Chapter

Team Exploration of Environments Using Stochastic Local Search

Ramoni O. Lasisi and Robert DuPont

Abstract

We investigate the use of *Stochastic Local Search* (SLS) technique to explore environments where agents' knowledge and the time to explore such environments are limited. We extend a work that uses evolutionary algorithms to evolve teams in simulated environments. Our work proposes a formalization of the concept of state and neighborhood for SLS and provides evaluation of agents' teams using number of interesting cells. Further, we modify the environments to include goals that are randomly distributed among interesting cells. Agents in this case are then required to search for goals. Experiments using teams of different sizes show the effectiveness of our technique. Teams were able to complete exploration of *more than* 70% of the environments, while in the *best cases*, they were able to complete explorations of *more than* 80% of the environments within limited time steps. These results compare with those of the previous work. It is interesting to note that all teams of agents were able to find on average all the goals in the three environments when the size of the grid is 12. This is a 100% achievement by the agents' teams. However, performance can be seen to degrade as the environments' sizes become larger.

Keywords: agents, teams, stochastic local search, state, neighborhood, environment, experiments

1. Introduction

Autonomous agents in complex environments may need to work together as *teams* to achieve desired goals. This is an important feature of most multiagent environments where individual agents lack all the required capabilities, skills, and knowledge to complete tasks alone. These environments can model real-world problem domains where agents' knowledge and available time to complete tasks in such domains are limited. Agents may thus resort to *team formation* to complete tasks. Team formation or coalition formation are simple models of short-term cooperation [1, 2] where agents complete specific tasks.

Examples of team formation can be found in *business* (e.g., organizations forming teams to make more sales and hence, more profits), in *academia* (e.g., professors forming teams to publish articles), in *search and rescue* (e.g., robotic agents forming teams in large natural disaster environments to save life and property), and in *network security* (e.g., agents forming teams to determine critical points

where checkpoints should be placed in a network to hinder adversaries' communication or movement). Since our mundane day to day activities are not exempted from team formation, we cooperate with others to solve problems that may be otherwise difficult for us to complete alone. A number of factors can be attributed to this difficulty, including time criticality of tasks, distribution of individual skills and resources, the need for an agent's presence in multiple work places simultaneously, and many more. An interesting number of works have employed various forms of team formation and search strategies in solving problems related to search or exploration. See for example the works of [3–9].

Here is a straightforward motivation for the problem we study. Consider a rescue operation in an aircraft crash site where search for survivors may be guaranteed in the first few hours of the crash. Agents neither know where survivors are located nor have enough time to explore the environment for victims. They need, preferably as teams, to devise methods that systematically explore the environment to achieve the desired goal. It is not difficult to see that this example and many other similar real-world domains can be modeled as that of *search problems*. This obviously raises the following important question: *How can teams of agents efficiently explore relatively difficult environments using appropriate search strategies that achieve acceptable outcomes?* This research provides an investigation of the use of *Stochastic Local Search* (SLS) technique to explore complex environments where agents' knowledge and the time to explore such environments are limited. We model the problem as that of an instance of a *search problem* and develop SLS techniques that enable intelligent exploration of such relatively difficult environments by teams of agents.

SLS algorithms have made significant success in solving many hard problems [10] which involve search of well-defined solutions spaces (or states). A model of SLS algorithms is defined to include a *neighborhood* and an *evaluation function*—both of which are specific to different problems. The goal of an agent using SLS algorithm is to seek a state s from the set of possible states S in the problem domain that optimizes some measures [11]. A neighborhood, $N(s)$, is defined for each state s. $N(s)$ is the set of all possible *successor states* that are local to s i.e., the set of all possible states that an agent transits into from the current state s. The evaluation function is defined to exploit the current knowledge of the neighborhood and then *stochastically* selects a successor state $s' \in N(s)$. This simple method of choosing the successor state by the evaluation function may further be guided towards solutions that optimize goals measures using heuristics. The neighborhood and evaluation function capture two interesting features of SLS algorithms that we exploit in this work.

We model the problem described in the motivation above by using three different two-dimensional grids to represent environments that agents need to explore. Some cells within the grids are referred to as being "interesting," such as *possibly* having victims in them. We then randomly distribute *goals* among the interesting cells. Goals in this work represent some desirable situation that we want agents to achieve. In the motivation above, the presence of a victim in an interesting cell will represent a goal. The three environments are further constrained such that not all interesting cells contain goals, thus agents do not have background knowledge of the environments. Agents in our model are required to devise techniques to search and find as many goals within a limited amount of time steps. The performance of agents in these environment are evaluated based on a number of factors, including the amount of goals found with respect to the number of agents in teams, type, and size of environments. The main contributions of this work are as follows:

1. We provide a formalization of the *concept of state and neighborhood* in three different simulated *random*, *clumped*, and *linear* environments described in [12].

2. We provide agents performance evaluation in the three environments using the number of interesting cells found by teams as done in [12] and our previous work [13]. We further modify the environments to include limited number of goals that are randomly distributed among the interesting cells. Agents in this case are thus required to search for goals rather than interesting cells as done in [12, 13].

3. We model agents, their methods of movements in the environments, and provide a clever design of an *evaluation function* that agents can use to navigate the three environments.

4. Finally, we provide an implementation of a model that allows agents to operate on these environments using the proposed evaluation function.

Several simulations to mimic real-world environments were completed using different dimensions of the three environments and varying agents' team sizes. The results of these simulations compare favorably with those of a previous work that was used as a benchmark. In particular, the proposed model avoids such expensive cost of extensive time requirements of evolutionary learning by agents. This is made possible as agents in this model are not subjected to training before being deployed to the testing environments. They only conduct local searches of the environments from their current locations using two important features from the SLS algorithms: neighborhood and evaluation function.

2. Related work

SLS algorithms have been successfully applied to many hard problems including Traveling Salesman Problem, Graph Coloring Problem, Satisfiability Problem in Propositional Logic, and more [10, 14]. Common SLS algorithms include *simulated annealing*, *hill climbing*, and evolutionary inspired *genetic algorithms* [15]. As highlighted in the previous section, the definitions of a neighborhood and its associated evaluation function in SLS algorithms are specific to the problem domain. The real *novelty* in the employment of SLS techniques to construct an algorithm comes from how elegant the neighborhood and the evaluation function are defined for the problem domain such that the algorithm is well-guided towards feasible solutions within a short period of time.

Soule and Heckendorn [12] describe three environments on which their work is based. We reproduce these environments and their descriptions since we have used them to evaluate our proposed SLS technique. Each of the three environments is composed of two-dimensional grids of 45×45 containing some percentages of *interesting* cells that are distributed in some ways within the environments. A cell is said to be *interesting* if it contains some sub-goals or information that leads to the desired goal of a team. Using the example of the previous section, a cell in this case will be *interesting* if it contains, say, a survivor or victim from a crash.

The environments are named according to how the number of interesting cells are distributed in the grids. They are referred to as *random*, *clumped*, and *linear*.

Figures 1–3 depict sample schematic views of random, clumped, and linear environments for a 45 × 45 two-dimensional grids. The interesting cells in each environments are shown shaded. In a random environment, each cell has a uniform 20% chance of being interesting. For clumped environment, exactly 20% of the cells are interesting and are stochastically clumped in the four corners of the grids. Finally, in the linear environment, exactly 10% of the cells are interesting and they are distributed randomly along eleven rows in the environment. The same eleven rows are always used, but the exact placement of interesting cells within the rows is random. These environments model applications in the real-world. An environment

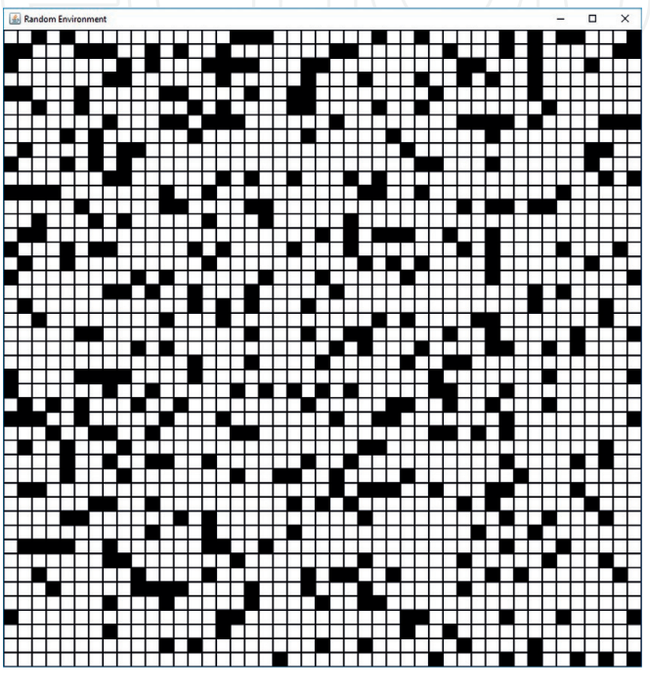

Figure 1.
A schematic view of a random environment for a 45 × 45 two-dimensional grid.

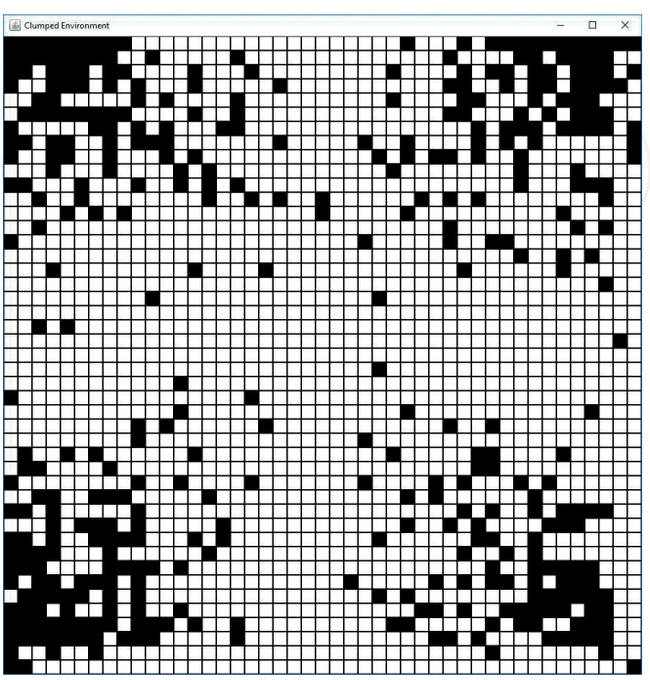

Figure 2.
A schematic view of a clumped environment for a 45 × 45 two-dimensional grid.

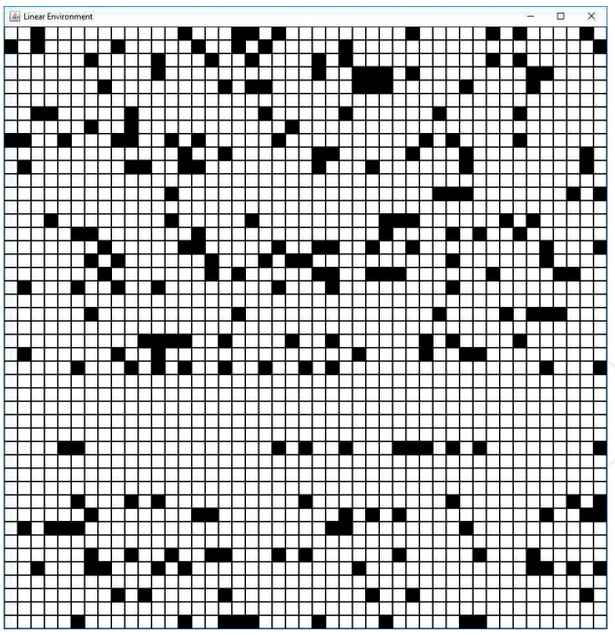

Figure 3.
A schematic view of a linear environment for a 45 × 45 two-dimensional grid.

might represent a minefield with the interesting cells representing positions of potential mines or geological formations [12]. Teams evolved to explore environments may also represent automated planetary surveying team [5].

Soule and Heckendorn use evolutionary algorithms to implement a multiagent team training algorithm called *Orthogonal Evolution of Teams* (*OET*) to *evolve* teams of agents. The three environments above alternatively serve as both the *training* and *testing* environments to evaluate the performance of their OET algorithm. They consider evolution of *heterogeneous* multiagent teams (i.e., teams of agents with specialized roles). There are two types of specialized agents in their work: *scouts* and *investigators*. The scouts and investigators are respectively responsible for finding as much as possible interesting cells and marking them as investigated. Unlike our approach however, where all agents are limited to moves of *length one* in a single time step in the environments, the scouts are allowed a move of *length two* in a single time step. Results from our work using SLS technique to explore different environments compare with those of Soule and Heckendorn's with performances within similar ranges. However, it is not yet clear how fair that comparison can be justified since their work employs evolutionary algorithm which come with extensive time requirements of evolutionary learning and huge time and costs of training for agents before they are deployed to actual testing environments.

3. Problem formalization and solution approach

Given any of the three environments (i.e., random, linear, and clumped) described in the previous section and a number of autonomous agents (each with a limited knowledge of the environments), the problem we attempt to solve is to form teams of agents that intelligently explore as much as possible interesting cells and/or goals in the grids within a limited amount of time. Our attempt in solving this problem uses a model that employs techniques from SLS algorithms. We provide in this section, a formalization of the framework and implementation details for state and neighborhood, evaluation function, and a description of the simulation employed in our research.

3.1 Framework for state and neighborhood

We present our framework of the concept of *state* and *neighborhood* in any of the three environments. Denote by c_{ij} a cell in any grid of an environment where $i, j \in \mathbb{N}$ are the Cartesian coordinates of the cell c_{ij}.

Definition 1. *A state s with a reference cell, c_{ij}, in an environment consists of the reference cell c_{ij}, and all immediate cells $c'_{i'j'}$ from c_{ij} such that $|i - i'| = 1$ or $|j - j'| = 1$.*

It is clear from Definition 3.1 that a state is composed of a 3×3 sub-matrix within an environment when the reference cell of the state is not close to or on any of the boundary cells. Note that for any $n \times n$ two-dimensional grid of the three environments, n is a multiple of 3. This constraint allows us to correctly map states to the $n \times n$ grids. An example of a state labeled s is shown in **Figure 4** with a reference cell c_{ij}. The immediate cells from c_{ij} are shaded in gray. The set of all possible states in the problem domain constitutes the search space we seek for feasible solutions (i.e., finding goals and/or as much as possible interesting cells).

Definition 2. *The neighborhood $N(s)$ of a state s consists of all states s' that share boundary with s.*

Figure 4 shows an example of a neighborhood $N(s)$ for a state s. States s_1, s_2, s_3, and s_4 (shaded in black) all share boundary with state s. Thus, $N(s) = \{s_1, s_2, s_3, s_4\}$. The size, $|N(s)|$, of the neighborhood $N(s)$ of a state s is $2 \leq |N(s)| \leq 4$, depending on whether or not s is close to any of the boundaries of the environments. The neighborhood of any regular state within the boundaries consists of only four neighboring states as shown above. **Figure 5** shows a schematic view of some possible neighborhoods for different states with reference cells, c_{ij}. Notice how incomplete, both in the number of cells and neighbors on how some of neighborhoods were defined because of the positions of the reference cells.

3.2 Implementation of state and neighborhood

We present implementation details of the framework developed in the previous section. We provide abstraction of state and neighborhood in an environment using

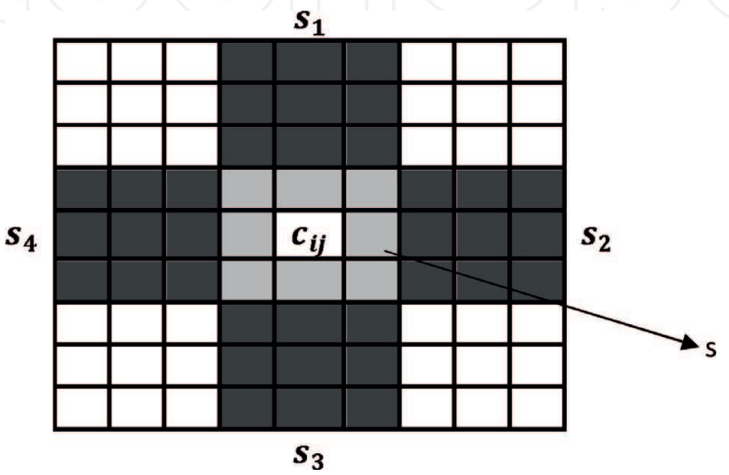

Figure 4.
A view of a state (with reference cell c_{ij}), denoted s, and a neighborhood $N(s)$ for s. $N(s) = \{s_1, s_2, s_3, s_4\}$.

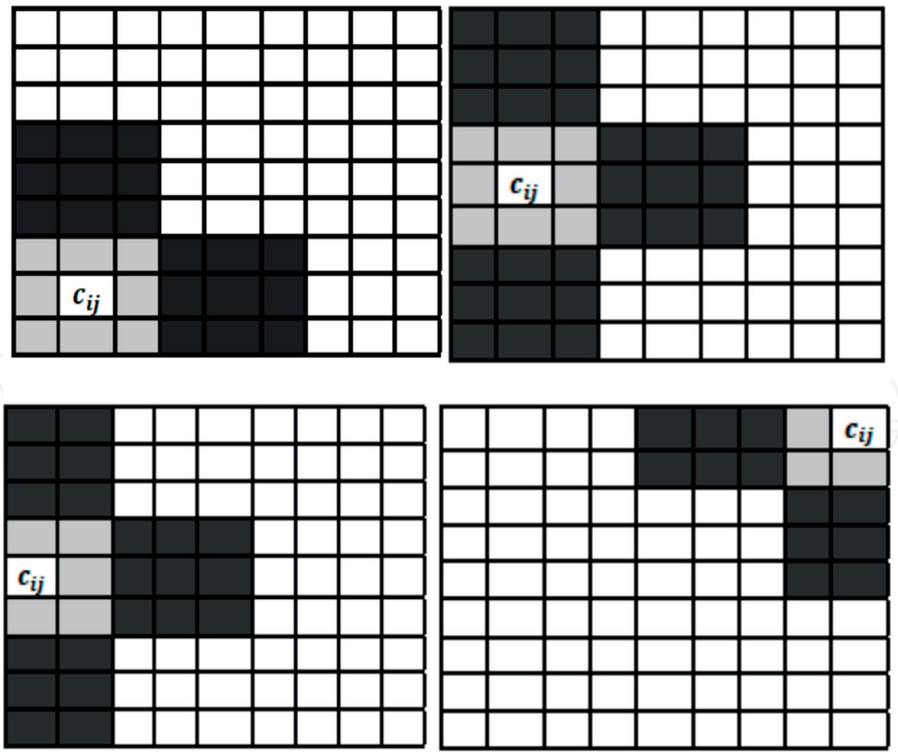

Figure 5.
A schematic view of some possible neighborhoods for different states with reference cells, c_{ij}.

the Java programming language. The following Java classes, *Cell*, *Agent*, *Environment*, and *State* are *partially* shown to include only relevant data fields and/or methods that are needed to understand the discussion of our implementation of state and neighborhood in this section.

```
public class Cell{
      //data fields
      private int x;
      private int y;
      //methods
      void setXY(int x, int y);
      int getX();
      int getY();
}
public class Agent {
      // data field
      private int[] location;
      // method
      int[] getLocation();
}
public class Environment implements Runnable {
       // data fields
      private int gridSize;
      private String [][] grid;
      private boolean [][] visitedCells;
      private Agent [] agents;
}
public class State {
      // data fields
```

```
    private int [] refCell;
    private int gridSize;
    // method
    Cell [] createCells(int [] refCell, int gridSize);
}
```

We first consider how to locate a state in an environment. Using the Agent class above, an agent is always aware of its current location. All agents start exploration of an environment from some locations that are determined randomly. These locations correspond to certain reference cells in the environment's grid. No two agents are allowed to start from the same reference cell. Since a state is made up of a 3×3 sub-matrix and the dimensions of an environment grid is a multiple of 3, we can exactly locate the starting cell of a state, given its reference cell. For any reference cell, $ref\ Cell[] = \{i,j\}$, i.e., a location $grid[i][j]$ in the Environment class above, the starting cell of the state that contains the reference cell is given as:

$$grid[(i/3)*3][(j/3)*3] \tag{1}$$

Thus, it is straightforward to determine the states that all the cells in the grid of an environment belongs. Given a reference cell for a state, the following provides an implementation that returns the start indices of the state.

```
public int [] getStateStartIndices(int [] refCell) {
    int i = refCell[0];
    int j = refCell[1];
    return new int [] {(i / 3) * 3, (j / 3) * 3};
}
```

When a state has been explored by an agent, the state is marked as investigated. A state is considered visited if the reference cell for the state and all of its immediate cells have been marked as investigated. Since agents are always aware of their locations, we implement this functionality for each agent using the following method:

```
public void setVisited(int [] refCell){
    if(!this.isVisited(refCell)) {
      int [] indices = getStateStartIndices(refCell);
      int x = indices[0];
      int y = indices[1];
      for(int i = x; i < gridSize && i < x + 3; i++)
        for(int j = y; j < gridSize && j < y + 3; j++)
          visitedCells[i][j] = true;
  }
}
```

Parameter *refCell* is the reference cell of the state, *gridSize* is the size of the grid, and *visitedCell* is a boolean 2-dimensional array that keeps track of cells in the grid that correspond to states already investigated by agents. We now turn attention to the implementation detail of neighborhood.

```
public State [] getNeighborhood(int [] refCell) {
    State [] neighborhood = null;
  //check top left corner
  if((refCell[0] == 0 || refCell[0] == 1) &&
        (refCell[1] == 0 || refCell[1] == 1)){
    neighborhood = new State[2];
```

```
    for(int i = 0; i < neighborhood.length; i++)
        neighborhood[i] = new State();
    //right state
    int [] s2 = {refCell[0], Math.min(refCell[1] + 3, gridSize - 1)};
    neighborhood[0].createCells(s2, gridSize);
    //bottom state
    int [] s3 = {Math.min(refCell[0] + 3, gridSize - 1), refCell[1]}; neighborhood
[1].createCells(s3, gridSize);
  }
//check left column
//check bottom left corner
//check bottom row
//check bottom right corner
//check right column
//check top right corner
//check top row
//everything else
return neighborhood;
}
```

Given the reference cell of a state, the method *getNeighborhood* above provides an implementation that determines the neighborhood of that state. The method first checks the location of the reference cell to determine the size of the neighborhood, then returns the appropriate states in the neighborhood as demonstrated in **Figure 5**. For the part of the code shown in the method, the reference cell of the state, say, s, under consideration falls on the top left corner of the grid, so only two states (i.e., the right and bottom states that share boundaries with s) are returned for the neighborhood. All other possible neighborhood are handled by the method as indicated by comments in the lower part of the method.

3.3 Evaluation function

Agents in our model use an *evaluation function* to guide selection of successor states to transit into. The evaluation function depends on the framework of the state and neighborhood we developed in the previous section. Algorithm 1 gives the pseudocode of our evaluation function.

Algorithm 1: *Successor State* $(\boldsymbol{a}, \boldsymbol{s})$**.**

Input: Agent a and the current state s of a.
Output: A successor state $s' \in N(s)$.

procedure *Successor* State (a, s)
  SuccStates $\leftarrow \emptyset$
  for each state $s' \in N(s)$ **do**
    if s' has not been visited **then**
      if there exists no other agent $a' \neq a$ in s' **then**
        SuccStates $\leftarrow$ *SuccStates* $\cup \{s'\}$
  if *SuccStates* is empty **then**
    randomly select an s' from $N(s)$
  else
    randomly select an s' from *SuccStates*
  return s'
end procedure

Algorithm 1 i.e., SuccessorState (a,s) accepts two inputs—an agent a, and the current state s, of the agent. It outputs a successor state s' if one exists, otherwise it stochastically selects any state in $N(s)$ as the successor. In a single exploration of an environment by all agents in our system, each agent calls *SuccessorState*(a,s) algorithm once to determine the next state to transit into. It is not difficult to see that the total running time of a single time step of the exploration of an environment is *linear* in the number of agents, since *SuccessorState*(a,s) algorithm only examines a constant number (i.e., the four) neighboring states to s. We also show that an agent does not remain infinitely in a particular state in situations where *SuccStates* is empty, at which time the evaluation function stochastically selects any state from $N(s)$. We refer to situation where *SuccStates* is empty as a situation of "no progress". The "no progress" situation is eliminated in the next attempt by the evaluation function to find a successor state and does not occur often as described below.

Suppose an agent a is currently in a state s. Consider a certain attempt t by the evaluation function to find a successor state for a which results in a situation of "no progress". The evaluation function stochastically selects a state s' from $N(s)$ for a to transit into. Now consider the next attempt t' by the evaluation function to find a successor state for a where, as we know, the agent is currently in a new state s' following state s. Observe that the neighborhood for state s' in this attempt t' is different from the neighborhood for state s in the previous attempt t i.e., $N(s') \neq N(s)$ and s is now one of the neighboring states of s' i.e., $s \in N(s')$. Suppose again that attempt t' by the evaluation function to select a successor state for s' results in a situation of "no progress". The evaluation function again stochastically selects a state from $N(s')$. Note that the probability of selecting the state s i.e., the previous state from $N(s')$ as the successor to s' is only $\frac{1}{4}$ as against the probability $\frac{3}{4}$ of selecting any of the remaining three new states from $N(s')$.

Observe that the number of attempts required by the evaluation function until any one of the three states in $N(s')$ apart from state s is selected follows a *geometric random distribution* with probability $p = \frac{3}{4}$. Thus, the *expected* number of attempts required by the evaluation function until any one of these three states is selected is

$$p \sum_{i=1}^{\infty} i \cdot (1-p)^{i-1} = \frac{1}{p} = \frac{4}{3} < 2. \tag{2}$$

3.4 Simulation

We form teams consisting of a certain number of agents. One of the team's members is designated as a *leader*. We assume that the leader has some additional computational power than other members. The leader is responsible for maintaining an updated status (i.e., visited states) and communicates same to other members when requested. The leader answers the following queries from members: *Has a given state been visited?* and *Is there an agent in a given state?* These are the queries that are used by the evaluation function. The leader agent does not participate in the actual exploration of individual states. Other agents are responsible for locating and visiting interesting cells and as well as finding goals in the grids. All visited cells, either interesting or not, and whether goals are found or not are marked as investigated. An agent can move from her current location in only one of four directions (i.e., north, east, west, and south) and is limited to moves of *length one* in a single time step. The following three actions, goForward, turnLeft, and *turnRight* are made available to all agents except the leader.

When starting, all agents (except the leader) are randomly distributed in the environment. We describe the procedure used by agents to explore the

environments next. Imagine an agent a currently in a state s that has not been visited. After the exploration of the current state, agent a invokes the evaluation function to determine the successor state to transit into. The successor state guides the decision of the agent on how it *exits* from the reference cell and the *order* it conducts the search of the current state. Having transited into a successor state, agent a determines if its current cell is interesting, and/or a goal is found, records a score, and marks the cell as visited. The agent then performs an *exhaustive search* of the immediate cells to the reference cell of state s. During the exhaustive search, the agent checks if the cells being searched are interesting, and/or goals are found, records scores as appropriate, and subsequently marks the cells as visited. On completion of the search of state s, the status of the state (i.e., visited) is communicated to the team leader.

Figure 6(a) provides a simple illustration of an agent currently in a state with reference cell x which later transits into a successor state s_4 with reference cell y.

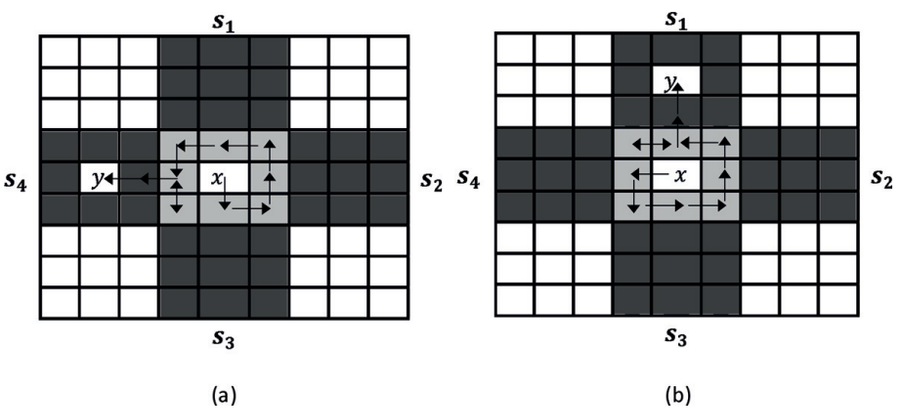

Figure 6.
Exhaustive search of a state by agents. (a) Agent exits reference cell x, search current state in the direction of the arrows, and transits into s_4 with reference y. (b) Agent exits reference cell x, search current state in the direction of arrows, and transits into s_1 with reference cell y.

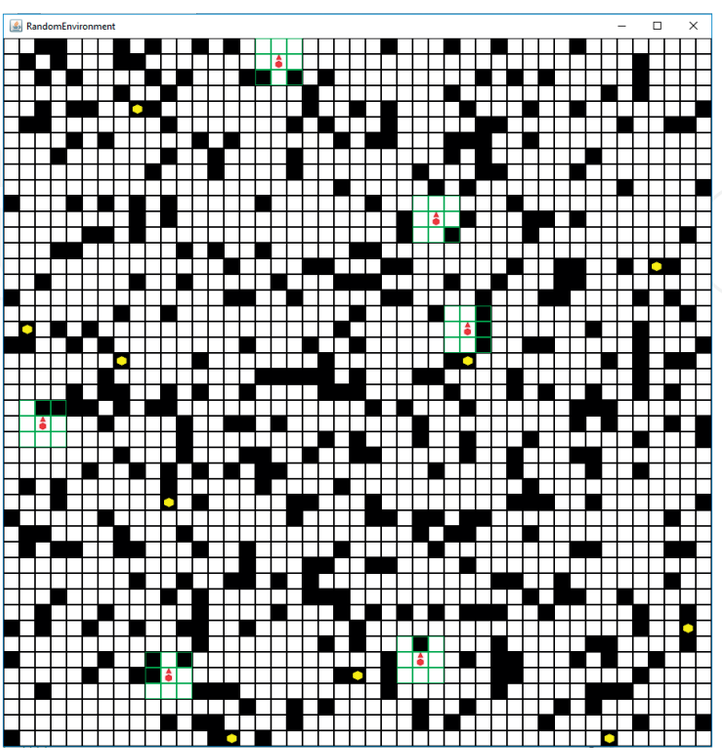

Figure 7.
Random environment showing initial deployment of agents (red) and goals (yellow).

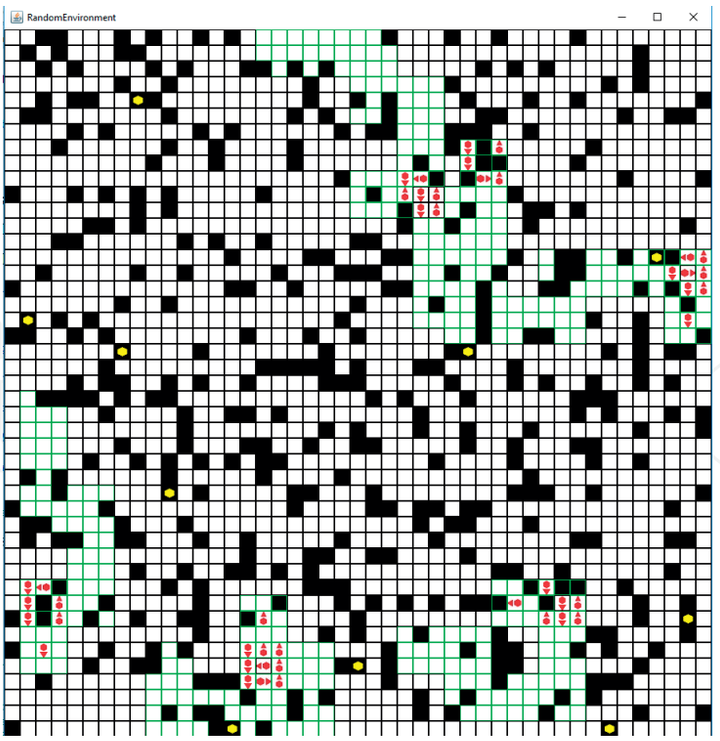

Figure 8.
Random environment showing partial search of the environment by agents. The highlighted areas covered in green have been explored by agents.

The agent first determines its successor state as s_4 using the evaluation function, then conducts an exhaustive search of the immediate cells to the reference cell x in the direction of the arrows for each time step, and finally transits into state s_4. A similar example is depicted in **Figure 6(b)** when the agent transits into state s_1 from the current state. Observe the difference in how the agents in the two figures exit the reference cells of their respective current states and the order in which they conduct their exhaustive search. This difference is due to the fact that the agents transit into different successor states from the current state. At the expiration of the exploration period, we compute the sum of the scores of interesting cells found by each agent as the total score achieved by the team of agents.

Figure 7 shows an example of a random environment with an initial deployment of six agents (red pictures with arrow heads) in the environment. There are also currently ten goals (diamond pictures in yellow) that are also randomly distributed across the grid. The agents will search the environment using the evaluation function and the concept of neighborhood proposed in this work to guide their search. A snapshot of what the search area looks like after few time steps of exploration is shown in **Figure 8**. The highlighted areas covered in green have been explored by agents, while the white areas are yet to be explored. Also, the rectangle of agents shown together illustrate an exhaustive search of that state by an agent. We make the rectangle of agents disappear in the real simulation when the agent complete exploration of the state.

4. Results and discussion

We present results of our extensive simulations in this section. Our findings are based on two different sets of experiments—first, on the percentage of interesting cells found by agents, and second, on the average number of goals found by agents in the three environments. For our study, we use a different set of parameters to

include number of agents in teams, size, and type (random, linear, and clumped) of environments.

4.1 Percentage of interesting cells found by teams of agents

Figure 9 shows the average percentage of interesting cells found by six-member teams of agents for 45×45 random, clumped, and linear environments using the SLS model for 100 trials of the experiments. The corresponding standard deviations from the average percentage of interesting cells found by the agent for the three environments are also shown in **Table 1**.

The average percentage of interesting cells found by agents' teams using the SLS model provides a measure of the level of difficulty of the three environments for the teams. This conversely implies a measure of the relative performance of the teams in each of the environments. **Figure 7** shows that the relative performance of the teams in the random environment ($\sim 74\%$) is higher than that of the linear environment ($\sim 72\%$), which in turn is higher than that of the clumped environment ($\sim 68\%$). Thus, the clumped environment appears to be the most difficult of the three environments, followed by the linear, and then, the random environment. The level of difficulty in the three environments may however be assumed to be relatively close considering how small the *spread* ($74 - 68 = 6$) among the average performance of the teams in the three environments is. This is further evidenced in **Table 1** by the tightness of the standard deviations around the average percentage of interesting cells found by agents' teams in the environments.

An implication of the closeness of the level of difficulty of the three environments is that the SLS model's performance has *less reliance* on these environments. Contrarily, Soule and Heckendorn [12] have shown that the performance of the evolved teams by their model depends on both the training and testing environments. They show that training in either the random or clumped environment is a

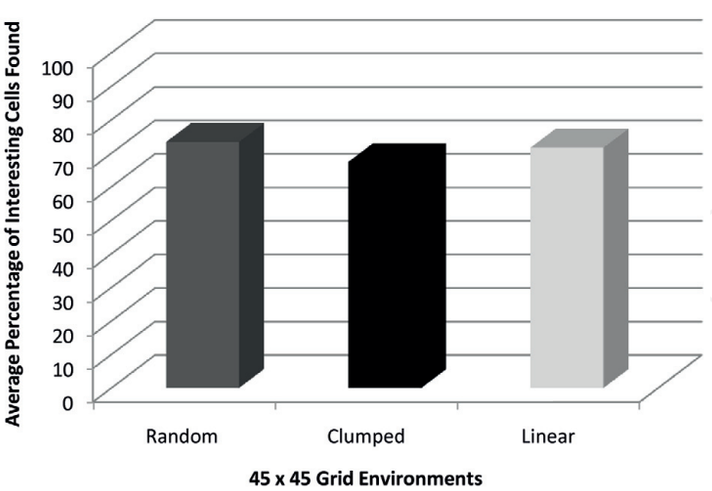

Figure 9.
Average percentage of interesting cells found by six-member teams in 45×45 *grid environments.*

	Random	Clumped	Linear
Standard deviation	1.05	1.31	1.09

Table 1.
Standard deviations from the respective average percentage of interesting cells found by six-member teams in 45×45 *grid environments.*

good training for the other environment, but neither is as good of a training environment for the linear environment. In fact, the performance of the evolved teams when they are trained in either of random or clumped, and later deployed in linear environment, is poor in comparison with when they are deployed in either the random or clumped environment. Recall also that agents in our model are not subjected to training before being deployed to the testing environments. They only conduct local searches of their environments using two important features from SLS algorithms: neighborhood and evaluation function.

For the next set of experiments, we evaluate the effectiveness of the SLS model by measuring the average percentage of interesting cells found by agents' teams, varying the number of agents in the teams, and the grid sizes in the three environments. **Figure 10** shows the average percentage of interesting cells found by agents' teams of different sizes in 45×45 random, clumped, and linear environments. The x-axis indicates the team member sizes while the y-axis is the average percentage of interesting cells found by these teams.

The six-member teams always discover more than 70% of the interesting cells for both the random and linear environments, and more than 65% for the clumped environment on the average. As the size of the teams increases, there is a significant increase in the average performance of the teams in the three environments. The average performance of the teams consistently increases with the teams sizes and reaching a peak value of 95% for both the random and linear environments, and 93% for the clumped environment when the team size is ten. It appears that 10-member team is the *optimal* team size when agents implement the SLS model for the three 45×45 grid environments. This can be confirmed from **Figure 10**.

Increasing the number of agents in the teams beyond ten does not appear to improve average performance of agents. We noticed marginal decrease in the average performance of larger teams—as teams' sizes increase past 10, the average percentage of interesting cells found drops below those of the 10-member teams. See **Figure 10** for performance of agents' teams of sizes 11 and 12 where the average percentage of interesting cells found by these teams are fairly smaller compared to those of the 10-member teams. Our explanation for this unexpected result is that as the number of agents increases, there is an increased chance of team members revisiting already visited cells. Such efforts by agents does not improve the scores (performance) of the teams since the team has already been rewarded during initial

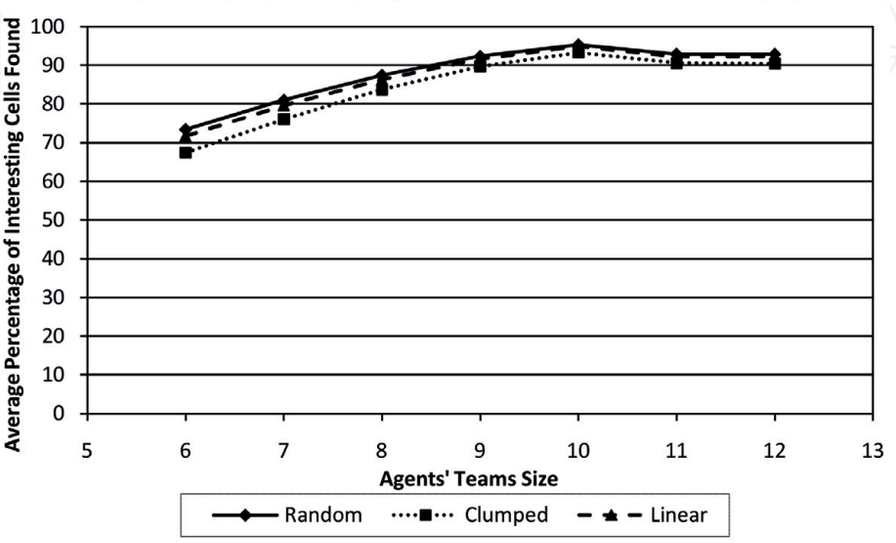

Figure 10.
Average percentage of interesting cells found by agents' teams of different sizes in 45×45 *grid environments.*

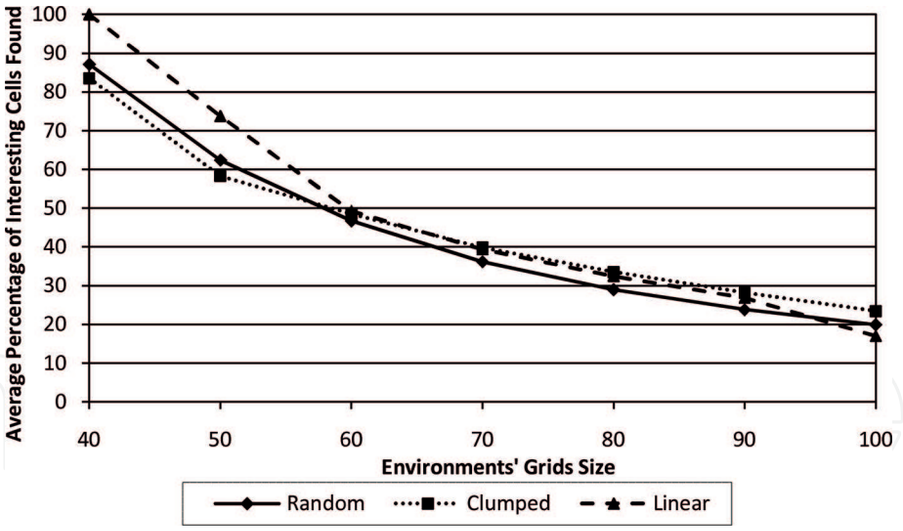

Figure 11.
Average percentage of interesting cells found by six-member teams for different grid sizes of environments.

visits of the cells by some members of the team. In other words, some agents in a team may become redundant as the size of the team becomes large.

Figure 11 shows the average percentage of interesting cells found by six-member teams for different grid sizes in the three environments. The x-axis indicates the grid sizes while the y-axis is the average percentage of interesting cells found by these teams.

The results show, perhaps not too surprising that in general, the average performance of the teams degrade for the three environments as the dimension of the grids increases. A partial explanation for this is that fixing the team size while increasing the dimension of the environments makes members of the teams to be sparsely distributed in the environments. Thus, it will then be more difficult for agents to cooperate as they now require several time steps to move closer to one another in order to cover different parts of the grids. Nonetheless, even at higher dimensions of the grids, agents' teams are still able to achieve some reasonable level of performance. For instance, when the grid size is 100×100, the 6-member teams found more than 20% of interesting cells for the random and clumped environments but below 20% for the linear environment.

4.2 Number of goals found by teams of agents

This second part of the experiments is based on the average number of goals found by agents' teams. We set the number of goals that are randomly distributed in the environments to be 10, and agents' teams are allowed to search for goals over a lifetime of 500 time steps. **Figures 12–14** respectively, show the average number of goals found by teams of agents in the random, clumped, and linear environments. The x-axes indicate the size of the grids while the y-axes are for the average number of goals found by teams. We vary team size from 1 to 5 members, and vary the grid size from 12×12 to 27×27 two-dimensional grid.

The results we observe from the figures suggest that the performance of agents' teams in the three environments are similar and consistent for agents' teams, environment types, as well as for the various grid sizes. It is interesting to see that all teams of agents were able to find on average all the 10 goals when the size of the environments is 12. This is a 100% achievement by the agents' teams. However, the performance of the teams can be seen to degrade as the size of the environments becomes larger. This degradation is expected since the agents' team sizes remain the

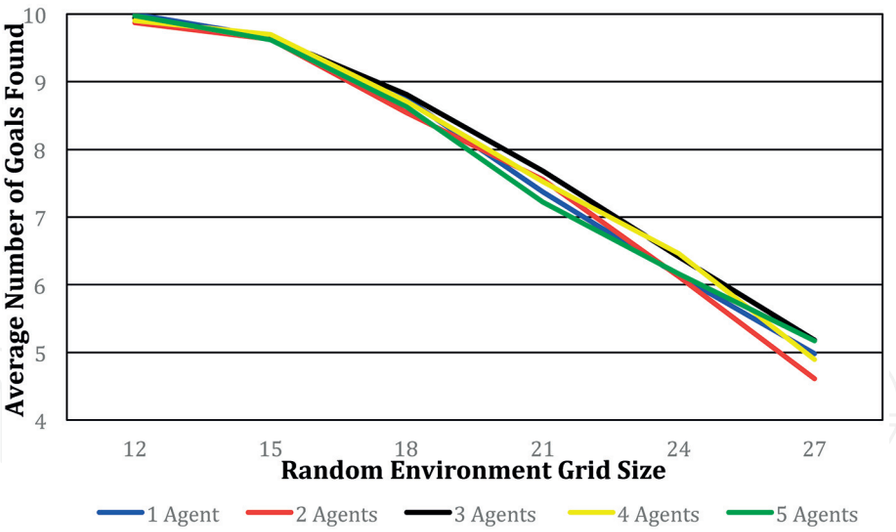

Figure 12.
Average number of goals found by agents' teams for different grid sizes in the random environment.

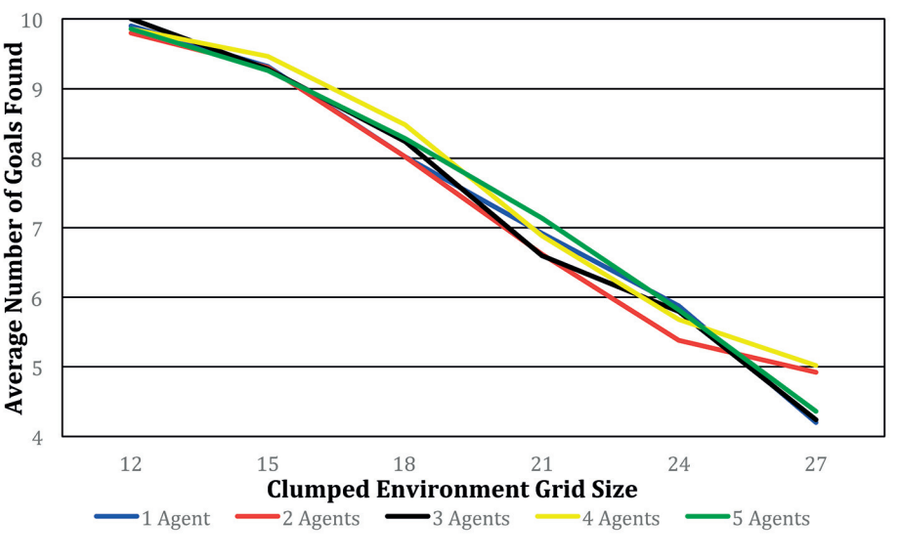

Figure 13.
Average number of goals found by agents' teams for different grid sizes in the clumped environment.

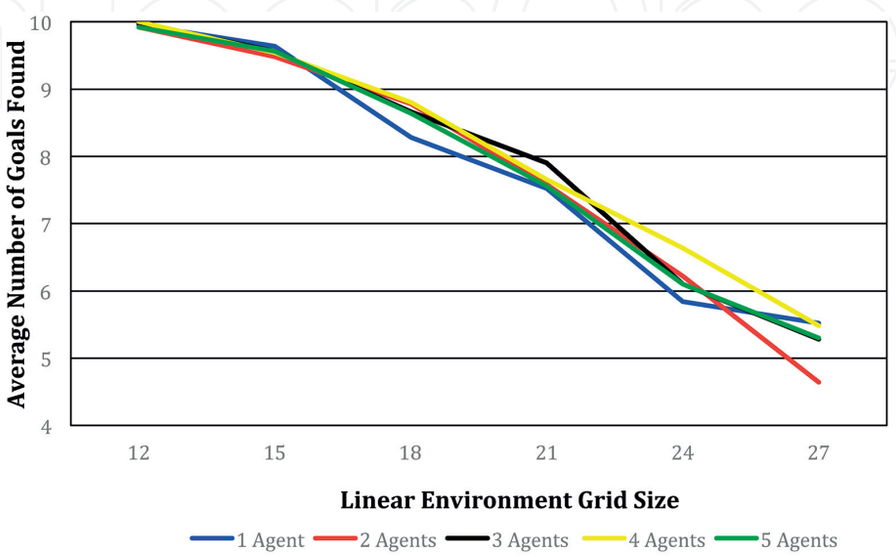

Figure 14.
Average number of goals found by agents' teams for different grid sizes in the linear environment.

same while the environments become larger. In a sense, the same number of agents need to do more work and cooperate in the larger environments. Again, these results are consistent with those of the previous section when agents are evaluated based on the percentage of interesting cells found in the environments. We note that the worst performance for all teams occurs when the size of the environments are of 27×27 dimensions. The clumped environment appears to be the most difficult in this case as all teams find less than five goals on the average.

We are interested in the lessons to be learnt from these experiments, as well as the implications of the similarity and consistency of the results across the three environments. The outcomes from these experiments suggest that our proposed SLS method is independent of the three environments, thus, agents using our search procedures are expected to perform about the same in any of the environments. This is also a confirmation of the results for agents' teams in the three environments when they are expected to find as many interesting cells from the previous section.

5. Conclusions and future work

We consider an investigation of the use of Stochastic Local Search (SLS) technique to explore complex environments where agents' knowledge and the time to explore such environments are limited. We model the problem as that of an instance of a search problem and develop a SLS technique that enables efficient exploration of such relatively difficult environments by teams of agents. Thus, we provide extensions to the work of Soule and Heckendorn [12] that uses evolutionary algorithms in evolving multiagent teams in the three different simulated random, clumped, and linear environments described in their work. We first provide a formalization of the concept of state and neighborhood in these environments and provide agents' performance evaluations in the three environments using the number of interesting cells found by teams as done in [12] and our previous work [13]. We further modify the environments to include a limited number of goals that are randomly distributed among the interesting cells. Agents in this case are thus required to search for goals rather than interesting cells.

Experiments using agents' teams of different sizes implementing our model in different problem environments show the effectiveness of our technique. In most cases of the problem instances, teams of agents were able to complete exploration of *more than* 70% of the environments. While in the *best cases*, they were able to complete explorations of *more than 80%* of the environments within short period of time. These results compare with those of Soule and Heckendorn's with performances within similar ranges. However, it is not yet clear how fair that comparison can be justified since their work employs evolutionary algorithms which come with extensive time requirements of evolutionary learning and huge time and costs of training for agents before they are deployed to actual testing environments. Our model avoids such expensive cost of extensive time requirements of evolutionary learning by agents. This is made possible as agents in our model are not subjected to training before being deployed to the testing environments. They only conduct local searches of the environments from their current locations using two important features from SLS algorithms: neighborhood and evaluation function.

We also evaluate the performance of agents' teams in another set of experiments requiring agents to search for goals in the three environments. It is interesting to note that all teams of agents were able to find on average all the goals in the three environments when the size of the grid is 12 . This is a 100% achievement by the agents' teams. However, the performance of the teams can be seen to degrade as the size of the environments becomes larger. A partial explanation for this is that since

the number of agents stays the same, they need to do more work in the larger environments. These results are consistent with those based on the evaluation of agents' teams on interesting cells found in the environments. The results of these experiments suggest that the level of difficulty of the three environments are relatively the same when agents' teams implement the SLS model. This is evidenced by the closeness of the teams' average performances in the two simulations. Thus, unlike Soule and Heckendorn's evolutionary model, the SLS model's performance has *less reliance* on the three environments.

There are several areas of ongoing research on this problem. Here are some directions for future work. A drawback of Soule and Heckendorn's model is the unlimited vision of the environments by all agents in their work. We avoid this problem by ensuring that agents in our model have only limited vision of the environments except the team leader that still has unlimited vision of the environments. We plan to address this issue in future work. Our proposed SLS model in this work still has some limitations. The search approach by the model is uninformed, thus, agents exhaustively search all states by slowly branching out of their neighborhood from their starting locations. We have commenced improvement of our model by allowing agents to do more informed search of the environments by using cleverly designed and admissible heuristics to guide the search. The expectation is that agents will now have direction of the search towards the goals in the environments at every new time step.

Author details

Ramoni O. Lasisi* and Robert DuPont
Department of Computer and Information Sciences, Virginia Military Institute, United States of America

*Address all correspondence to: lasisiro@vmi.edu

References

[1] Griffiths N, Luck M. Coalition formation through motivation and trust. In: International Conference on Autonomous Agents and Multiagent Systems; Melbourne, Australia. 2003

[2] Chalkiadakis G, Elkind E, Wooldridge M. Computational Aspects of Cooperative Game Theory. California, USA: Morgan & Claypool Publishers; 2011

[3] Batalin MA, Sukhatme GS. Efficient exploration without localization. In: International Conference on Robotics and Automation; Taipei, Taiwan. 2003. pp. 2714-2719

[4] Macedo L, Cardoso A. Exploration of unknown environments with motivational agents. In: 3rd International Conference on Autonomous Agents and Multi Agent System; NYC, USA. 2004

[5] Thomason R, Heckendorn RB, Soule T. Training time and team composition robustness in evolved multi-agent systems. In: O'Neill M et al., editors. EuroGP 2008. Vol. 4971. Berlin, Heidelberg: Springer. 2008. pp. 1-12

[6] Hollinger G, Singh S, Kehagias A. Efficient, guaranteed search with multi-agent teams. In: Robotics: Science and Systems. 2009. pp. 265-272

[7] Ray DN, Majumder S, Mukhopadhyay S. A behavior-based approach for multi-agent q-learning for autonomous exploration. International Journal of Innovative Technology and Creative Engineering. 2011;**1**(7):1-15

[8] Rochlin I, Aumann Y, Sarne D, Golosman L. Efficiency fairness in team search with self-interested agents. In: 13th International Conference on Autonomous Agents and Multiagent Systems; Paris, France. 2014

[9] Okimoto T, Ribeiro T, Bouchabou D, Inoue K. Mission oriented robust multi-team formation and its application to robot rescue simulation. In: 25th International Joint Conference on Artificial Intelligence; New York City, USA. 2016

[10] Hoos HH, Stutzle T. Stochastic Local Search: Foundations and Applications. San Francisco, CA, USA: Morgan Kaufmann; 2005

[11] Neller TW. Teaching stochastic local search. In: Proceedings 19th International FLAIRS Conference on Artificial Intelligence. American Association for Artificial Intelligence. 2005:8-14

[12] Soule T, Heckendorn RB. Environmental robustness in multiagent teams. In: Genetic and Evolutionary Computation Conference; Montreal, Quebec, Canada. 2009

[13] Lasisi RO. Efficient exploration of environments using stochastic local search. In: 9th International Conference on Agents and Artificial Intelligence (ICAART 2017); Porto, Portugal. 2017. pp. 244-251

[14] Hoos HH, Stutzle T. Local search algorithms for SAT: An empirical evaluation. Journal of Automated Reasoning. 2000;**24**:421-481

[15] Russell S, Norvig P. Artificial Intelligence: A Mordern Approach. 3rd ed. New Jersey, USA: Prentice Hall; 2010