tutte le patch nel raggio di esecuzione del drone:

- 1.1. Se sono dei target e non sono stati trovati:

- 1.1.1. Cambiano il loro stato in *found*

- 1.1.2. Il drone stesso:

- 1.1.2.1. Diventa coordinatore

- 1.1.2.2. Assume la colorazione gialla

- 1.1.2.3. Tutti i droni nel raggio wireless del drone coordinatore:

- 1.1.2.3.1. Se avevano già scelto il drone coordinatore di riferimento, ripetono il processo di selezione (vedere il punto (\*5))

---

## UPDATE COLOR REPULSIVE PATCHES

```

to updateColorRepulsivePatches
  if(pcolor != red and pcolor != 5)[
    if(track.repulsiveIntensity >= 80.0)[
      set pcolor 41
    ]
    if(track.repulsiveIntensity > 40.0 and track.repulsiveIntensity < 80.0)[
      set pcolor 42
    ]
    if(track.repulsiveIntensity > 20.0 and track.repulsiveIntensity < 40.0)[
      set pcolor 43
    ]
    if(track.repulsiveIntensity > 5.0 and track.repulsiveIntensity < 20.0)[
      set pcolor 44
    ]
    if(track.repulsiveIntensity > 4.0 and track.repulsiveIntensity < 5.0)[
      set pcolor 45
    ]
    if(track.repulsiveIntensity > 3.0 and track.repulsiveIntensity < 4.0)[
      set pcolor 46
    ]
    if(track.repulsiveIntensity > 2.0 and track.repulsiveIntensity < 3.0)[
      set pcolor 47
    ]
    if(track.repulsiveIntensity > 1.0 and track.repulsiveIntensity < 2.0)[
      set pcolor 48
    ]
    if(track.repulsiveIntensity > 0.0 and track.repulsiveIntensity < 1)[
      set pcolor 49
    ]
    if(track.repulsiveIntensity = 0.0)[
      set pcolor 0
    ]
  ]
end

```

Procedura di debug per evidenziare la quantità di feromone depositata sulle varie patch, gli intervalli di intensità feromonica con colorazione differente sono stati scelti in maniera euristica.

## STRATEGIA FTS

### ALGORITHM 2 (FTS\_RR)

```

to fts_rr
    updateState
    ask drones with [drone.state = "RECRUITED"] [
        let index 0
        set index targetChoice drone.targetPatchList
        set drone.targetSelected item index drone.targetPatchList
        computeNextStep drone.targetSelected
    ]
end

```

1. Le variabili dei robot vengono aggiornate mediante *updateState*
2. Per ogni drone con stato “RECRUITED”:
  - 2.1. Viene calcolato il coordinatore con l’attrattività maggiore
  - 2.2. Viene calcolata la prossima posizione del robot in relazione al coordinatore scelto

### TARGET CHOICE

```

to-report targetChoice[targetList]
    let dist 0
    let xcoord 0
    let ycoord 0
    let beta 0
    let maxAttractiveness 0
    let chosenTarget 0
    let index 0
    foreach targetList [
        set xcoord item 0 ?
        set ycoord item 1 ?
        set dist sqrt((pxcor - xcoord) ^ 2 + (pycor - ycoord) ^ 2)
        set beta beta0 * e ^ (- gamma * dist ^ 2)
        if (beta > maxAttractiveness) [
            set maxAttractiveness beta
            set chosenTarget index
        ]
        set index index + 1
    ]
    set drone.beta maxAttractiveness
    report chosenTarget
end

```

1. Per ogni coordinatore (elementi nella lista dei target):
  - 1.1. Viene calcolata l’attrattività mediante la formula:

$$\beta = \beta_0 e^{-\gamma r_{kz}^2}$$

2. Viene calcolato e restituito il coordinatore con attrattività maggiore

### COMPUTE NEXT STEP

```

to computeNextStep [targetCoordinates]
    let targetx (item 0 targetCoordinates)
    let targety (item 1 targetCoordinates)
    let nextXCoor xcor
    let nextYCoor ycor
    if(drone.beta * (targetx - xcor) + drone.apha_rr * (sigma - (1 / 2)) > 0)[
        set nextXCoor nextXCoor + 1
    ]
    if(drone.beta * (targetx - xcor) + drone.apha_rr * (sigma - (1 / 2)) < 0)[
        set nextXCoor nextXCoor - 1
    ]
    if(drone.beta * (targety - ycor) + drone.apha_rr * (sigma - (1 / 2)) > 0)[
        set nextYCoor nextYCoor + 1
    ]
    if(drone.beta * (targety - ycor) + drone.apha_rr * (sigma - (1 / 2)) < 0)[
        set nextYCoor nextYCoor - 1
    ]
    let nextPatch patch nextXCoor nextYCoor
    if(not (nextXCoor = xcor and nextYCoor = ycor))[
        face nextPatch
        obstacleAvoidanceRandom
    ]
end

```

- Viene fatto ruotare il robot verso la patch con coordinate calcolate mediante:

$$\begin{cases} x_k^{t+1} = x_k^t + 1 & \text{se } \beta_0 e^{-\gamma r_{kz}^2} (x_z - x_k^t) + \alpha \left( \sigma - \frac{1}{2} \right) > 0 \\ x_k^{t+1} = x_k^t - 1 & \text{se } \beta_0 e^{-\gamma r_{kz}^2} (x_z - x_k^t) + \alpha \left( \sigma - \frac{1}{2} \right) < 0 \\ x_k^{t+1} = x_k^t & \text{se } \beta_0 e^{-\gamma r_{kz}^2} (x_z - x_k^t) + \alpha \left( \sigma - \frac{1}{2} \right) = 0 \end{cases}$$

$$\begin{cases} y_k^{t+1} = y_k^t + 1 & \text{se } \beta_0 e^{-\gamma r_{kz}^2} (y_z - y_k^t) + \alpha \left( \sigma - \frac{1}{2} \right) > 0 \\ y_k^{t+1} = y_k^t - 1 & \text{se } \beta_0 e^{-\gamma r_{kz}^2} (y_z - y_k^t) + \alpha \left( \sigma - \frac{1}{2} \right) < 0 \\ y_k^{t+1} = y_k^t & \text{se } \beta_0 e^{-\gamma r_{kz}^2} (y_z - y_k^t) + \alpha \left( \sigma - \frac{1}{2} \right) = 0 \end{cases}$$

- Il robot si sposta mediante la funzione *obstacleAvoidanceRandom*

## STRATEGIA PSO

### ALGORITHM 3 (PSO\_RR)

```
to pso_rr
  updateState
  ask drones with [ drone.state = "RECRUITED" ] [
    let index 0
    if (length drone.targetPatchList > 1) [
      set index computeProbabilityPso drone.targetPatchList
    ]
    set drone.targetSelected item index drone.targetPatchList
    computePositionPso drone.targetSelected
  ]
end
```

- Le variabili dei robot vengono aggiornate mediante *updateState*
- Per ogni drone con stato “RECRUITED”:
  - Se è nel raggio di reclutamenti di almeno un robot coordinatore:
    - Viene calcolato il coordinatore più vicino
  - Viene calcolata la prossima posizione del robot in relazione al coordinatore scelto

### COMPUTE PROBABILITY PSO

```
to-report computeProbabilityPso [lst]
  let dist world-width
  let index 0
  let tmpIndex 0
  let xcord 0
  let ycord 0
  let finded false
  let tmpdist 0
  foreach lst [
    set xcord item 0 ?
    set ycord item 1 ?
    set tmpdist sqrt((pxcor - xcord) ^ 2 + (pycor - ycord) ^ 2)
    if( tmpdist < dist)[
      set index tmpIndex
      set dist tmpdist
    ]
    set tmpIndex tmpIndex + 1
  ]
  report index
end
```

Restituisce semplicemente il coordinatore più vicino.

---

**COMPUTE POSITION PSO**

```

to computePositionPso [targetCoordinates]
  let targetX (item 0 targetCoordinates)
  let targetY (item 1 targetCoordinates)
  let newDestinationCellX 0
  let newDestinationCellY 0
  set drone.virtualVx ( (drone.inertialWeight * drone.virtualVx)
    + (drone.r_rr * accelerationCoefficient * (targetX - xcor)) )
  set drone.virtualVy ( (drone.inertialWeight * drone.virtualVy)
    + (drone.r_rr * accelerationCoefficient * (targetY - ycor)) )
  if drone.virtualVx = 0 [
    set newDestinationCellX xcor
  ]
  if drone.virtualVx > 0 [
    set newDestinationCellX xcor + 1
  ]
  if drone.virtualVx < 0 [
    set newDestinationCellX xcor - 1
  ]
  if drone.virtualVy = 0 [
    set newDestinationCellY ycor
  ]
  if drone.virtualVy > 0 [
    set newDestinationCellY ycor + 1
  ]
  if drone.virtualVy < 0 [
    set newDestinationCellY ycor - 1
  ]
  if(not (patch newDestinationCellX newDestinationCellY = nobody))[ 
    face patch newDestinationCellX newDestinationCellY
  ]
  if(not (drone.virtualVx = 0 and drone.virtualVy = 0))[ 
    obstacleAvoidanceRandom
  ]
end

```

- Viene calcolata la velocità "virtuale" del robot mediante la formula:

$$\begin{cases} v_{xk}^{t+1} = \omega v_{xk}^t + r \cdot c(x_z - x_k^t) \\ v_{yk}^{t+1} = \omega v_{yk}^t + r \cdot c(y_z - y_k^t) \end{cases}$$

- Viene calcolata la posizione successiva del robot mediante la formula:

$$\begin{cases} x_k^{t+1} = x_k^t + 1 & se \quad v_{xk}^{t+1} > 0 \\ x_k^{t+1} = x_k^t - 1 & se \quad v_{xk}^{t+1} < 0 \\ x_k^{t+1} = x_k^t & se \quad v_{xk}^{t+1} = 0 \end{cases}$$

$$\begin{cases} y_k^{t+1} = y_k^t + 1 & se \quad v_{yk}^{t+1} > 0 \\ y_k^{t+1} = y_k^t - 1 & se \quad v_{yk}^{t+1} < 0 \\ y_k^{t+1} = y_k^t & se \quad v_{yk}^{t+1} = 0 \end{cases}$$

- Se la patch successiva non è uguale alla precedente:

- Il robot ruota verso la patch successiva
- Il robot si sposta mediante la funzione *obstacleAvoidanceRandom*

**STRATEGIA ABC**


---

**ALGORITHM 4 (ABC\_RR)**

```

to abc_rr
  updateState
  ask drones with [drone.state = "RECRUITED"] [
    let index 0
    let TargetOutOfRange false
  
```

```

    if(not (drone.targetSelected = [])){
        let x (item 0 drone.targetSelected - xcor)
        let y (item 1 drone.targetSelected - ycor)
        if (distance (patch-at x y) > drone.wirelessRange)[
            set TargetOutOfRange true
        ]
    }

    if(drone.targetSelected = [] or TargetOutOfRange = true)[
        if (length drone.targetPatchList > 1) [
            set index computeProbability drone.targetPatchList
        ]
        set drone.targetSelected item index drone.targetPatchList
    ]

    computePosition drone.targetSelected
]

end

```

1. Le variabili dei droni vengono aggiornate mediante *updateState*
2. Per ogni drone con stato “RECRUITED”:
  - 2.1. Se ha già selezionato il proprio coordinatore ma è uscito dal suo raggio di reclutamento wireless:
    - 2.1.1. Il drone ricalcolerà il proprio coordinatore al passo 2.2 come indicato da (\*4)
  - 2.2. Se il coordinatore non è ancora stato scelto o il drone è uscito dal wireless range del precedente:
    - 2.2.1. Se esiste più di un coordinatore che ha inviato richiesta di reclutamento al drone:
      - 2.2.1.1. Viene scelto un nuovo coordinatore per quel drone attraverso la “spinning wheel”
      - 2.2.2. Viene assegnato il coordinatore al drone
  - 2.3. Viene calcolata la posizione del drone al passo successivo mediante *computePosition*

## COMPUTE PROBABILITY

```

to-report computeProbability [lst]
let dist 0
let mu 0
let sumMu 0
let probability 0
let highest 0
let index 0
let retIndex 0
let xcord 0
let ycord 0
let probabilityList []
let finded false
let chosen 0
foreach lst [
    set xcord item 0 ?
    set ycord item 1 ?
    set dist sqrt((pxcor - xcord) ^ 2 + (pycor - ycord) ^ 2)
    set mu dist ^ -1
    set sumMu sumMu + mu
]
foreach lst [
    set xcord item 0 ?
    set ycord item 1 ?
    set dist sqrt((pxcor - xcord) ^ 2 + (pycor - ycord) ^ 2)
    set mu dist ^ -1
    set probability mu / sumMu
    set probabilityList lput probability probabilityList
]
let p 0
while [ finded = false ] [
    foreach probabilityList [
        if( random-float 1 < (p + ?) )[
            if ( finded = false)[
                set finded true
                set chosen index
            ]
        ]
    ]
]

```

```

        set p p + ?
        set index ((index + 1) mod (length probabilityList))
    ]
report index
end

```

Seguendo le espressioni per il calcolo della probabilità associata ad ogni coordinatore di essere scelto, dove  $\mu_z^k$  è il reciproco della distanza del drone dal coordinatore:

$$p_z^k = \frac{\mu_z^k}{\sum_{b=1}^{RR_k} \mu_b^k}$$

1. La procedura riceve in ingresso la lista delle coordinate delle patch sottostanti ai coordinatori che hanno inviato richiesta di reclutamento al drone
2. Sommando i contributi di ogni elemento della lista viene calcolato il denominatore dell'espressione sovrastante
3. Per ogni elemento della lista:
  - 3.1. Viene calcolata la probabilità  $p$  associata al coordinatore
  - 3.2. La probabilità viene inserita in una lista delle probabilità di somma unitaria
4. Finché non è stato scelto un coordinatore:
  - 4.1. Per ogni elemento della lista delle probabilità:
    - 4.1.1. Viene casualmente estratto un numero tra 0 e 1
    - 4.1.2. Se il numero estratto ha probabilità minore del nuovo elemento della lista sommato agli elementi già estratti:
      - 4.1.2.1. Il nuovo elemento è il coordinatore scelto.
5. Viene restituito l'indice del coordinatore scelto

### COMPUTE POSITION

```

to computePosition [targetCoordinates]
  let targetx (item 0 targetCoordinates)
  let targety (item 1 targetCoordinates)
  let nextXCor xcor
  let nextYCor ycor

  let xcoeff (drone.phi_rr * (xcor - targetx))
  let ycoeff (drone.phi_rr * (ycor - targety))

  if(xcoeff > 0)[
    set nextXCor nextXCor + 1
  ]
  if(xcoeff < 0)[
    set nextXCor nextXCor - 1
  ]
  if(ycoeff > 0)[
    set nextYCor nextYCor + 1
  ]
  if(ycoeff < 0)[
    set nextYCor nextYCor - 1
  ]

  let nextPatch patch nextXcor nextYcor

  if( not (nextPatch = nobody))[  
    face nextPatch
  ]
  if(not (nextXCor = xcor and nextYCor = ycor))[  
    obstacleAvoidanceRandom
  ]
end

```

Date le espressioni per il calcolo della posizione dei droni reclutati:

$$\begin{cases} x_k^{t+1} = x_k^t + 1 & \text{se } [\phi(x_k^t - x_z) > 0] \\ x_k^{t+1} = x_k^t - 1 & \text{se } [\phi(x_k^t - x_z) < 0] \\ x_k^{t+1} = x_k^t & \text{se } [\phi(x_k^t - x_z) = 0] \end{cases}$$

$$\begin{cases} y_k^{t+1} = y_k^t + 1 & \text{se } [\phi(y_k^t - y_z) > 0] \\ y_k^{t+1} = y_k^t - 1 & \text{se } [\phi(y_k^t - y_z) < 0] \\ y_k^{t+1} = y_k^t & \text{se } [\phi(y_k^t - y_z) = 0] \end{cases}$$

1. La procedura riceve in ingresso le coordinate del drone coordinatore
2. Vengono calcolate le espressioni di  $\phi(x_k^t - x_z)$  e  $\phi(y_k^t - y_z)$
3. Vengono calcolate le coordinate del drone al passo successivo
4. Se il drone non deve rimanere fermo ( $\phi = 0$ ):
  - 4.1. Il drone si gira verso la nuova posizione
5. Se il drone non deve rimanere fermo:
  - 5.1. Il drone esegue la procedura di obstacle avoidance mediante *obstacleAvoidanceRandom*

## STRATEGIA DI RECLUTAMENTO

Quando un robot trova un target entra nello stato di *coordinatore*, in questo stato, ad ogni tick, invia una richiesta di reclutamento a tutti i robot all'interno del suo *wirelessRange*. I robot esploratori inseriscono le coordinate dei robot coordinatori da cui hanno ricevuto richieste in una lista, che verrà utilizzata per scegliere il miglior coordinatore in base alle tre tecniche di robot recruitment descritte precedentemente.

---

### SEND REQUEST

```
to sendRequest
  let xcord pxcor
  let ycord pycor
  let dist 0
  ask drones with [(drone.state = "EXPLORER") or (drone.state = "RECRUITED")]
    [set dist sqrt((pxcor - xcord) ^ 2 + (pycor - ycord) ^ 2)
     if (dist < drone.wirelessRange)
       [set drone.targetPatchList lput (list xcord ycord) drone.targetPatchList
     ]
   ]
end
```

1. Per ogni robot con stato “EXPLORER” o “RECRUITED”:
  - 1.1. Viene calcolata la distanza tra esso e il robot coordinatore
  - 1.2. Se la distanza calcolata al punto precedente è minore del wireless range del coordinatore:
    - 1.2.1. Il coordinatore viene inserito nella lista dei coordinatori del robot

---

### UPDATE STATE

```
to updateState
  let tmpx 0
  let tmpy 0
  let trgtX 0
  let trgtY 0
  let coorDrone nobody
  let dist 0
  let rotationAngle 0
  ask drones with [drone.state = "RECRUITED"]
    [set tmpx xcor
     set tmpy ycor
     set coorDrone drone.coordinator
     let dr other drones
     ask dr[
       set dist distancexy tmpx tmpy
       if (dist < drone.executionRange) and (drone.state = "COORDINATOR") and (coorDrone = nobody)
         [set coorDrone self
          set trgtX xcor
          set trgtY ycor
          set drone.nRecruited drone.nRecruited + 1
        ]
      ]
    ]
end
```

```

        if(drone.nRecruited >= sufficientRobots - 1) [
            set drone.nRecruited 0
            set drone.state "EXECUTION"
            set color violet
        ]
    ]

    if not (coorDrone = nobody) [
        set drone.state "WAITING"
        set color brown ;; per debug
        set drone.coordinator coorDrone
        face drone.coordinator
    ]
    set coorDrone nobody
]

```

1. Per ogni drone con stato uguale a “RECRUITED”:

1.1. Per ogni drone che non è stato reclutato:

1.1.1. Viene calcolata la distanza tra il drone e quello reclutato

1.1.2. Se la distanza è inferiore al raggio di esecuzione e il drone è coordinatore:

1.1.2.1. Il drone reclutato entra nello stato di “WAITING” al punto 1.2.1

1.1.2.2. Viene aggiornato il numero di droni in attesa presso il coordinatore

1.1.2.3. Se il numero di droni in attesa è sufficiente:

1.1.2.3.1. Il drone coordinatore entra nello stato di “EXECUTION”

1.1.2.3.2. Il drone coordinatore assume la colorazione viola

1.2. Se il drone è nel raggio di esecuzione di un coordinatore:

1.2.1. Il drone entra nello stato di WAITING

1.2.2. Il drone assume la colorazione marrone

1.2.3. Al drone viene associato il coordinatore

1.2.4. Il drone si gira verso il coordinatore

```

ask drones with [ drone.state = "COORDINATOR" ] [
    sendRequest
]
ask drones with [ (drone.state = "EXPLORER") or (drone.state = "RECRUITED") ] [
    ifelse (length drone.targetPatchList > 0) [
        if(not (drone.state = "RECRUITED")){
            set drone.phi_rr (1 - random 3)
            set drone.apha_rr random-float 1
            set drone.r_rr random-float 1
            set drone.virtualVx cos heading
            set drone.virtualVy sin heading
            set drone.state "RECRUITED"
            set color blue ;; per debug
        }
    ][
        set drone.state "EXPLORER"
        set color green ;; per debug
    ]
]
ask drones with [ drone.state = "RECRUITED" and drone.phi_rr = 0 ][
    set drone.IdleTicksLeft drone.IdleTicksLeft - 1
    if(drone.IdleTicksLeft <= 0)[
        set drone.IdleTicksLeft drone.maxIdleTicks
        set drone.phi_rr (1 - random 3)
    ]
]
ask drones with [drone.state = "COORDINATOR"][
    ask patches in-radius (drone.executionRange)[
        if(target = true and not(status = "found")){
            set status "found"
        }
    ]
]
end

```

2. Per ogni drone con stato “COORDINATOR”:

a. Viene inviata una richiesta di reclutamento ai droni circostanti mediante *sendRequest*

3. Per ogni drone con stato “EXPLORER” o “RECRUITED”:
  - a. Se almeno un coordinatore ha inviato una richiesta di reclutamento ad esso e non è ancora stato reclutato:
    - 3.a.1. vengono calcolati i coefficienti di randomizzazione  $\phi, \alpha, r$  seguendo la metodologia (\*3)
    - 3.a.2. Dato che la velocità dei robot è unitaria, le componenti verticali e orizzontali sono calcolate come il seno e il coseno del verso in cui è orientato
    - 3.a.3. Viene deselezionato un eventuale coordinatore selezionato precedentemente
    - 3.a.4. Il drone entra nello stato “RECRUITED”
    - 3.a.5. Il drone assume la colorazione blu
  - b. Altrimenti:
    - 3.b.1. Il drone entra nello stato “EXPLORER”
    - 3.b.2. Il drone assume la colorazione verde
3. Per ogni drone con stato “RECRUITED” e  $\phi = 0$ :
  - 3.1. Il contatore dei tick di attesa rimasti diminuisce di uno
  - 3.2. Se il numero di tick di attesa rimasti è nullo:
    - 3.2.1. Il numero di tick di attesta viene ripristinato al valore di default
    - 3.2.2. Il valore di  $\phi$  viene ricalcolato come indicato in (3\*)
4. Per ogni drone con stato “COORDINATOR”:
  - 4.1. Per ogni patch di tipo target non ancora trovata nel raggio di esecuzione del drone:
    - 4.1.1. Il target viene settato come trovato

## STRATEGIA DI ESECUZIONE

La strategia di esecuzione prevede che, una volta raggiunto un numero sufficiente di droni reclutati (*sufficientRobots*), il coordinatore e i droni in stato “WAITING” presso di esso entrino nello stato di esecuzione.

L'esecuzione rappresenta l'elaborazione che i droni devono effettuare sul target ed è implementata tramite un semplice contatore di tick di valore *drone.ExecutionTicksLeft*. Una volta esaurito il contatore, i droni escono dallo stato “EXECUTION” ed il target viene considerato elaborato.

---

## EXECUTION ALGORITHM

```
to executionAlgorithm
  ask drones with [ drone.state = "EXECUTION" ] [
    ask drones in-radius (drone.executionRange)
      with[drone.state = "WAITING" and drone.coordinator = myself ][
        set drone.state "EXECUTION"
        set color violet
      ]
      set drone.ExecutionTicksLeft drone.ExecutionTicksLeft - 1
      if(drone.ExecutionTicksLeft <= 0)[
        endExecution
      ]
    ]
    checkTargetsComplete
  end
```

1. Per ogni drone con stato “EXECUTION”
  - 1.1. Ogni robot, con stato “WAITING”, nel raggio di esecuzione e avente come coordinatore il drone in esecuzione:
    - 1.1.1. Entra nello stato di esecuzione
    - 1.1.2. Assume una colorazione viola
  - 1.2. Il contatore di tick rimasti per l'esecuzione viene decrementato di uno
  - 1.3. Se il contatore è nullo:
    - 1.3.1. L'esecuzione termina
2. Vengono controllati i target la cui esecuzione è stata completata mediante *checkTargetsComplete*

---

## EXECUTION COMPLETE

```

to endExecution
  ask drones in-radius (drone.executionRange)
    with [drone.state = "EXECUTION" and drone.coordinator = myself] [
      set drone.state "EXPLORER"
      set drone.targetPatchList []
      set drone.ExecutionTicksLeft executionTime
      set drone.coordinator nobody
    ]
  ask patches in-radius (drone.executionRange) with[target = true] [
    set execution.complete true
  ]
  set drone.state "EXPLORER"
  set drone.targetPatchList []
  set drone.ExecutionTicksLeft executionTime
  set drone.coordinator nobody
end

```

1. Per ogni drone nel raggio di esecuzione, con stato uguale a “EXECUTION” e come coordinatore questo drone:
  - 1.1. Il drone entra nello stato “EXPLORER”
  - 1.2. Viene svuotata la lista dei coordinatori
  - 1.3. Viene resettato il timer di esecuzione
  - 1.4. Viene de assegnato il coordinatore
2. Per ogni target nel raggio di esecuzione del drone:
  - 2.1. L'esecuzione viene settata come completata
3. Il drone entra nello stato “EXPLORER”
4. Viene svuotata la lista dei coordinatori
5. Viene resettato il timer di esecuzione del drone
6. Viene de assegnato il coordinatore

---

## CHECK TARGETS COMPLETE

```

to checkTargetsComplete
  let nTarget count patches with [target = true ]
  let nExecuted 0
  ask patches [
    if(target = true and execution.complete = true)[
      set pcolor white
      set nExecuted nExecuted + 1
    ]
  ]
  ifelse(nTarget = 0)[
    set percentageTargetsExecuted 0
  ][
    set percentageTargetsExecuted (nExecuted / nTarget) * 100
  ]
end

```

1. Per ogni patch:
  - 1.1. Se è un target e l'esecuzione è completata:
    - 1.1.1. La patch assume la colorazione bianca
2. La percentuale di target la cui esecuzione è completa viene aggiornata

## SCIADRO CON ELABORAZIONE DEI TARGET

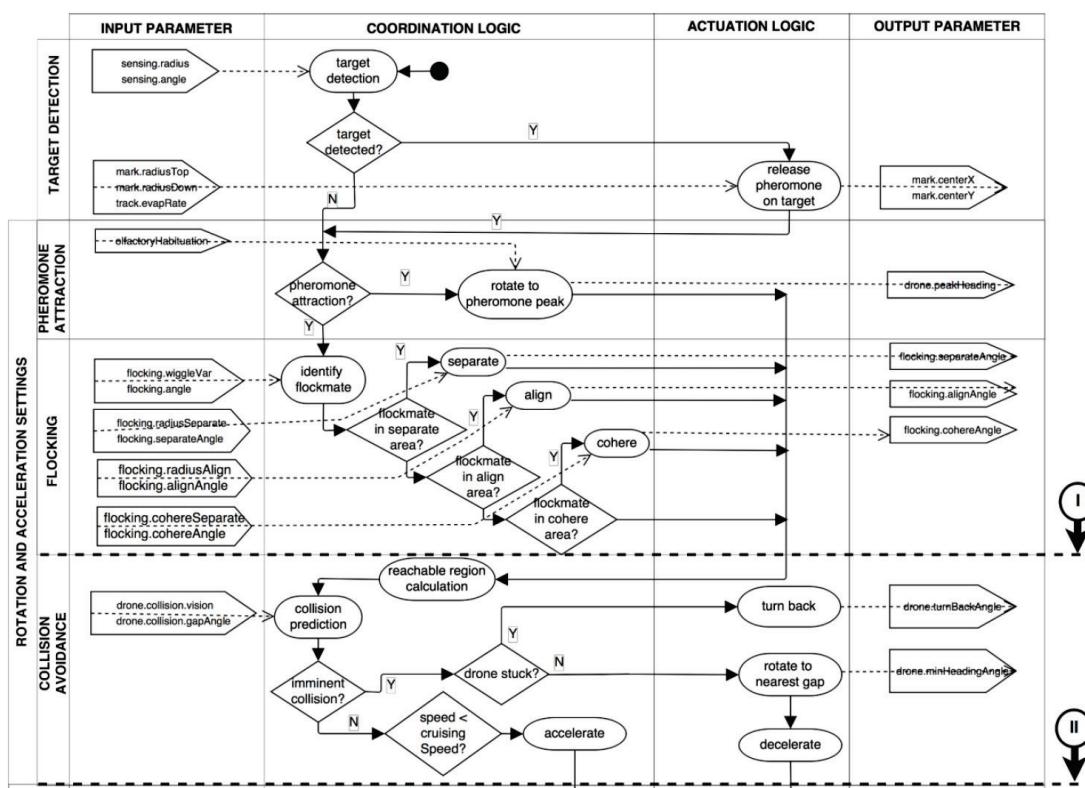
### MODELLO

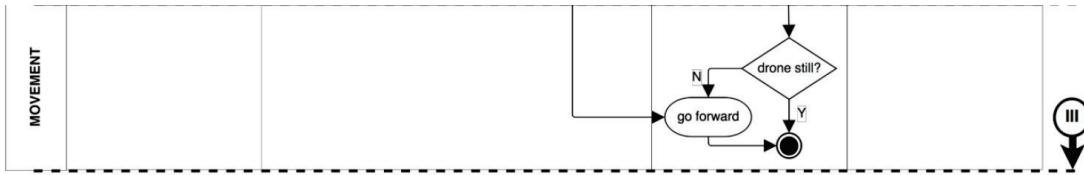
#### PROGETTO SCIADRO

L'obiettivo del modello *Sciadro*, analogamente a quello tratto da “*Comparison of bio-inspired algorithms applied to the coordination of mobile robots considering the energy consumption*”, è di esplorare un territorio nel minor tempo possibile alla ricerca di target.

In *Sciadro* vengono utilizzati diversi meccanismi di auto coordinamento, che intervengono con diversi livelli di priorità:

- La priorità maggiore è riservata al meccanismo di **target detection**: se un target entra nel *sensing range* del drone, esso emette un *feromone attrattivo*, l'idea è che possano essere presenti più target ravvicinati in *cluster*, il feromone perciò attirerà altri droni nelle zone limitrofe.
- Il secondo livello di priorità è dato dal meccanismo di **collision avoidance**: i droni cercano di non sovrapporsi mediante la prenotazione della prossima patch in cui si muoveranno (*overlap avoidance*) e di non collidere con gli ostacoli (*obstacle avoidance*).
- Il terzo livello di priorità è dato dal meccanismo di **pheromone attraction**: se viene percepita una qualsiasi quantità di feromone, i droni tenderanno a dirigersi verso nel punto in cui la concentrazione di feromone è maggiore. È importante far notare che esiste un meccanismo di **olfactory habituation** che, dopo un certo numero di tick, rende il drone insensibile al feromone fintanto che ne continua a percepire.
- Il quarto livello di priorità è dato dal meccanismo di **flocking**: i droni tendono a rimanere uniti in stormi seguendo le regole di *separate*, *align*, *cohere*. Se due droni sono troppo vicini si separano, i droni tendono ad allinearsi ai loro compagni di stormo, se due droni entrano in un raggio di coesione si avvicinano.
- Il livello di priorità più basso è dato dal movimento libero del drone.



Figura 4: Schema di funzionamento di *Sciadro*

## MECCANISMO DI ELABORAZIONE DEI TARGET

La versione di *Sciadro* antecedente alle modifiche non prevedeva la possibilità di *elaborare* un target: l'elaborazione equivale ad un qualsiasi tipo di task che più droni debbano contemporaneamente eseguire su uno stesso target, un esempio può essere l'analisi simultanea con più strumenti di misurazione di uno stesso obiettivo.

La nuova versione di *Sciadro* prevede la fusione di un meccanismo a stati, simile a quello di *“Comparison of bio-inspired algorithms applied to the coordination of mobile robots considering the energy consumption”*, con un innovativo utilizzo del feromone attrattivo. Sono possibili tre stati, con priorità superiore ai normali meccanismi di esplorazione di *Sciadro*:

- **EXPLORER:** il drone esplora il territorio come nella vecchia versione di *Sciadro*.
- **WAITING :** i droni, se trovano un target nel loro raggio di sensing, entrano nello stato di *waiting*: essi si fermano in prossimità del target causando l'emissione di feromone attrattivo nella zona. Il feromone attira altri droni verso il target in maniera analoga al reclutamento degli algoritmi bio-inspirati, è importante settare un valore di *olfactory habituation* abbastanza elevato. I droni nello stato di *waiting* rilasciano tutte le patch, prenotate a causa del meccanismo di *overlap avoidance*, permettendo ad altri droni di avvicinarsi al target. Si presuppone che una quantità non elevata di droni possa stazionare nei pressi di una stessa patch senza rischio di collisioni, volando per esempio a quote differenti. Ogni drone nello stato di *waiting* emette feromone fintanto che permane in tale stato, perciò la quantità di feromone presente nell'intorno di un target è proporzionale al numero di droni in attesa presso di esso e al tempo per cui sono rimasti in attesa.
- **EXECUTION:** i droni, in prossimità di uno stesso target, permangono nello stato di *waiting*, fino al raggiungimento di un numero sufficiente di droni in attesa, superato il quale entrano nello stato di *execution*. In questo stato rimangono fermi per un numero prefissato di tick (ad emulare un eventuale elaborazione) prima di ritornare nello stato di esploratori e segnalare l'avvenuta elaborazione del target. Nello stato di esecuzione i droni non producono più feromone attrattivo, per evitare di attirare inutilmente altri droni.

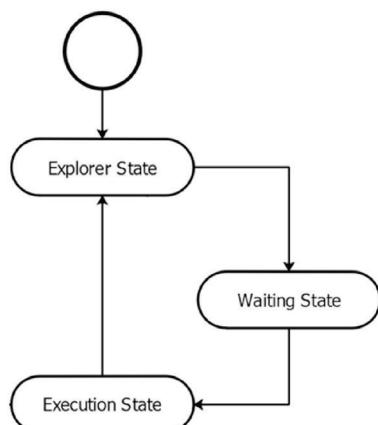


Figura 5: Possibili stati

## IMPLEMENTAZIONE

Di seguito sono riportate esclusivamente le procedure che hanno subito modifiche dalla vecchia versione di *Sciadro*.

### DRONE LOGIC

```
to droneLogic
  ask drones with[drone.state = "EXPLORER"] [
    checkDronesOverlap
    releasePheromoneNearestTgt
    let phAttraction pheromoneAttraction
    if (phAttraction = false) [
      let flocking flockingMechanism
    ]
  ]
  patchesScheduling
  ask drones with[drone.state = "EXPLORER"] [
    let noObstacle? obstacleAvoidance
    if else (noObstacle? = true) [
      accelerate
    ]
    [
      decelerate
    ]
    fd drone.speed
  ]
  ask drones with [drone.state = "EXECUTION"] [
    waitExecution
  ]
  ask drones with [drone.state = "WAITING"] [
    releasePheromoneNearestTgt
  ]
end
```

- I droni con stato uguale a “EXPLORER” si comportano allo stesso modo che nella precedente versione di *Sciadro*
- I droni con stato uguale a “EXECUTION” aspettano l’elaborazione del target mediante *waitExecution*
- I droni con stato “WAITING” producono feromone attrattivo mediante *releasePheromoneNearestTgt*

### PATCHES SCHEDULING

```
to patchesScheduling
  ask drones [
    unlockPatches
    setCurrentPatchVisited
  ]
  ask drones with[drone.state = "EXPLORER"] [
    lockFreePatches
  ]
end
```

Solo i droni esploratori possono prenotare la patch al quale si sposteranno nel tick successivo.

### RELEASE PHEROMONE IN PATCH

```
to releasePheromoneInPatch [ selectedPatch ]
  ask selectedPatch [
    releasePheromone
    set status "found"
    set pcColor yellow
    checkPossibleExecution selectedPatch
  ]
end
```

I droni esploratori in prossimità di un target rilasciano feromone e controllano la possibilità di entrare nello stato di esecuzione mediante *checkPossibleExecution*. I target vengono marcati come scoperti.

---

## CAMBIO DI STATO

```

to becomeWaiting
  set drone.state "WAITING"
  set color brown
  face min-one-of patches with [target = true] [distance myself]
end

to becomeExecutor
  set drone.state "EXECUTION"
  set color violet
end

to becomeExplorer
  set drone.state "EXPLORER"
  set color green
end

```

1. I droni che entrano nello stato di *waiting*:
  - 1.1. Assumono la colorazione marrone
  - 1.2. Si volgono verso il target più vicino
2. I droni che entrano nello stato di *execution*:
  - 2.1. Assumono la colorazione viola
3. I droni che entrano nello stato di *explorer*:
  - 3.1. Assumono la colorazione verde

---

## CHECK POSSIBLE EXECUTION

```

to checkPossibleExecution [selectedPatch]
  let dronesInExecutionRange (count drones in-radius execution.radius)
  if (dronesInExecutionRange >= sufficientRobots) [
    Execution selectedPatch
  ]
end

```

1. Si contano i droni nel raggio di esecuzione intorno ad un target
2. Se il numero di droni è sufficiente:
  - 2.1. Inizia l'elaborazione di quel target

---

## EXECUTION

```

to Execution [selectedPatch]
  ask selectedPatch [
    set status "underExecution"
    set pcolor sky
  ]
  ask n-of sufficientRobots drones in-radius execution.radius [
    becomeExecutor
    set executionTicksLeft executionTime
    set selectedTarget selectedPatch
  ]
  ask drones in-radius execution.radius with[ drone.state = "WAITING"] [
    becomeExplorer
  ]
end

```

1. Il target selezionato:
  - 1.1. Viene marcata come *underExecution*
  - 1.2. Assume la colorazione azzurra
2. Una quantità di droni pari al numero sufficiente per iniziare l'elaborazione, all'interno del raggio di esecuzione del target:
  - 2.1. Entrano nello stato di *execution*
  - 2.2. Impostano il conto alla rovescia per l'esecuzione
3. I rimanenti droni nel raggio di esecuzione del target:
  - 3.1. Ritornano esploratori

## WAIT EXECUTION

```

to waitExecution
  if (executionTicksLeft = 0) [
    ask selectedTarget [
      set status "executed"
      set pcolor 98
      set tgt.checked tgt.checked + 1
    ]
    becomeExplorer
  ]
  set executionTicksLeft executionTicksLeft - 1
end

```

1. Se l'elaborazione è giunta al termine:
  - 1.1. Il target viene marcato come *executed*
  - 1.2. Il target assume la colorazione azzurro chiaro
  - 1.3. I droni esecutori tornano nello stato di esploratori
2. Il contatore dei tick rimasti viene scalato di uno

## TESTING

### PIANO DI COLLAUDO

Gli algoritmi sviluppati per cui si è eseguito un collaudo sono:

1. Algorithm 1 + Algorithm 2 (FTS)
2. Algorithm 1 + Algorithm 3 (PSO)
3. Algorithm 1 + Algorithm 4 (ABC)
4. Sciadro con elaborazione dei target

Gli algoritmi sono stati testati principalmente in due scenari: “Dump” e “Rural Mine”:

Il primo è caratterizzato da grossi ostacoli e target distribuiti in cluster, il secondo presenta un elevato numero di piccoli ostacoli distribuiti sul territorio e una disposizione di target omogenea.

I due scenari, di tipologia statica, presentano pertanto due situazioni di utilizzo del tutto differenti.

Le schermate riportate successivamente rappresentano i seguenti istanti di esecuzione:

1. Lo stato iniziale della simulazione
2. Simulazione con più del 25% di target trovati
3. Simulazione con più del 50% di target trovati
4. Simulazione con più del 75% di target trovati
5. Simulazione con più del 95% di target trovati
6. Simulazione con più del 95% di target elaborati

I valori dei parametri per la simulazione sono stati settati in maniera euristica, in quanto il collaudo è stato eseguito per verificare esclusivamente la conformità e correttezza dell'implementazione rispetto ai modelli e non per eseguire un'ottimizzazione dei parametri. Pertanto, l'efficienza, misurata in numero di tick, degli algoritmi, non è indicativa del loro potenziale reale, infatti i parametri possono essere soggetti ad ottimizzazione, per esempio mediante l'utilizzo di algoritmi di evoluzione differenziale.

Con gli algoritmi bio-inspirati, per il testing è stata utilizzata una traccia feromonica di tipo quadrato, ritenuta più conforme al modello originale (*\*1*).

Il numero di droni da reclutare è stato impostato su tre (compreso il coordinatore) e il tempo di elaborazione del target a 5 tick.

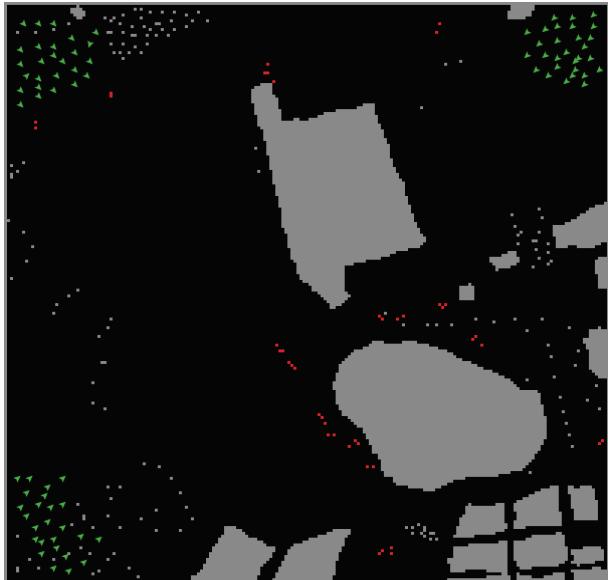
**ALGORITHM 1 + ALGORITHM 2 (FTS)****"DUMP"**

Figura 6: Algorithm 1 + FTS – Dump – Stato Iniziale

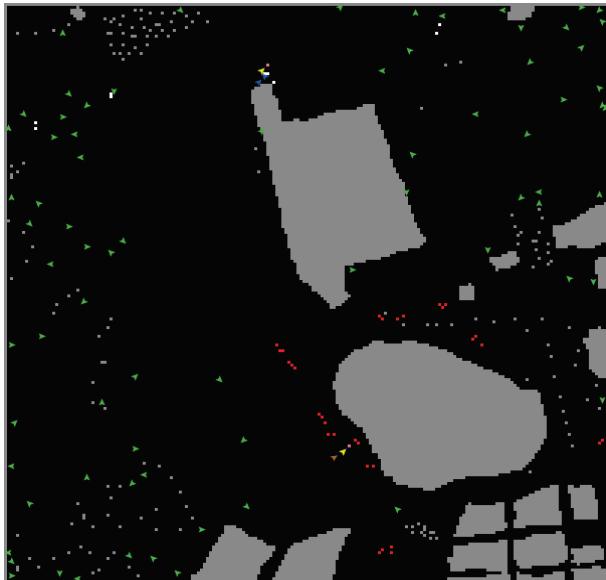


Figura 7: Algorithm 1 + FTS – Dump – 25% di target trovati

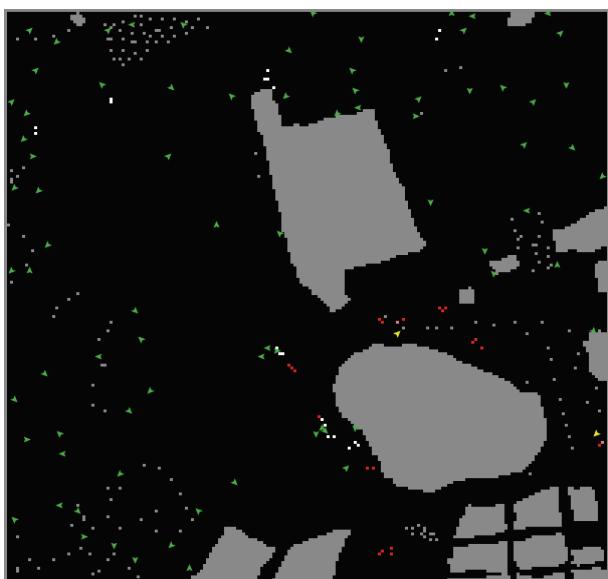


Figura 8: Algorithm 1 + FTS – Dump – 50% di target trovati

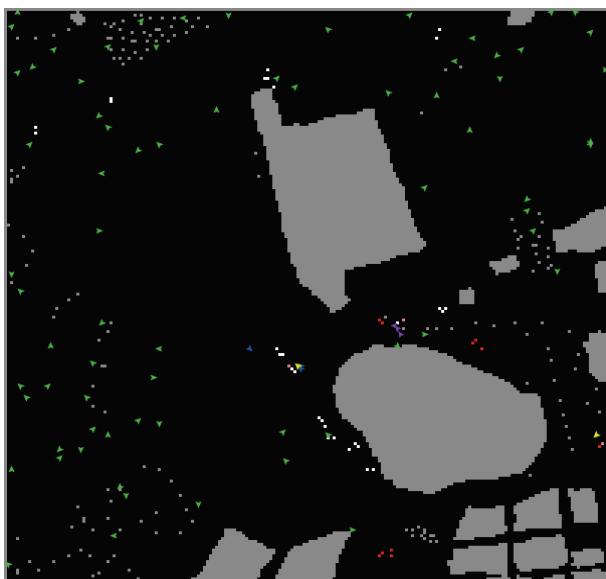


Figura 9: Algorithm 1 + FTS – Dump – 75% di target trovati

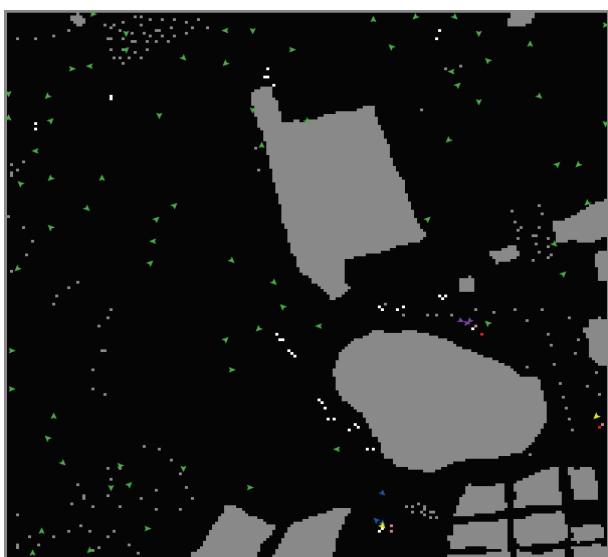


Figura 4: Algorithm 10 + FTS – Dump – 95% di target trovati

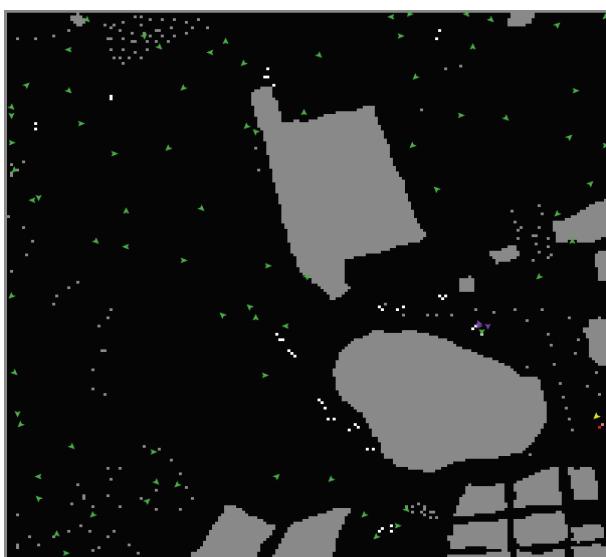


Figura 11: Algorithm 1 + FTS – Dump – 95% di target Elaborati

---

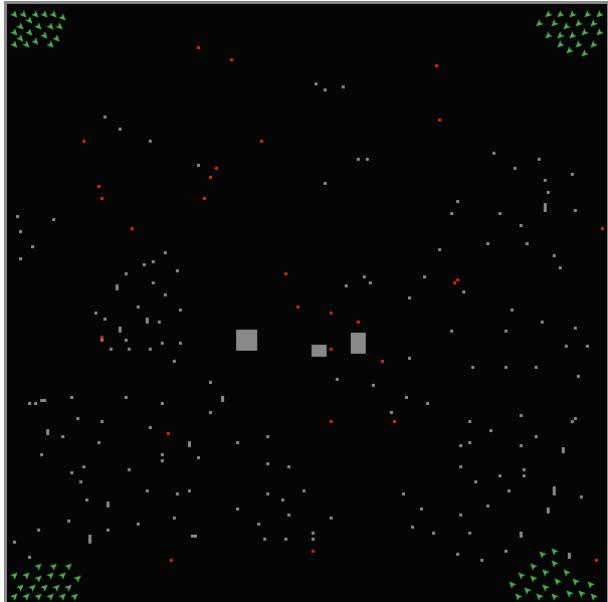
"RURAL MINE"

Figura 12: Algorithm 1 + FTS – Rural Mine – Stato Iniziale

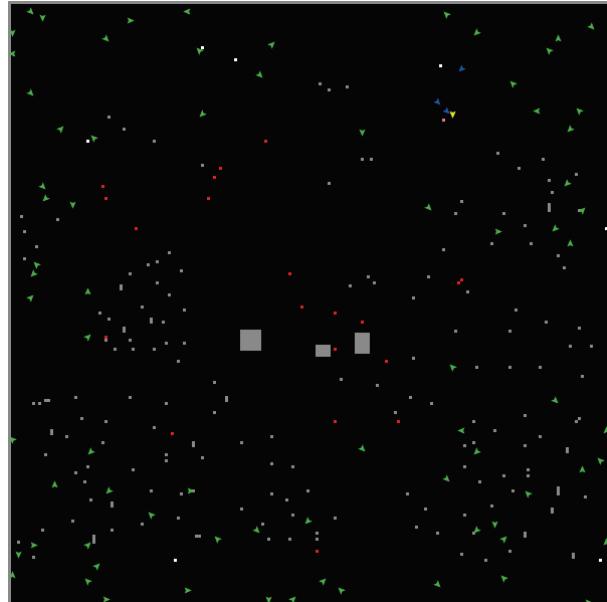


Figura 13: Algorithm 1 + FTS – Rural Mine – 25% di target trovati

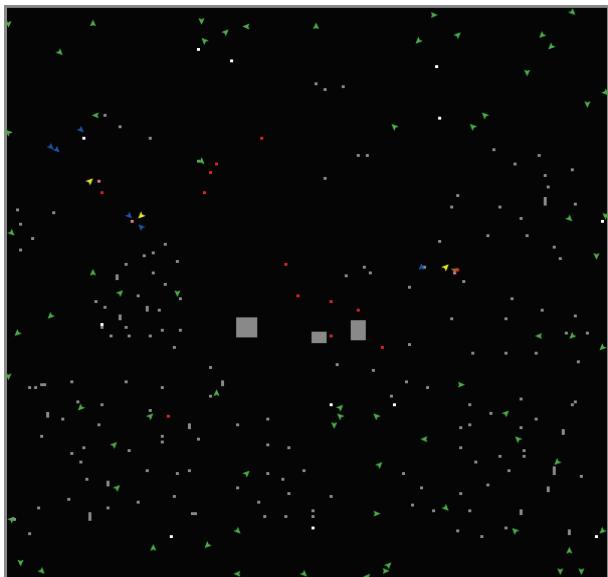


Figura 14: Algorithm 1 + FTS – Rural Mine – 50% di target trovati

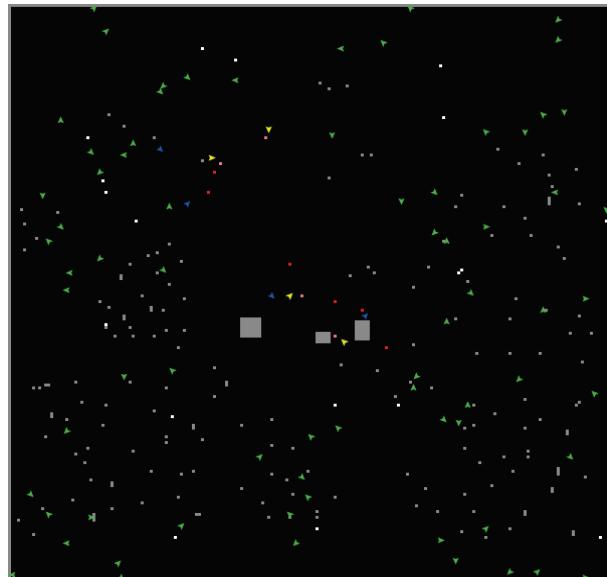


Figura 15: Algorithm 1 + FTS – Rural Mine – 75% di target trovati

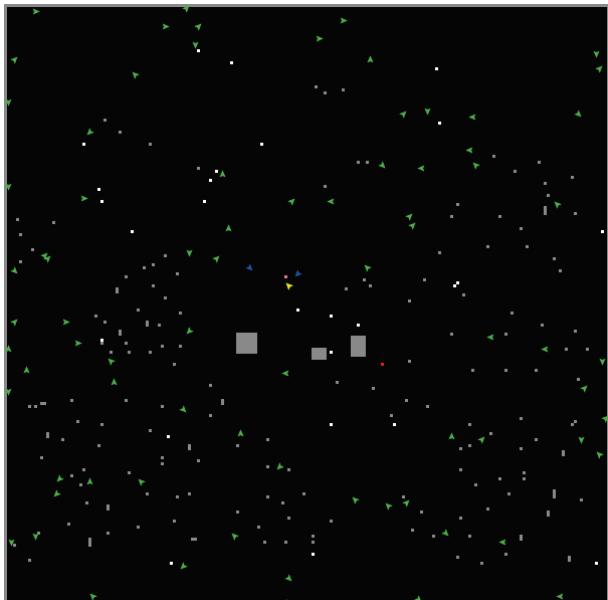


Figura 16: Algorithm 1 + FTS – Rural Mine – 95% di target trovati

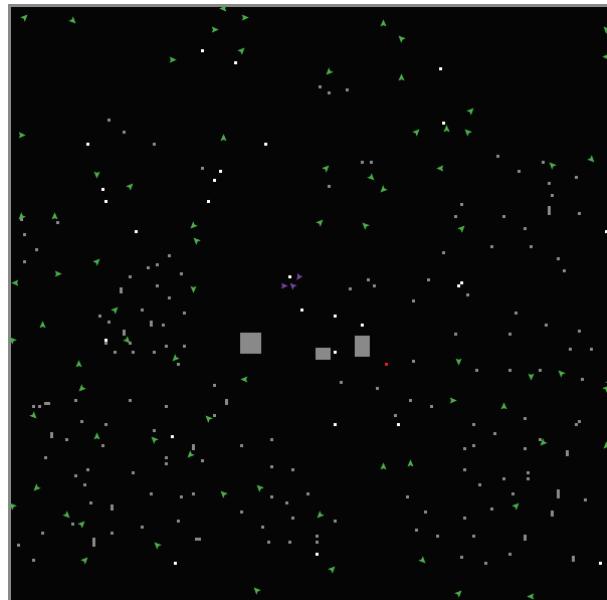


Figura 17: Algorithm 1 + FTS – Rural Mine – 95% di target Elaborati

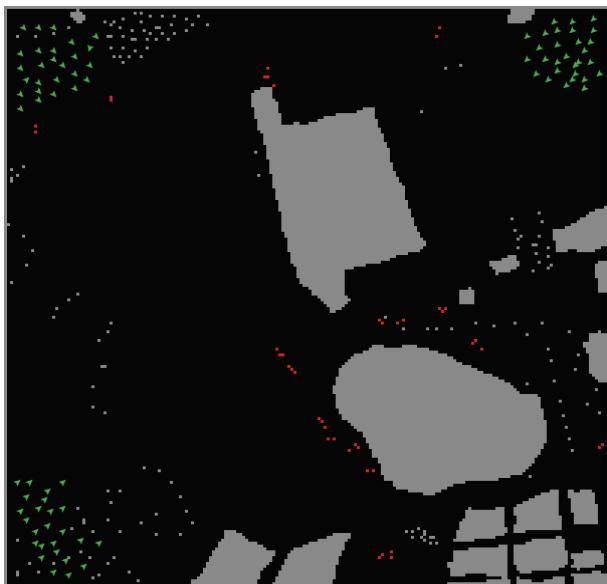
**ALGORITHM 1 + ALGORITHM 3 (PSO)****"DUMP"**

Figura 18: Algorithm 1 + PSO – Dump – Stato Iniziale

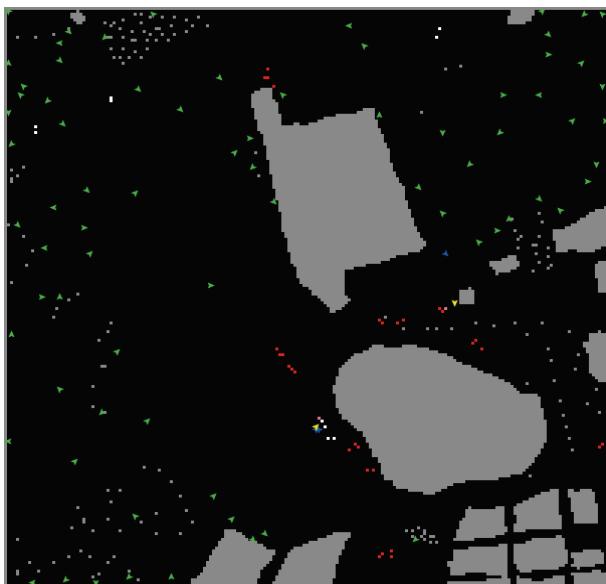


Figura 19: Algorithm 1 + PSO – Dump – 25% di target trovati

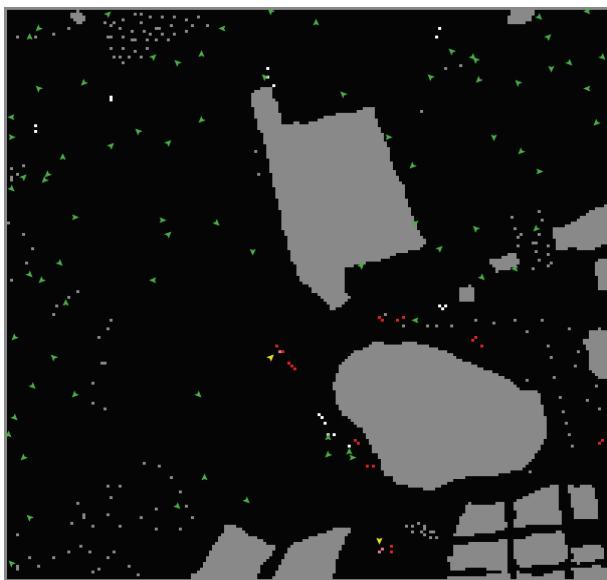


Figura 20: Algorithm 1 + PSO – Dump – 50% di target trovati

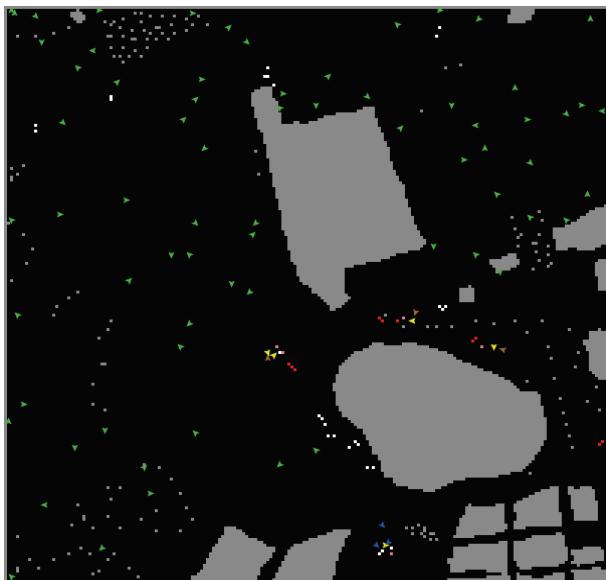


Figura 21: Algorithm 1 + PSO – Dump – 75% di target trovati

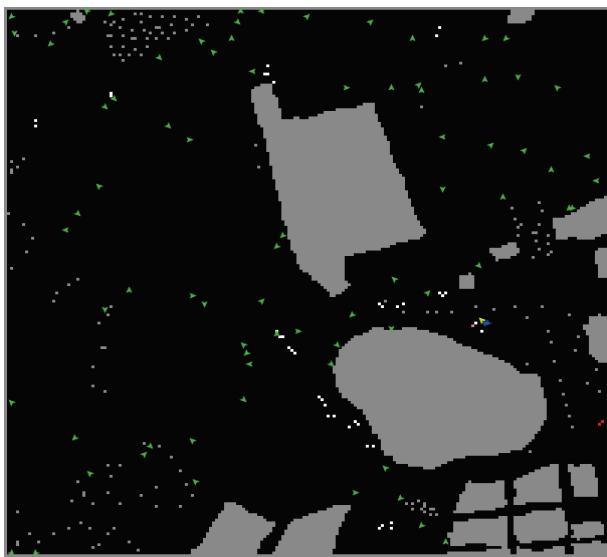


Figura 22: Algorithm 1 + PSO – Dump – 95% di target trovati

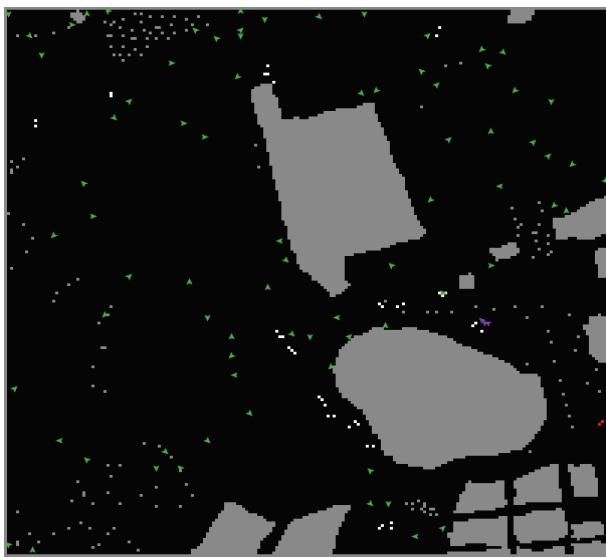


Figura 23: Algorithm 1 + PSO – Dump – 95% di target Elaborati

---

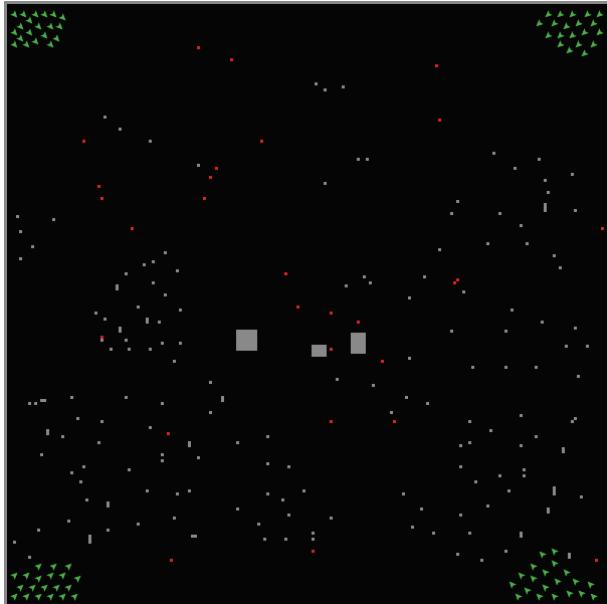
"RURAL MINE"

Figura 24: Algorithm 1 + PSO – Rural Mine – Stato Iniziale

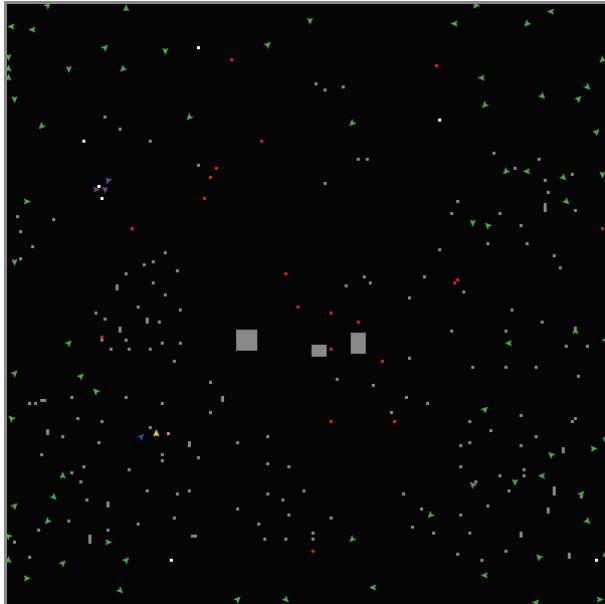


Figura 25: Algorithm 1 + PSO – Rural Mine – 25% di target trovati

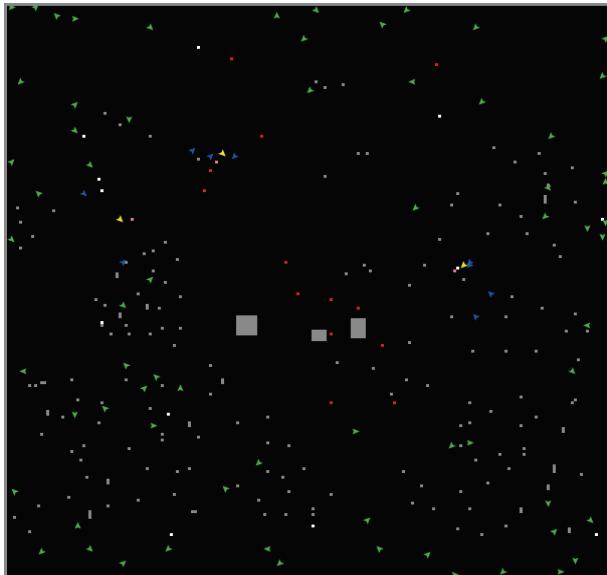


Figura 26: Algorithm 1 + PSO – Rural Mine – 50% di target trovati

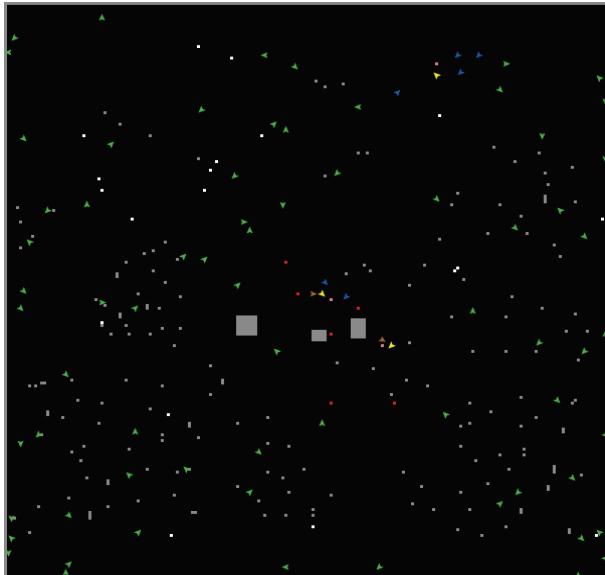


Figura 27: Algorithm 1 + PSO – Rural Mine – 75% di target trovati

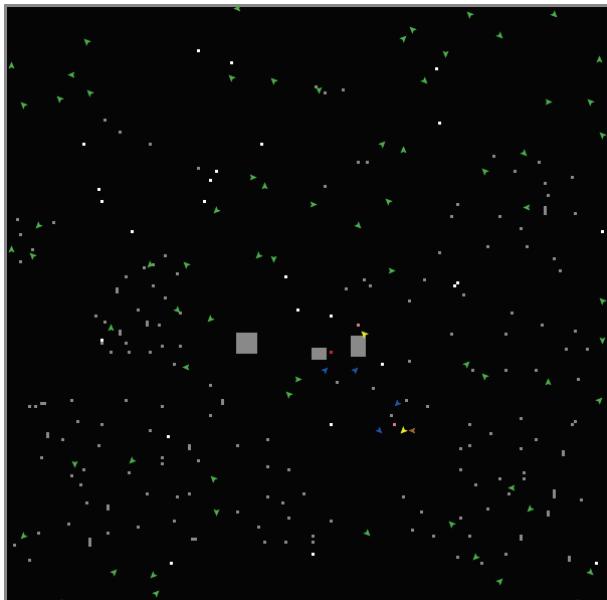


Figura 28: Algorithm 1 + PSO – Rural Mine – 95% di target trovati

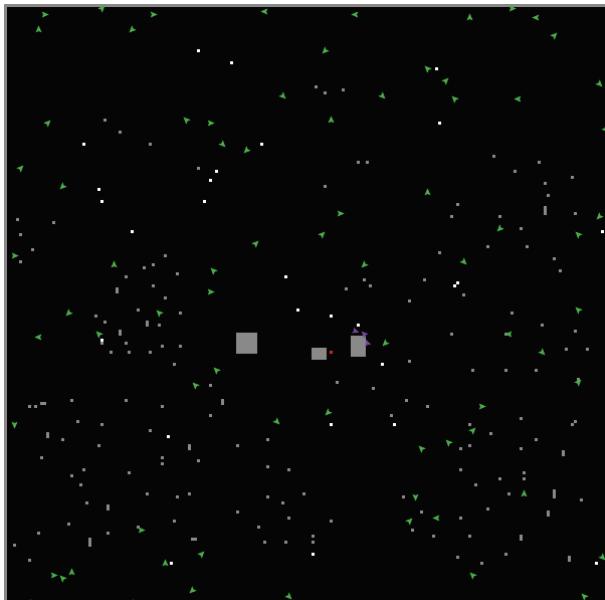


Figura 29: Algorithm 1 + PSO – Rural Mine – 95% di target Elaborati

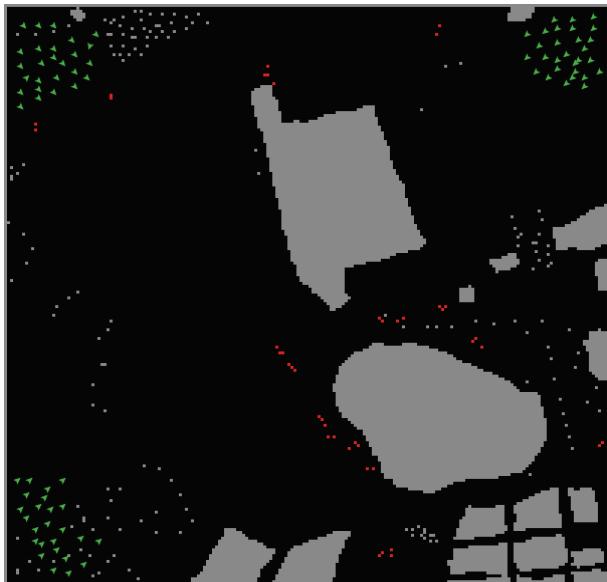
**ALGORITHM 1 + ALGORITHM 4 (ABC)****"DUMP"**

Figura 30: Algorithm 1 + ABC – Dump – Stato Iniziale

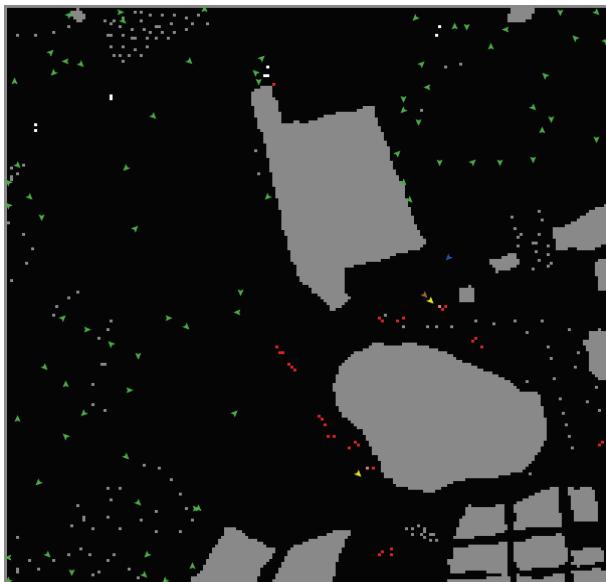


Figura 31: Algorithm 1 + ABC – Dump – 25% di target trovati

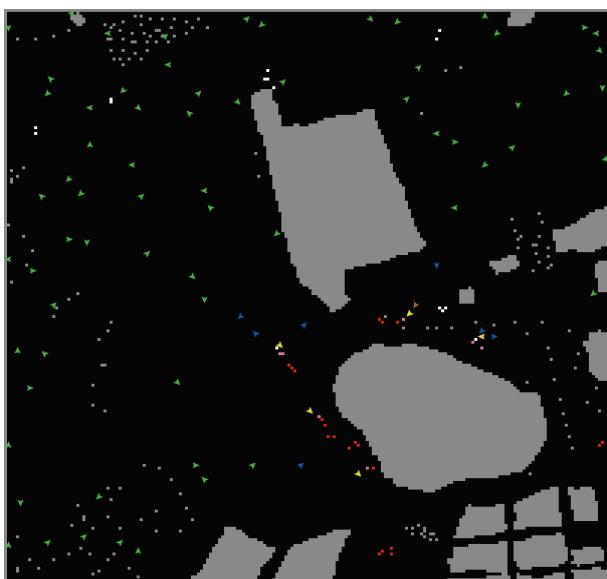


Figura 32: Algorithm 1 + ABC – Dump – 50% di target trovati

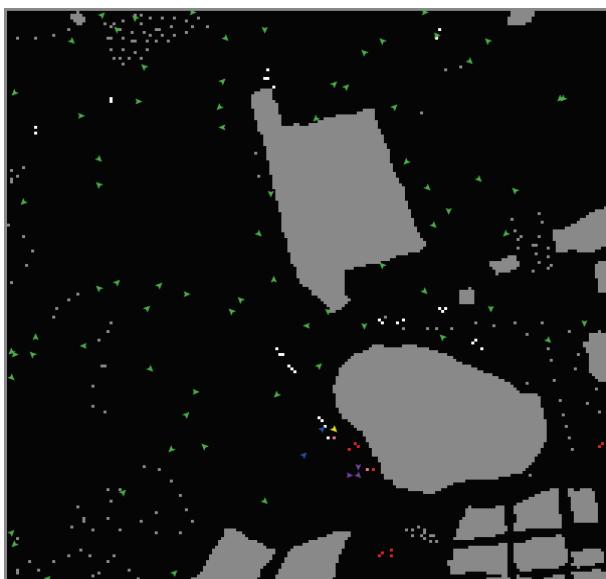


Figura 33: Algorithm 1 + ABC – Dump – 75% di target trovati

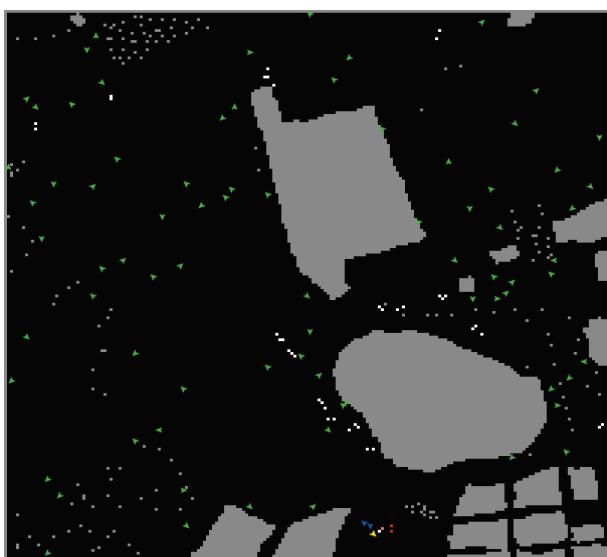


Figura 34: Algorithm 1 + ABC – Dump – 95% di target trovati

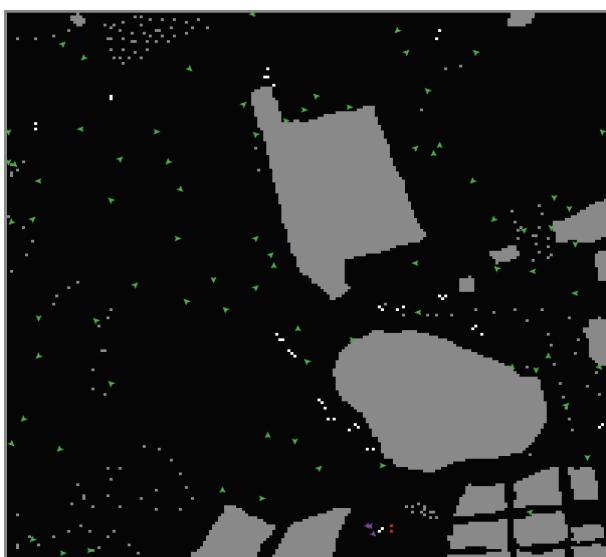


Figura 35: Algorithm 1 + ABC – Dump – 95% di target Elaborati

---

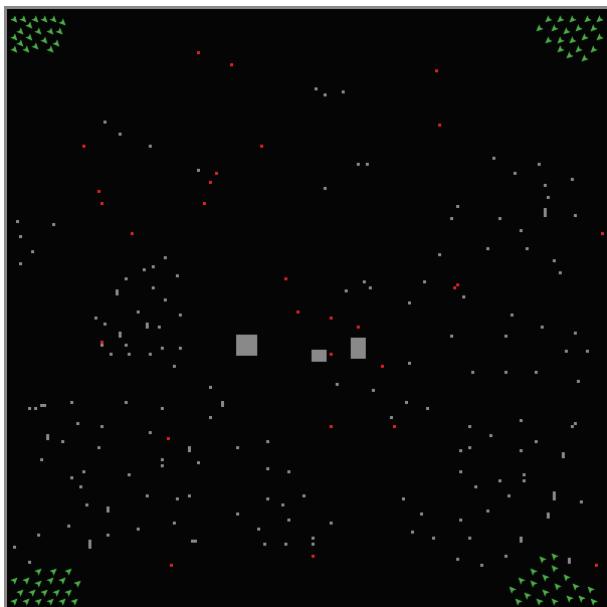
"RURAL MINE"

Figura 36: Algorithm 1 + ABC – Rural Mine – Stato Iniziale

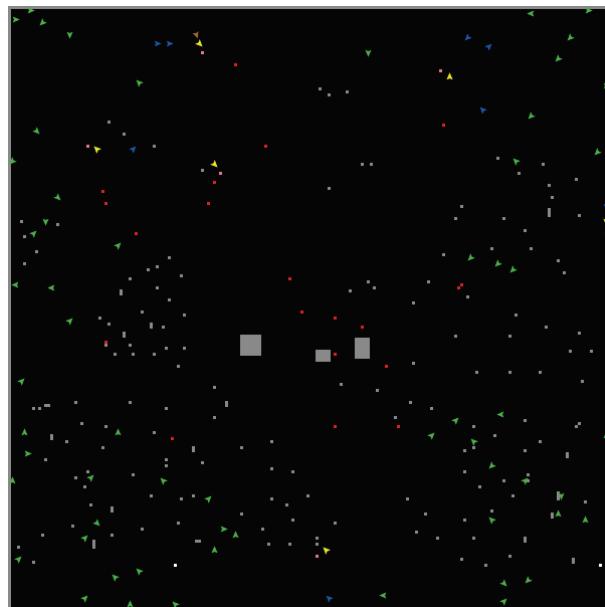


Figura 37: Algorithm 1 + ABC – Rural Mine – 25% di target trovati

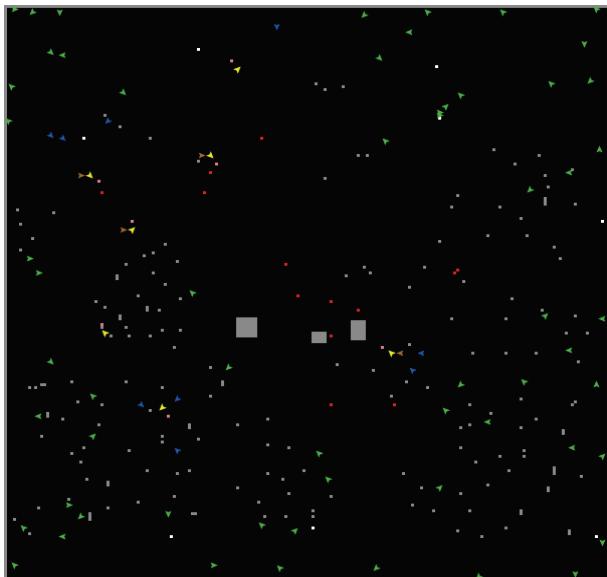


Figura 38: Algorithm 1 + ABC – Rural Mine – 50% di target trovati

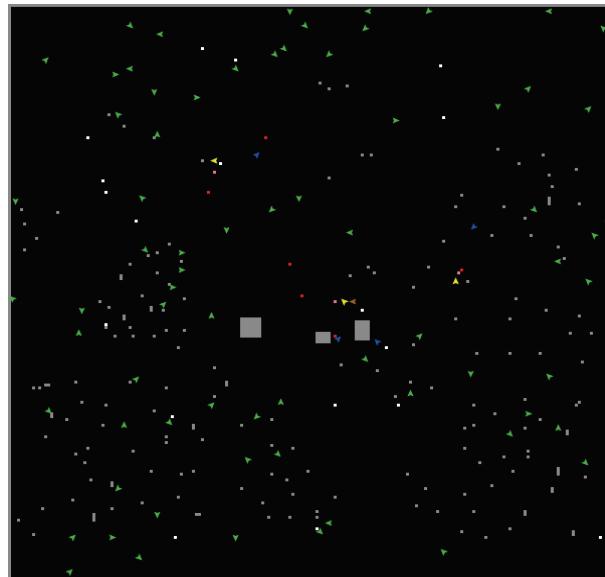


Figura 39: Algorithm 1 + ABC – Rural Mine – 75% di target trovati

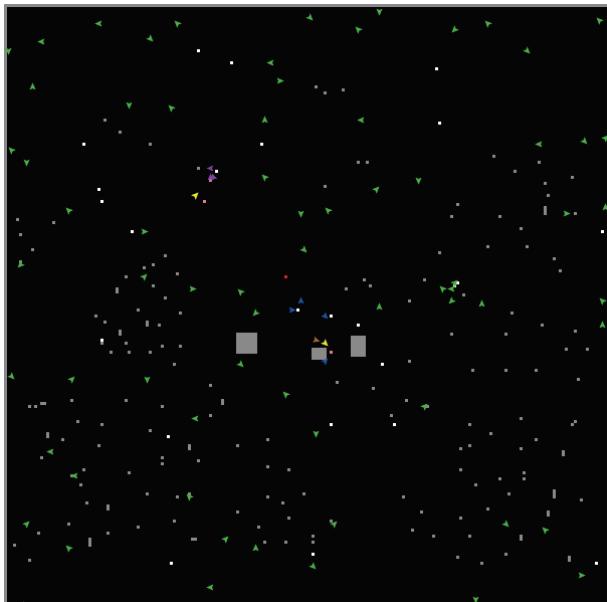


Figura 40: Algorithm 1 + ABC – Rural Mine – 95% di target trovati

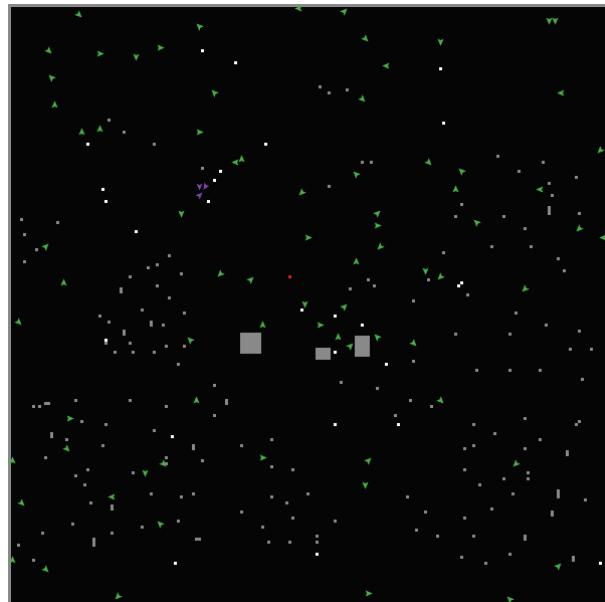


Figura 41: Algorithm 1 + ABC – Rural Mine – 95% di target Elaborati

## SCIADRO CON ELABORAZIONE

“DUMP”

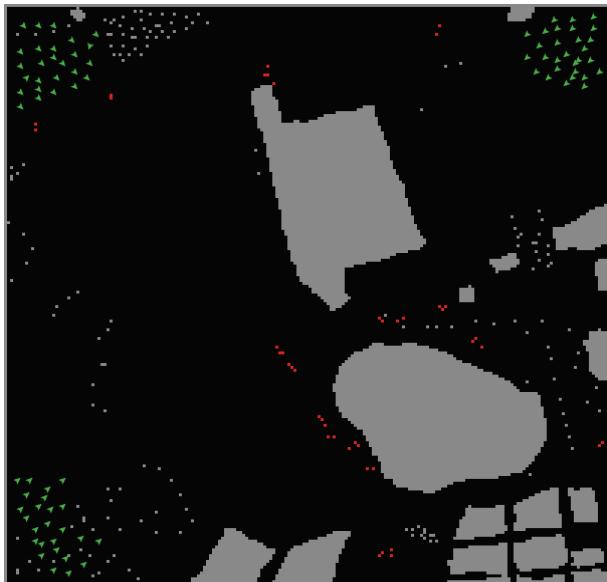


Figura 42: Sciadro con elaborazione – Dump – Stato Iniziale

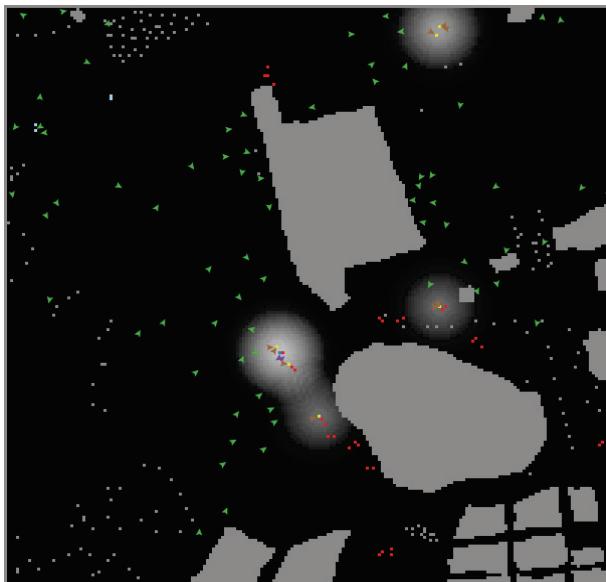


Figura 43: Sciadro con elaborazione – Dump – 25% di target trovati

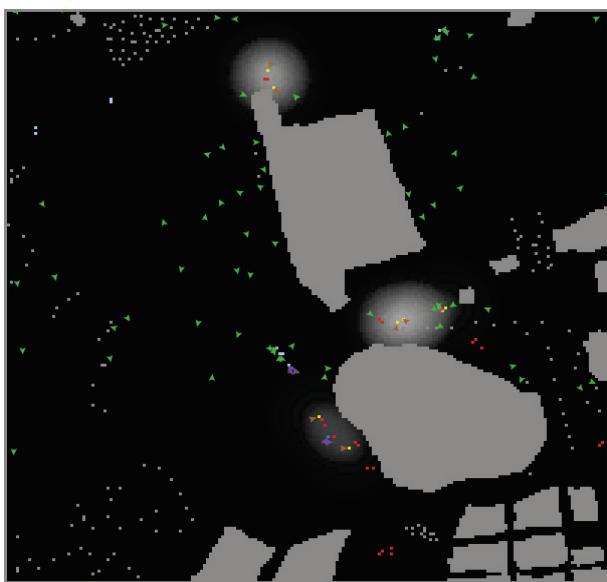


Figura 44: Sciadro con elaborazione – Dump – 50% di target trovati

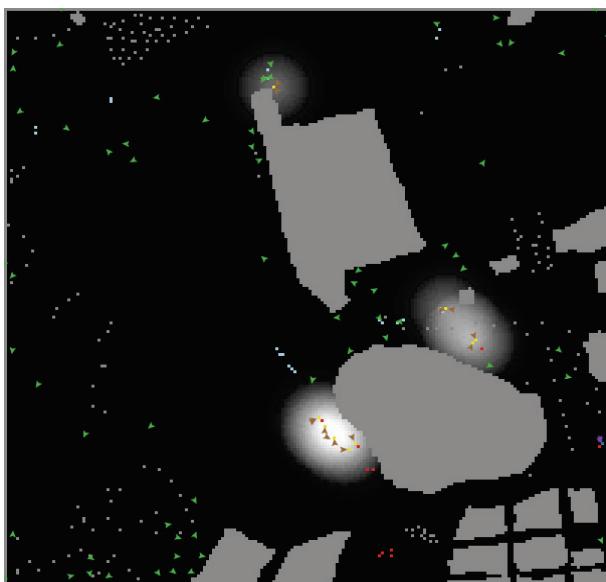


Figura 45: Sciadro con elaborazione – Dump – 75% di target trovati

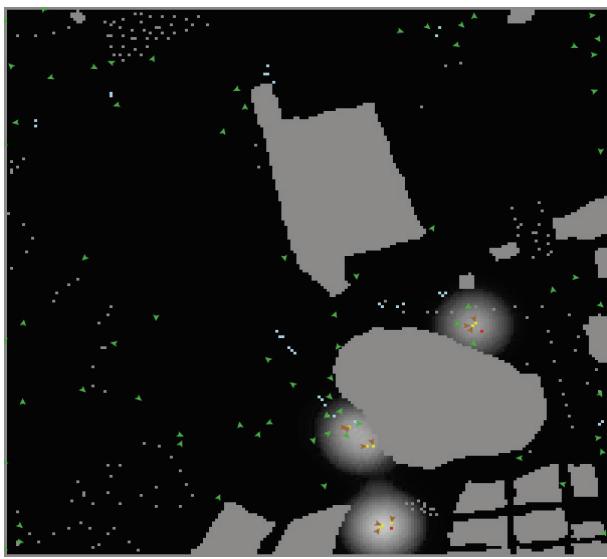


Figura 46: Sciadro con elab. – Dump – 95% di target trovati

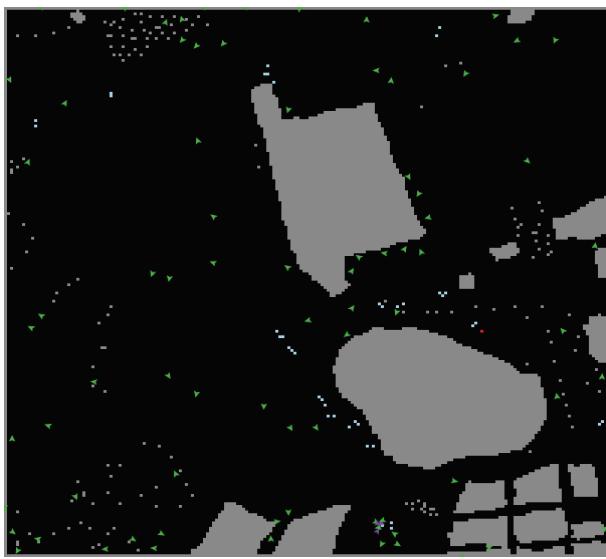


Figura 47: Sciadro con elaborazione – Dump – 95% di target Elaborati

---

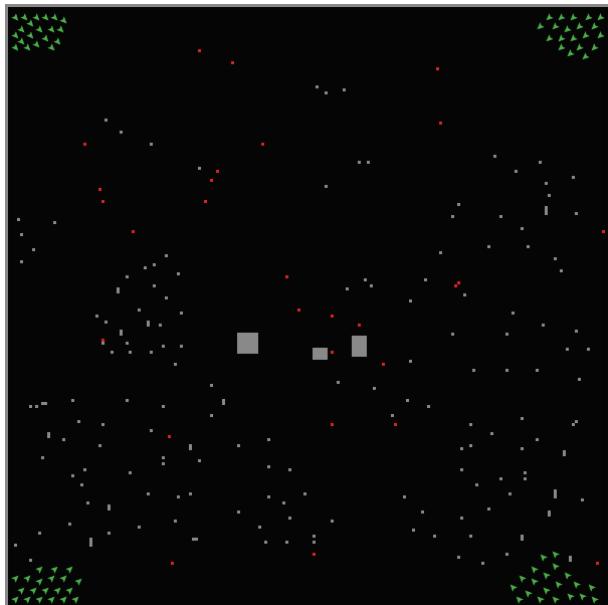
"RURAL MINE"

Figura 48: Sciadro con elaborazione – Rural Mine – Stato Iniziale

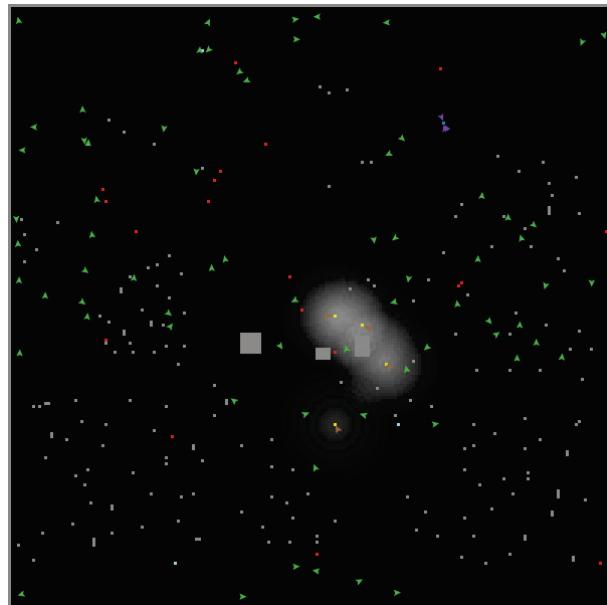


Figura 49: Sciadro con elab. – Rural Mine – 25% di target trovati

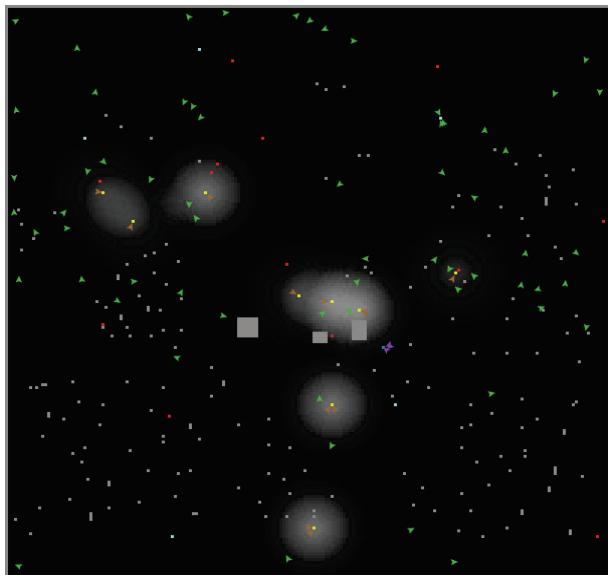


Figura 50: Sciadro con elab. – Rural Mine – 50% di target trovati

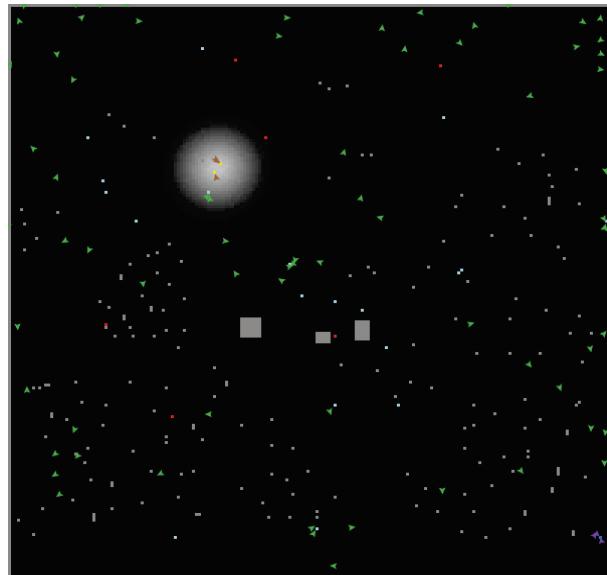


Figura 51: Sciadro con elab. – Rural Mine – 75% di target trovati

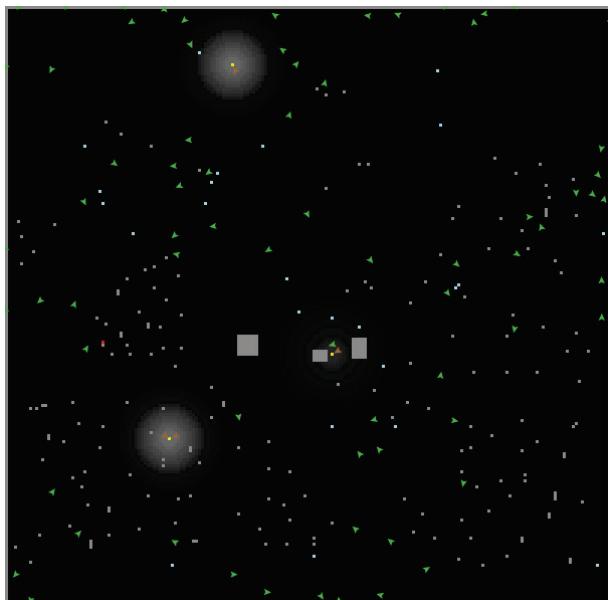


Figura 52: Sciadro con elab. – Rural Mine – 95% di target trovati

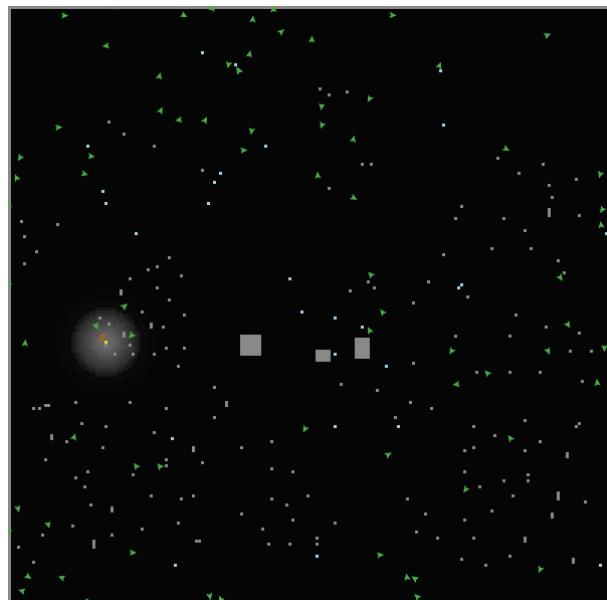


Figura 53: Sciadro con elab. – Rural Mine – 95% di target Elaborati

## OSSERVAZIONI

### ALGORITHM 1 + ALGORITHM 2 (FTS)

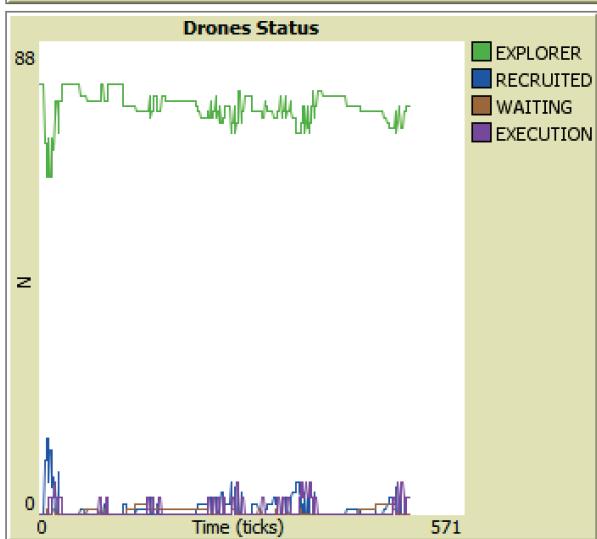
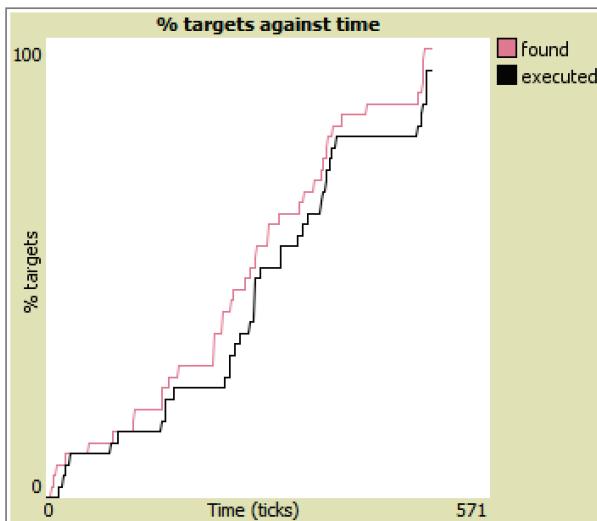


Figura 54: Grafici di Dump

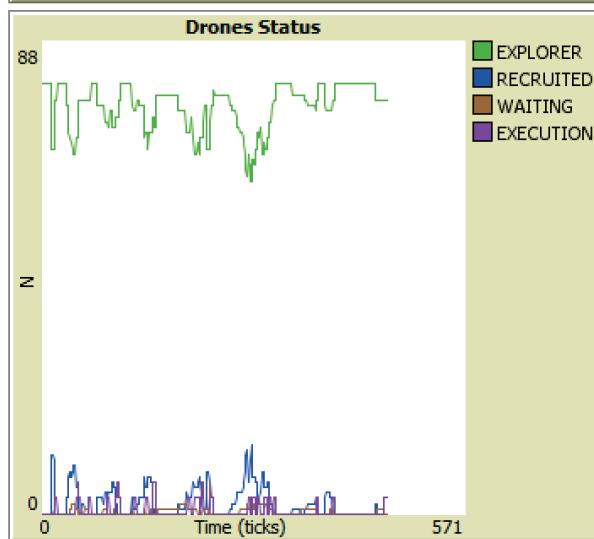
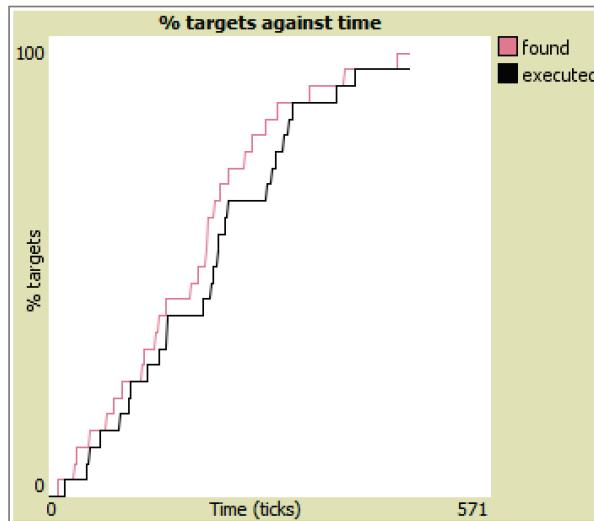


Figura 55: Grafici di Rural Mine

Dai test effettuati si può osservare:

- Come i droni tendano a distribuirsi omogeneamente sul territorio, dopo un certo numero di tick, grazie alle tracce feromoniche negative.
- Che la velocità di esplorazione ed esecuzione dei target, da parte dei droni, è risultata essere di tipo pressoché lineare in entrambi gli scenari. Si può però notare come in *Dump* la presenza di cluster porti le funzioni ad avere dei periodi di stallo seguiti da rapide ascese nel numero di target esplorati ed elaborati, in concomitanza della scoperta dei due principali cluster (figura 8-9).
- Che i tempi di elaborazione non si discostano molto da quelli di esplorazione, ciò indica che generalmente sono presenti un numero sufficiente di droni nel raggio di reclutamento di un coordinatore per iniziare l'elaborazione.
- Come i droni in stato di esplorazione rimangano saldamente la maggioranza assoluta, ciò permette di evitare situazioni di eccessiva attesa da parte di droni coordinatori o in stato di waiting e un'esplorazione più veloce del territorio.

---

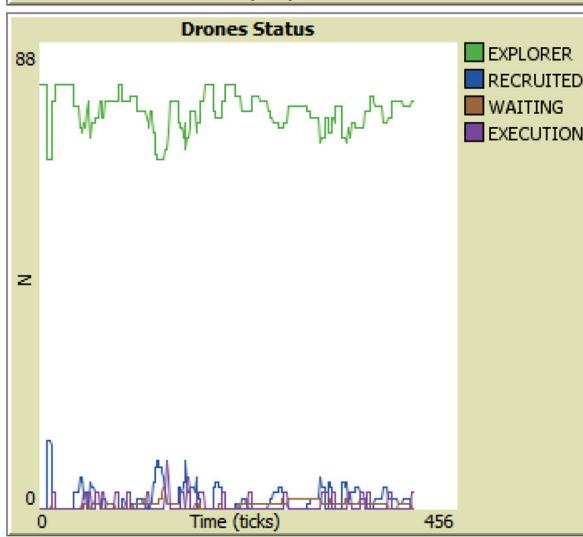
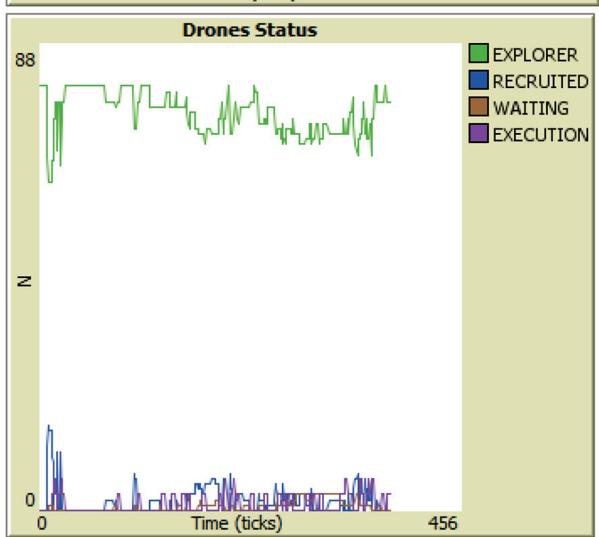
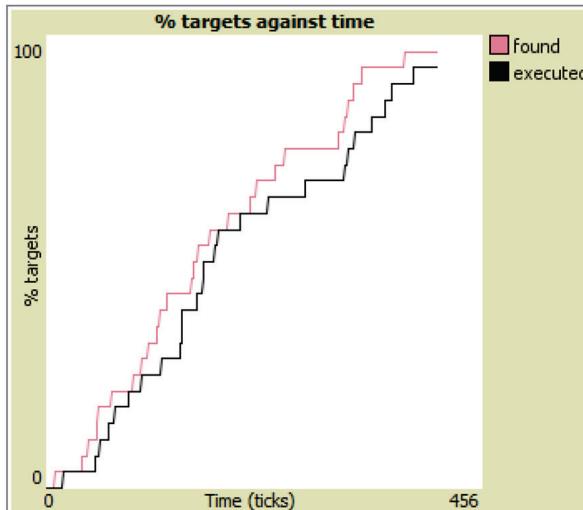
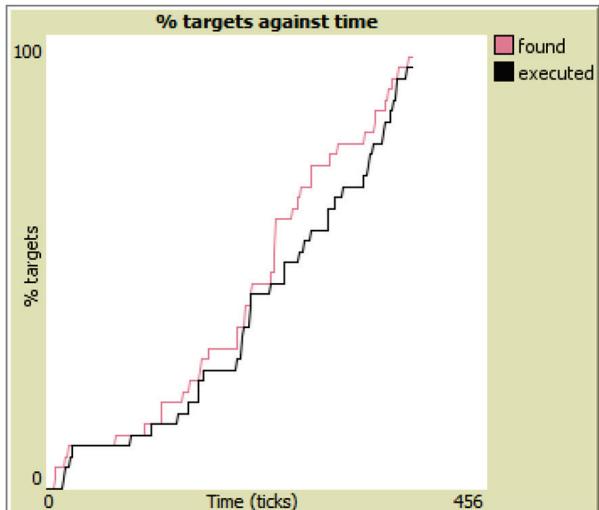
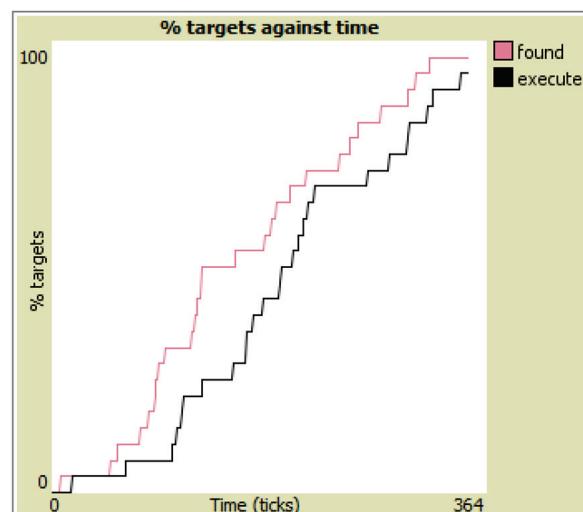
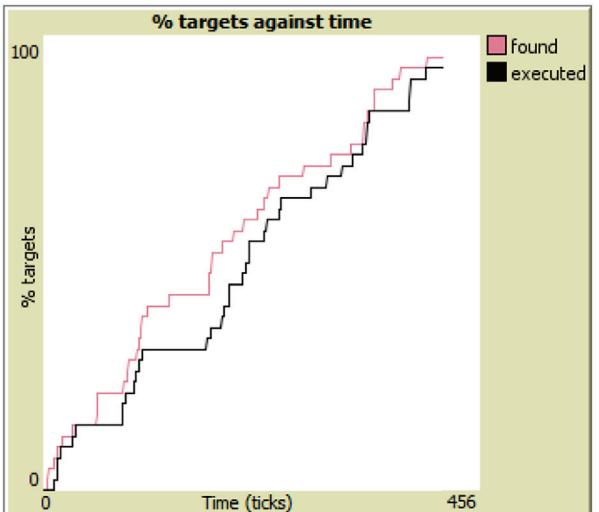
**ALGORITHM 1 + ALGORITHM 3 (PSO)**


Figura 56: Grafici di Dump

Figura 57: Grafici di Rural Mine

Il comportamento di PSO è analogo a quello di FTS anche se la velocità di esplorazione ed elaborazione è più lineare in *Dump*, ciò è probabilmente dovuto alla scoperta simultanea dei principali cluster (figura 20-21).

---

**ALGORITHM 1 + ALGORITHM 4 (ABC)**


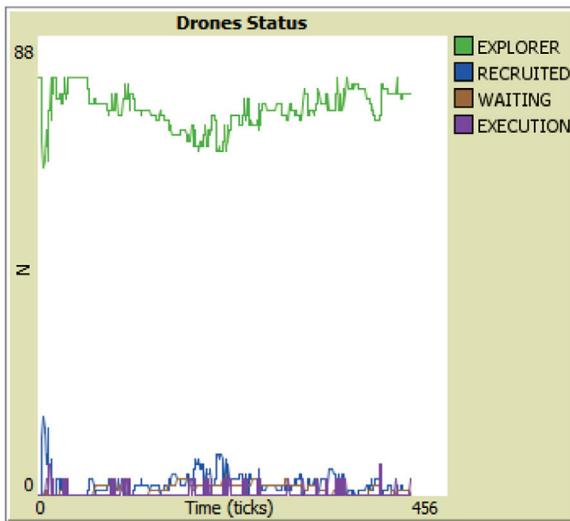


Figura 58: Grafici di Dump

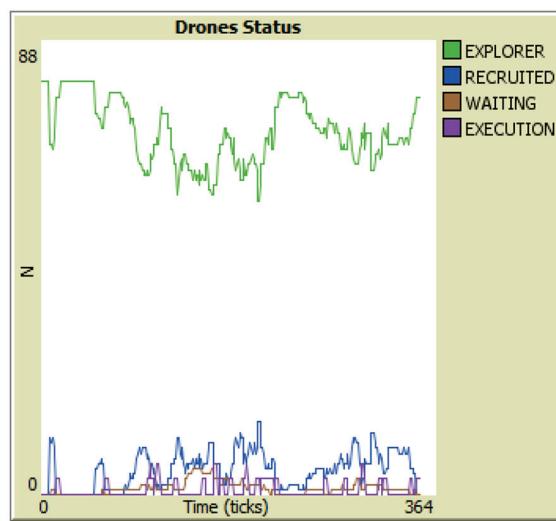


Figura 59: Grafici di Rural Mine

Il comportamento di ABC è simile a quello degli altri algoritmi bio-inspirati, l'aggiunta del meccanismo per cui due terzi dei droni reclutati tendono a uscire dall'area di reclutamento del coordinatore o rimanere fermi, comporta però un aumento della differenza tra i tempi di esplorazione ed elaborazione, ciò probabilmente è dovuto alla presenza di un numero maggiore di droni reclutati. Da notare come in uno scenario quale *Rural Mine*, in cui sono presenti tanti target omogeneamente distribuiti, il numero di droni reclutati contemporaneamente risulti essere ancora maggiore (figura 38). Da queste osservazioni si evince come un elevato numero di target potrebbe portare ad una dilatazione dei tempi maggiore in ABC rispetto a FTS e PSO.

#### CONFRONTO TRA “SCIADRO CON ELABORAZIONE” E GLI “ALGORITMI BIO-INSPIRATI”

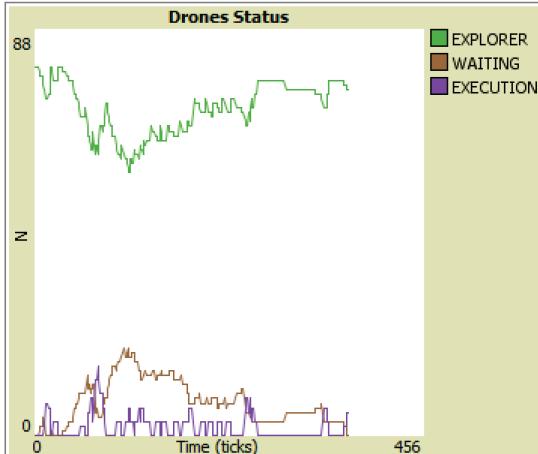
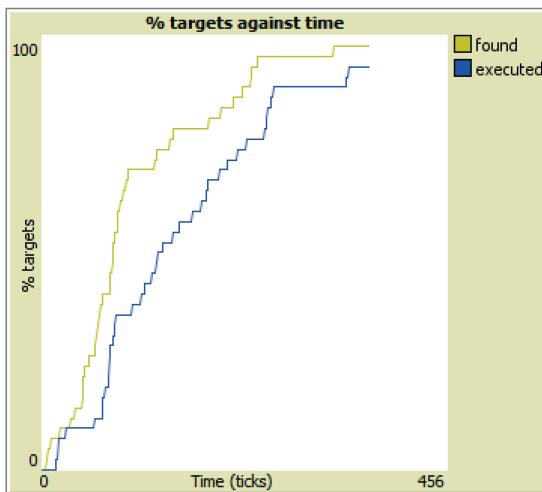


Figura 60: Grafici di Dump

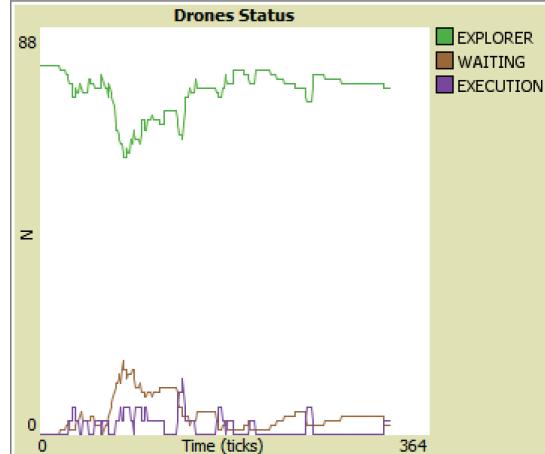
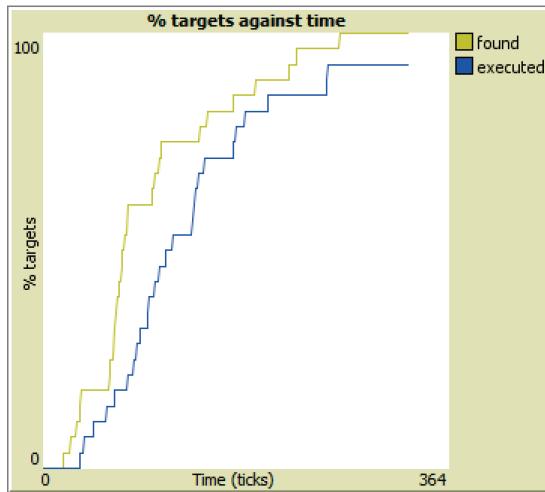


Figura 61: Grafici di Rural Mine

Dalle simulazioni si possono trarre alcune osservazioni:

- Come i droni in *Sciadro* tendano a volare a gruppi, a causa del meccanismo di *flocking*, al contrario dei droni degli algoritmi bio-inspirati, che tendono a distribuirsi. Inoltre, in sciadro i droni si muovono linearmente in assenza di eventi a priorità maggiore, perciò tenderanno ad andare verso i bordi degli scenari. Questi due fenomeni facilitano l'esplorazione e l'elaborazione di target in scenari con ostacoli di dimensioni considerevoli, mentre penalizzano l'esplorazione in scenari con pochi ostacoli, in quanto esploreranno meno le zone centrali dello scenario. Negli algoritmi bio-inspirati i droni sono facilitati nell'esplorazione di aree con pochi ostacoli, sono però penalizzati in aree con presenza di grossi ostacoli che possono perfino portare allo stallo di droni reclutati in strutture di forma concava, se dal lato convesso è presente un coordinatore.
- Che anche in *Sciadro* i tempi di esplorazione e elaborazione sono di tipo lineare, anche se presentano dei rallentamenti nella fase finale dell'esplorazione, probabilmente dovuti al fenomeno della distribuzione dei droni verso i bordi.
- Che in *Sciadro* i tempi di esplorazione e elaborazione dei target possono discostarsi anche significativamente l'uno dall'altro, ciò lo fa assomigliare al comportamento di ABC in presenza di molti target distribuiti, con i relativi problemi.
- Come anche in *Sciadro* i droni esploratori rimangano in maggioranza assoluta, ma, analogamente ad ABC, i tempi di elaborazione possono dilatarsi in maniera maggiore rispetto a FTS e PSO al crescere del numero di target, a causa del maggior numero di droni in stato di waiting che si verrebbero a formare.



*Stefano Petrucci*

