The Cooperative Behavior Of A Human Work Group: A Distributed Learning Approach

Tapesh Santra
Department of Electrical Engineering
Indian Institute of Technology Kanpur
Kanpur-208016
Email: tapesh@iitk.ac.in

K.S. Venkatesh
Department Of Electrical Engineering
Indian Institute Of Technology Kanpur
Kanpur-208016
Email: venkats@iitk.ac.in

Amitabha Mukerjee
Department Of Computer Science
and Engineering
Indian Institute Of Technology Kanpur
Kanpur - 208016
Email: amit@iitk.ac.in

***Abstract*— Self managing systems are collective systems capable of accomplishing difficult task in dynamic and varied environment without any guidance and control. Some particular examples of this kind of systems are swarms and self organizing maps. These algorithms are being used to solve a great deal of complex problems like clustering, classification, optimization etc. In this paper we have tried to develop an algorithm to solve such problems easily and in a computationally efficient way. Our algorithm is based on a simulation of the cooperative behavior of a human work group. Like other multi agent systems we also have multiple agents each of which are connected to its topological neighbors. But instead of using any existing popular learning rules like competitive learning etc. here we have developed a simple decision theory based learning process in which each agent can communicate to its neighbors in order to help them to make an useful decision. This paper also contains some experimental results that we have achieved when we applied our algorithm to some image processing applications. Later we have discussed the scope of our algorithm in many different areas of computation.**

I. Introduction

Meta heuristics are high level strategies which guide an underlying subordinate heuristic to efficiently produce high quality solutions and increase their performance. These heuristics which were commonly referred to as modern heuristics attempt to combine intelligently different concepts for exploring and exploiting the search space in order to find near optimal solutions. There are several flavors of Meta-heuristics. Some are intelligent extensions of local search that attempt to avoid getting trapped in local minima by exploring the solution space. Examples of such meta-heuristics include Tabu Search [1], GRASP [2] and Simulated Annealing [3]. Others, such as Ant Colony Optimization [8] and Genetic Algorithms [4] are population based and tend to identify high quality areas in the search space by sampling or recombination respectively. Some other self managing systems are self organizing maps [10] which follow competitive learning algorithms like hebbian learning algorithm [9].

Our algorithm can be seen as a hybridization and modification of some of the popular self managing systems. We call it hybridization and modification because we have taken the idea of neighborhood connection from self organizing maps like Growing Neural Gas [11, 12, 13] and we have used the idea of cooperation from ant colony optimization. However the modification lies in the fact that we did not use the approach of pheromen information as in ant colony optimization techniques. Rather we have used a direct information exchange scheme among the neighbors. The main inspiration behind this work is the cooperating behavior of human beings in order to get rid of a complex situation. While working in a group human beings use to communicate to their neighboring people to inform them about their current situation and also collect information from their neighbors. Then from all the available informations the individual decides his or her best possible move. We used this fact in our algorithm to solve some engineering problems and studied the results in order to compare its efficiency with other algorithms.

II. MOTIVATION

This algorithm is motivated by the behavior of a group of people searching for some object of interest in a completely unknown search space. Let us consider the scenario of a large forest area with a harmful creature hiding somewhere in it. Suppose a group of people wishes to discover its whereabouts. Since nobody knows the exact location of the creature, the entire area of the forest needs to be thoroughly searched. In order to accomplish this, they will first divide themselves into smaller search parties, and allocate a separate region of the forest to each party, taking care to cover every part of the forest. Each party will search for the creature in the region around its own location. Once one of the parties finds some trace of the beast, it informs all parties nearest to itself. Then, in turn,these secondary parties inform yet others - tertiary parties -that are in their vicinity, as well as proceed to move towards the primary party in order to help it.

Next, suppose there is more than one harmful creature hiding in the jungle, each in a different place, In the process of the search, suppose two of the parties find two of them. Suppose a third party, located in the closeneighborhood of both of them, has got information from both of the parties. This party has now to decide which way to move. It will decide its move according to the relative urgency at each of the two places and the relative distances it will need to travel to catch up with either party. This situation highlights the increased complexity of the problem when multiple calls arrive simultaneously from different locations. Even with the simplifying assumption that a party can communicate only with its immediate neighbors, this possibility of multiple simultaneous calls, and the resulting conflicting demands for action needs to be addressed.

In this paper, we simulate this behavior of a group to find out the state of the overall system at different time instants and find some surprising and interesting results. We find that this behavior of a group may yield a good optimal solution in reasonably less time for especially those problems where the search space is completely unknown. In the following sections we shall go through this simulation and its results and also describe some of its applications.

III. PROPOSED ALGORITHM

Our algorithm consists of a population of agents, each member of which is connected to, and only to, its topological neighbors. These agents explore the search space for a near optimal solution to a given problem. While thus searching, they may continuously communicate with their neighbors to inform the neighbors about the attractiveness of their current locations, as well as to let those neighbors reciprocate. From the information gathered about its own, as well as its neighbors' locations, an agent will decide to move in a certain direction. According to the nature of the problem on which our algorithm is applied, our algorithm may have several variants. This paper will concentrate mainly on two of the most important variants. The variations are essentially introduced in the exact procedure by which an agent determines the attractiveness of its current location. In some cases an agent can determine the attractiveness of its current location by knowing only the coordinate of its location and its own region. In other cases, it may have to examine its neighboring regions also to estimate the attractiveness of its current location. As the attractiveness estimation procedure is one of the most fundamental concerns of our algorithm, we shall now discuss this matter in some more detail.

We may begin by supposing the simplest possible form for the attractiveness estimation function. If we consider an m dimensional search space, $\mathcal{S}$, a point in it may be addressed as $\mathbf{x} = (x_1, x_2, \ldots, x_m)$. Let the collection $f_p(\mathbf{x}); p = 1, \ldots, P$ be a set of functions defined on $\mathcal{S}$. We shall say that the attractiveness $A(\mathbf{x})$ is in nature if there exists F_A such that $A(\mathbf{x}) = F_A(f_1(\mathbf{x}), \ldots, \mathbf{f_P}(\mathbf{x}))$

We next consider a rather more complex formulation for the attractiveness A. We shall say A is *nonlocal* if there does not exist an F_A such that A may be deemed local in the sense described above, but there does exist an F_A such that $A(\mathbf{x}) = F_A(f_1(\mathbf{x}'), \ldots, \mathbf{f_P}(\mathbf{x}')); \mathbf{x}' \in \mathcal{R}(\mathbf{x})$, where $\mathcal{R}(\mathbf{x}) \subseteq \mathcal{S}; \mathbf{x} \in \mathcal{S}$. As a concrete instance of nonlocal attractiveness, consider $A(\mathbf{x}) = \iint_{\mathcal{R}(\mathbf{x})} f_1(\mathbf{x}') d\mathbf{x}'$ where we have taken $P = 1$. A nonlocal attractiveness is more general, but obviously claims more computation than the local counterpart.

We shall contend later that we may replace the exhaustive evaluation of $A(\mathbf{x})$ by using attribute information over each point in $\mathcal{R}(\mathbf{x})$ with an evaluation over a sparse but representative subset consisting of only a smaller number of randomly chosen points within $\mathcal{R}(\mathbf{x})$. An example of this kind of evaluationis shown in the sequence of figures Fig.2, Fig.3 and Fig.4. Fig.2 shows the sample data set. Fig.3 shows the attribute function f_1 and Fig.4 shows the sparsely evaluated attractiveness function A.

We shall now discuss our algorithm in the context of the two variants of attractiveness estimation procedure given above. Our proposed algorithm has some common steps for every variant.

1) Initialization: In this step the population of agents is built and their initial locations are defined.
2) Attractiveness Estimation: During this process each agent use to estimate the attractiveness of their current locations.
3) Communication: During this process each agent communicates to its topological neighbors to inform about the attractiveness of their current location.
4) Decision making and random walk: During this process each agent decides about their next movement and moves accordingly.
5) Convergence: After each iteration the convergence criteria is checked and the whole process is repeated from step 2 if the algorithm is not converged yet.

These steps are illustrated in figure 5.

These are the basic steps of our algorithm while there may be minor variations of these steps for different kinds of problems. The overall structure of the proposed algorithm would remain largely unchanged. Each of these steps is discussed in detail in the following subsections.

A. Initialization

We have a set of K agents $\Lambda = \{a_i : i = 0, 1, \ldots K-1\}$, which, to begin with, are distributed uniformly over the whole search space. Each agent a_i is connected to its topological neighbors, constituting the set $N_i \subset \Lambda$. We present our formulation as follows.

$$\begin{aligned}
&\Lambda = \{a_i : i = 0, 1, ...K-1\} \\
&a_i = \langle X^{\mathrm{t}}_{\mathrm{a_i}}, D^t_i, N_i \rangle \\
&D^t_i = \{d^t_j : j = 0, 1, ...n_i - 1\} \\
&d^t_j = \langle X^t_j, r^t_j, p^t_j \rangle \\
&X^t_j = \{x^t_1, x^t_2, ...x^t_m\}, m > 0 \\
&N_i \subset \Lambda
\end{aligned} \tag{1}$$

X^t_i is the current location of a_i in the search space and D^t_i is the set of the informations currently in its possession. This set of informations includes all the informations it has received from neighboring agents in N_i and the information it discovers by itself. At $t = t_0$, the starting point of the algorithm, evidently $D^{t_0}_i = \emptyset$, because at the start, no agent has yet coommunicated any attractive location in the search space to a_i. We call each member d^t_j of D^t_i a *decision metric*, and the whole set D^t_i a *decision set.*

Each decision metric d^t_j will have three elements, a location X^t_j in the search space, a reward r^t_j and a punishment p^t_j associated with the location X^t_j. These $X^t_j; j = 0, \ldots, n_i - 1$

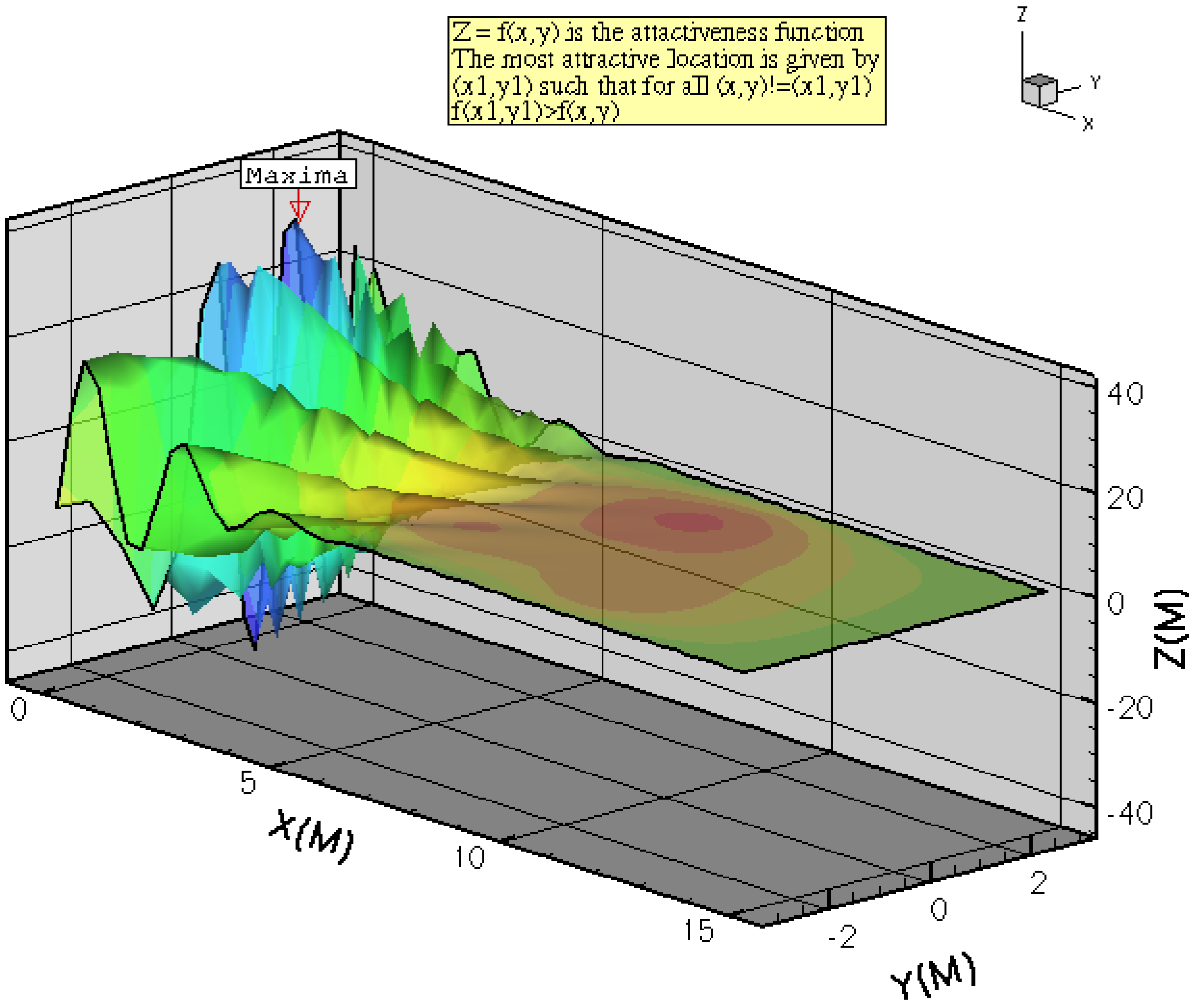
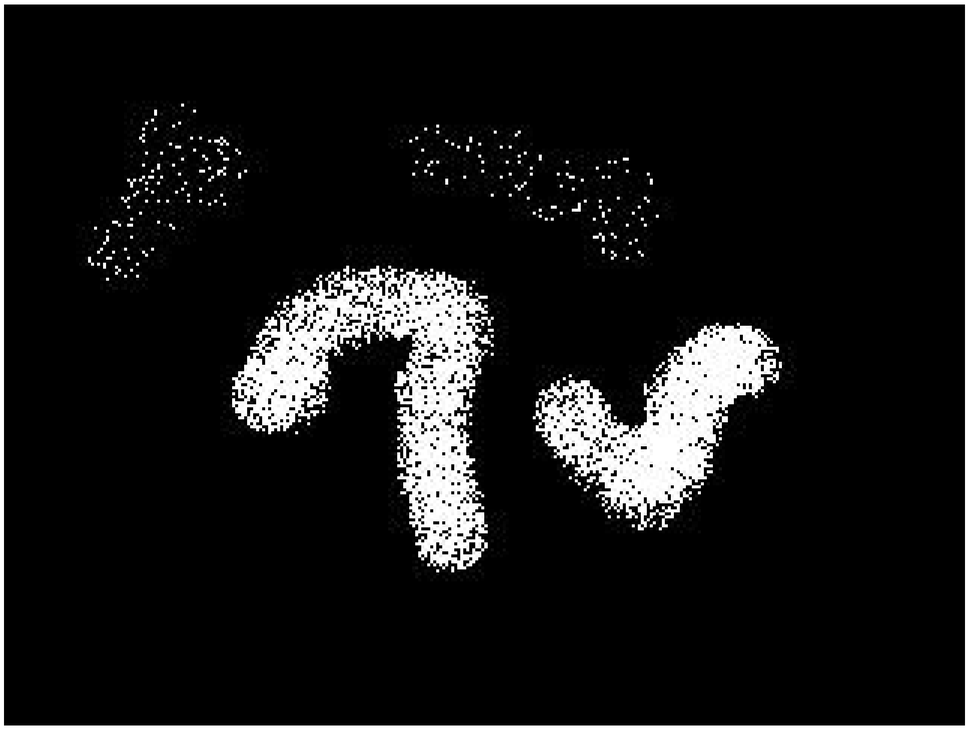


Fig. 1. In this figure we have represented an example search space of two dimensions X and Y. The search space is defined as $0 \leq x \leq 15, -2 \leq y \leq 2$. The Z dimension is showing the attractiveness values. Our intention is to find out the maxima in a heuristic way. Here attractiveness is direct function of X and Y.

are nothing but the locations of the neighbors of a_i who have reported some attractive areas in their local search spaces. If a_i itself discovers some attractive location, then its own metric d_i^t will also be included in the decision set D_i: as $a_i \in N_i$. The cardinality of D_i^t, n_i satisfies $n_i \leq |N_i|$. However we have no need for a superscript t for the neighbor set N_i of any a_i because the set of neighbors is preassigned and does not change with time. The assumption of a non-dynamic neighbor set may seem to deviate from the basic model of comunication between individuals in a human work group, but we shall justify this by some experiments later. In Fig.6(a), we see there are some white pixels scattered amongst black pixels. In some regions, the white pixels are seen to be much more densely clustered than elsewhere. Our goal is to find out the region with high density of white pixels. We experiment with both static neighbor sets and dynamically changing neighbor sets and present the results in Fig.6(b) and Fig.6(c) respectively. The rules for changing neighbors in the dynamic neighbor set case are stated below.

1) Let a_i have a neighbor set N_i^t at time instant t and $N_i^t = \{a_j : j = 0, 1, ...|N_i^t| - 1\}$. At time instant t a_j has a home location X_j. In each iteration, the agent a_i will calculate the distances $\mathrm{dist}_j = |X_i - X_j| : j = 0, 1, ...|N_i^t| - 1$ if $|N_i^t| > 1$. If for any$j$ $\mathrm{dist}_j > \mathrm{Th}_i^t$, where Th_i^t is the upper bound of the distance upto which the agent a_i wishes to communicate at the time instant t, then we shall delete the corresponding agent a_j from the neighbor set of the agent a_i. We should keep in mind the fact that this deletion occurs only when $|N_i^t| > 1$
2) Here we calculate the threshold dynamically. The threshold could be different in general for each agent a_i at every time instant t. We have calculated the threshold



(a)

Fig. 2. A sample data set.

in the following way.

We shall make the reasonable assumption that a person is more likely to communicate with the person(s) who is/are closest to him. According to this assumption, we first determine a weighting factor for each neighbor. The weighting factor $w_j = (1/(|N_i^t|-1))(1-\frac{\text{dist}_j}{\sum_{k=0}^{|N_i^t|-1}\text{dist}_k})$. The threshold $Th_i^t = \sum_{j=0}^{|N_i^t|-1} \text{dist}_j * w_j$. The threshold, instead of using a uniform average uses a weighted average.

3) After deleting the distant neighbors an agent a_i will calculate the distances $\text{dist}_k = |X_i - X_k| : k = 0, 1 \ldots, K-1$ and $a_k \notin N_i^t$. If for any k $\text{dist}_k < \text{Th}_i^t$ then the agent a_i will consider a_k as its neighbor at $t+1$.

From the results it is clear that the population with static neighbor sets yields a better result. We can see from Fig.6(c) that some of the agents have converged to a local maxima rather than the global maxima. This is because after some iterations they have lost communication with other agents and formed small groups of agents to converge at local maximas. There is another shortcoming in dynamic neighbor sets: it raises the computational burden on the whole process. So we shall use static neighbor sets instead of dynamic sets.

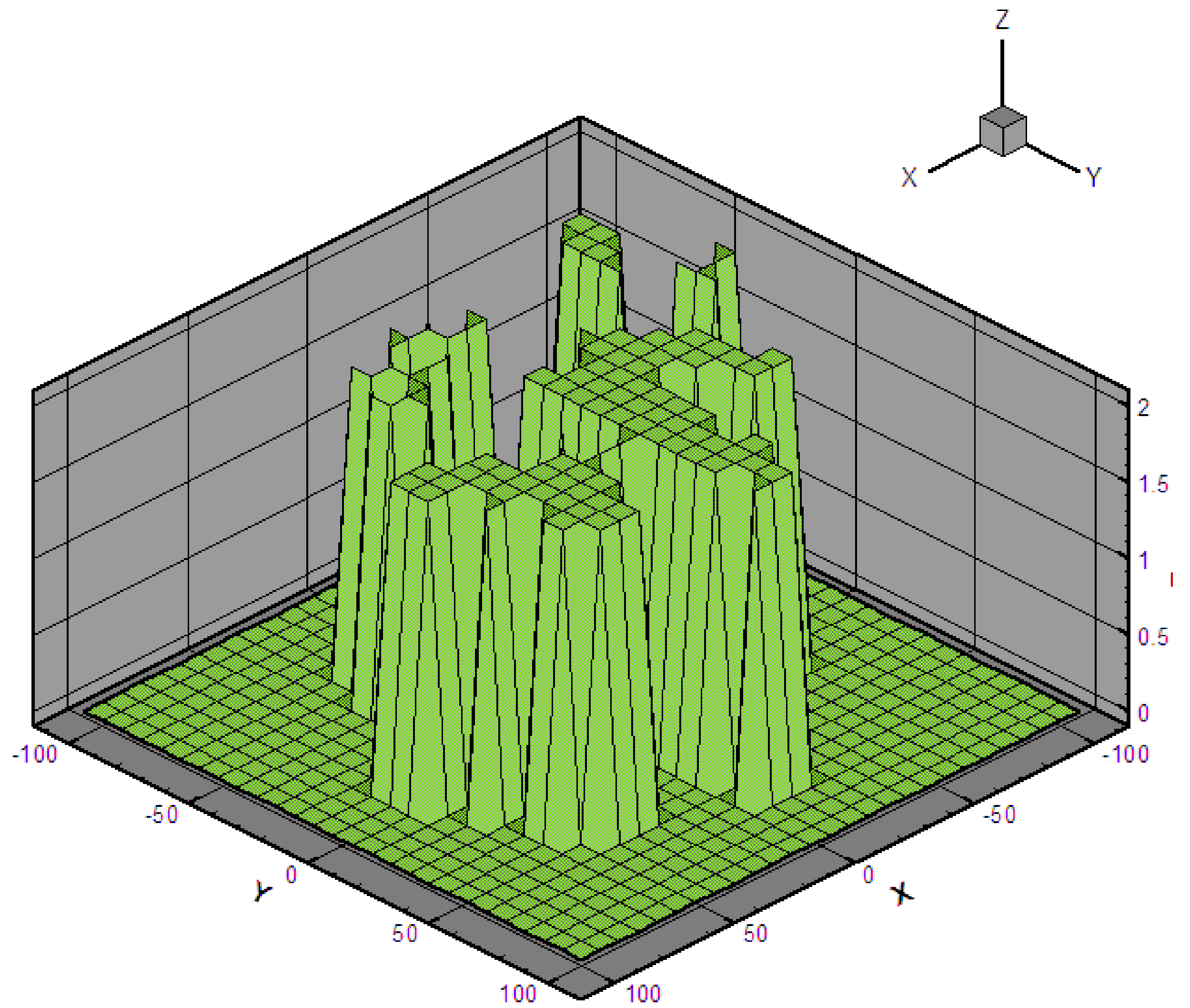
After assigning its neighbor set to each agent a_i, we next also assign their initial home locations. To get better results, we uniformly distribute the agents to cover the whole of $\mathcal{S}$. Initially, we partition the $m > 0$ dimensional search space $\mathcal{S}$ into K uniform regions. Each of the K agents will occupy the centre of one of these regions to begin with.

$$\begin{aligned} &\mathcal{S} = \{S_i^{\mathrm{t_0}} : i = 0, 1, ..., K-1\} \\ &C\{S_i^{t_0}\} = L\{a_i\}^{\mathrm{t_0}} = X_{\mathrm{a_i}}^{\mathrm{t_0}}, a_i \in \Lambda \end{aligned} \tag{2}$$

Here the center of the i^{th} region at the time instant t_0 is denoted by $C\{S_i^{t_0}\}$ which is nothing but the location of the ith agent a_i at the time instant t_0. In equation (2) $L\{a_i\}$ denotes the location of a_i and t_0 is the initial time instant of the algorithm i.e. the algorithm starts at t_0. As we are considering that initially $S_i^{\mathrm{t_0}}$ are nonoverlapping regions, we may define the global search space as follows.

$$\mathcal{S} = \bigcup_{i=0}^{i=K-1} S_i^{\mathrm{t_0}} \tag{3}$$

For problems where a nonlocal attractiveness has been specified, we would need a local search to compute the attractiveness $A(\mathbf{X_i^{t_0}})$ of the home location of each agent. We thus need to define the initial local search region $\mathcal{R}(\mathbf{X_i^{t_0}})$ for each agent. We initially assign a radius τ_i to each agent a_i and this radius can be simply chosen to be the radius of the region of the partition alloocated initially to a_i, the center of which is $\mathbf{X_i^{t_0}}$. If the radius of the ith region is π_i, we may then say that the radius of local search region of a_i is $\tau_i = \pi_i$. But it is



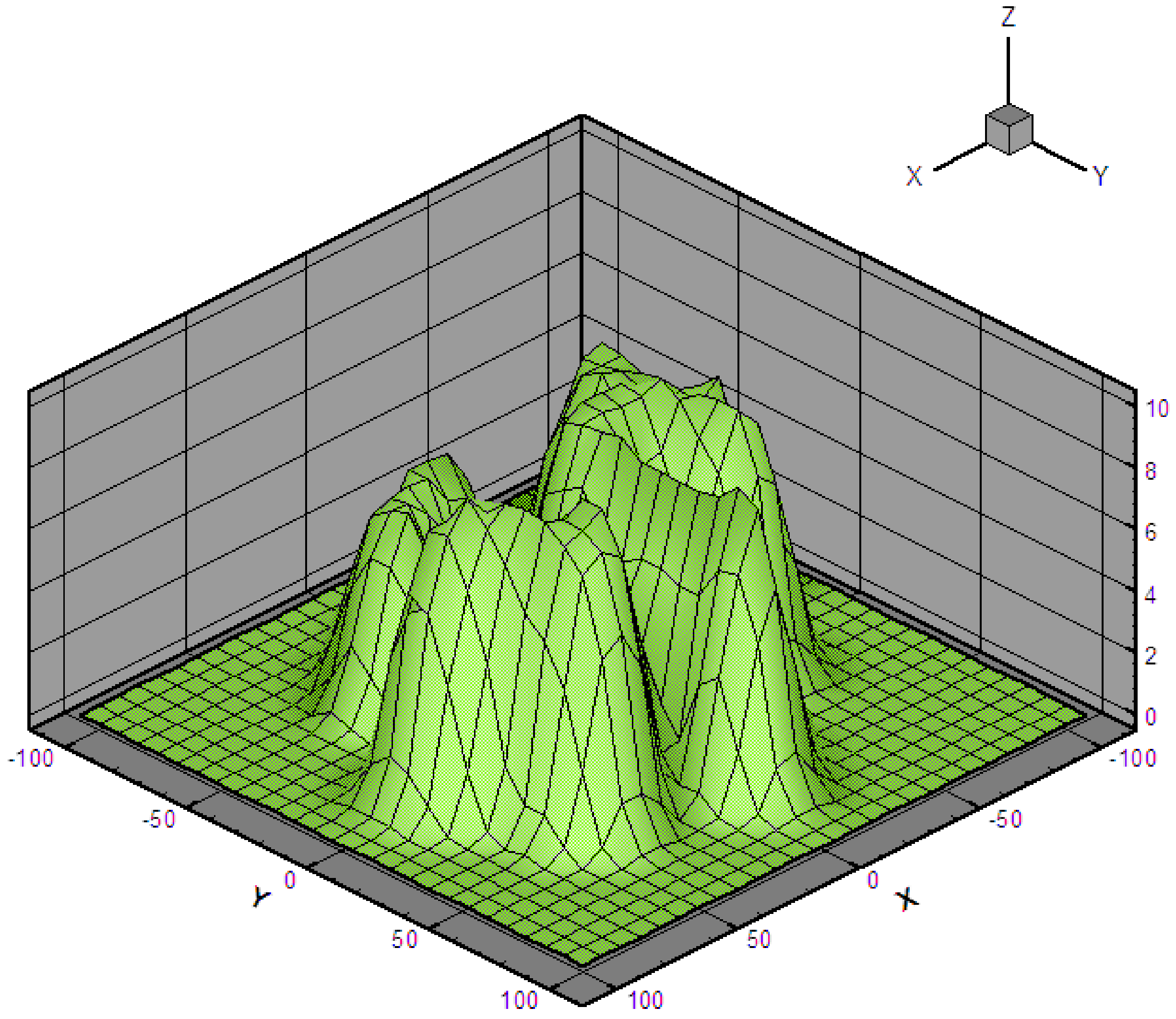
(a)

Fig. 3. The score function of figure 2. Though, to determine the attractiveness we don't have to find out the scores at each location of the image, we have done so and shown in this picture to visualize the output of the score function.Here the 'Z' coordinate represents the score values.

not mandatory. If $\mathcal{S}$ is very large, then we may choose to take $\tau_i < \pi_i$ or even $\tau_i << \pi_i$. If we use $\tau_i < \pi_i$ then it's better to go through a fresh local search at each iteration. Otherwise the agents could miss some important information. However if $\tau_i = \pi_i$ at $t = t_0$ then we can avoid local search at each iteration in favour of some simple estimation method. This procedure will be discussed later in this section. The novelty of this algorithm will become apparent when we use $\tau_i << \pi_i$. In that case, some of the attractive locations remain unrevealed even after some initial iterations. But they will be revealed in later iterations. So as time goes on, the search space will be more and more fully explored by the agents and they will try to converge to the optimal solution for the given problem. But for the sake of clarity, in our paper we will consider $\tau_i = \pi_i$.

B. Attractiveness Estimation

Whenever the specification of the attractiveness is nonlocal, we need to go through a local search to estimate the attractiveness at any location $\mathbf{x}$. For different problems, this local search procedure will be different. But we shall present a general approach in this subsection. At the first iteration, all agents will make random searches for an attractive location in their respective local search regions. Attractiveness can be estimated from the reward of the search location. During this search operation, the location of the agent will not be changed until some attractive point is found. If any point seems to be attractive the agent will adapt its location towards that point by a small amount. Once one agent finds an attractive point, it's local search region will no longer necessarily remain non



(a) The attractiveness function of the subsampled image

Fig. 4. We have first defined a $5X5$ window around the pixel of interest to pick up 10 pixel locations randomly from within the window and summed them up to determine the attractiveness of the pixel location. However to display the attractiveness values in a convenient way we have calculated the attractiveness at each location and then displayed a subsampled attractiveness graph. Here the Z coordinate represents the attractiveness values.

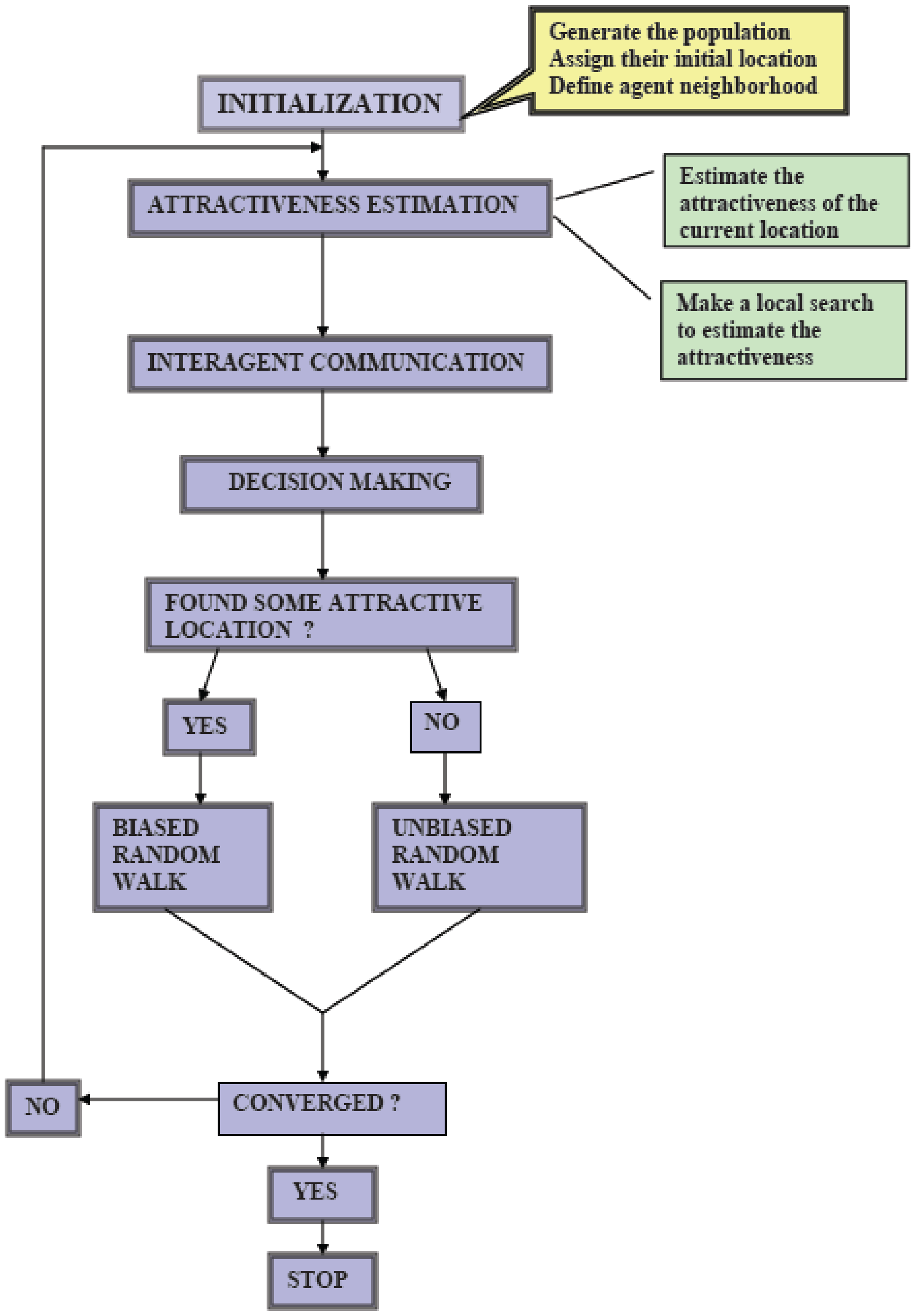
overlapping with the search regions of other agents. For every agent that finds attractive points, the home location will be changed according to the adaptation rule described in equation 4. This adaptation of home locations will take place each time they find an attractive point within their current local search region. The total portion of points in each local search region that are randomly explored in each iteration will be a very very small fraction of the total volume of its local search region. The location adaptation rule is described below.

$$X_i^{t_0} = X_i^{t_0} + \delta * (X_i^{\mathrm{k}} - X_i^{t_0}) \\ \delta = \alpha * r(X_i^{\mathrm{k}}), 0 < \alpha \leq 1. \tag{4}$$

Here δ is the rate of adaptation and and it is proportional to the normalized reward of the point X_i^{k} which is the k^{th} attractive location for i th agent. $X_{\mathrm{a_i}}^{\mathrm{t}}$ is the current location of the agent a_i as said before.

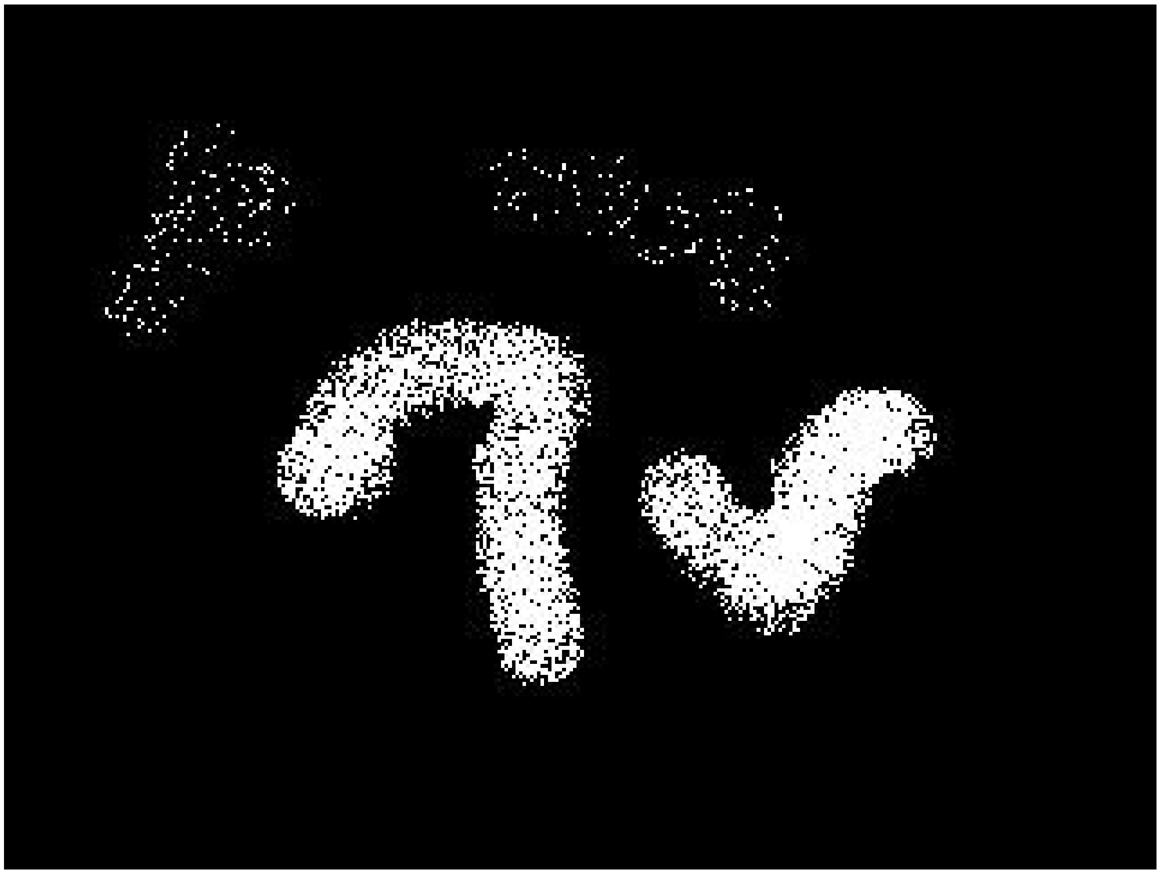
After each agent completes its initial local search those who have found some attractive locations will move to a new home location (the others will retain their current home locations). They will then update their their own decision sets D_i as well as transmit the update to their neighbors so that they can also update their own decision sets D_j.

Figure 7 shows how the decision set will look like after the information exchange. Due to lack of space we only have shown the decision sets of first, second, third and fifth agent. The decision sets of the other agents will follow the same

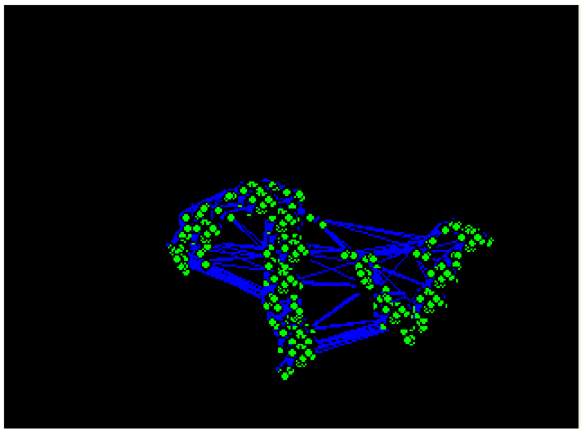

Generate the population
Assign their initial location
Define agent neighborhood
INITIALIZATION
ATTRACTIVENESS ESTIMATION
Estimate the attractiveness of the current location
Make a local search to estimate the attractiveness
INTERAGENT COMMUNICATION
DECISION MAKING
FOUND SOME ATTRACTIVE LOCATION ?
YES
NO
BIASED RANDOM WALK
UNBIASED RANDOM WALK
NO
CONVERGED ?
YES
STOP

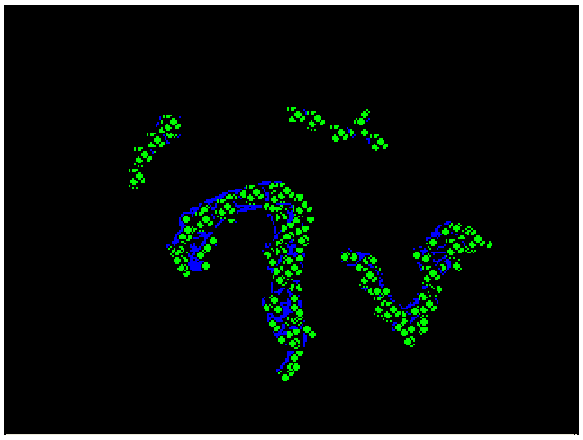
(a)

Fig. 5. In this figure we have shown a flowchart for our algorithm.



(a) Test data for dynamic population set





(b) Result with static neighbor sets

(c) Result with dynamic neighbor set

Fig. 6. (a) shows the image on which we applied our algorithm with static and dynamic neighbor sets. Figure (b) shows the result with static neighbor sets and figure (c) shows the result with dynamic data sets. We can see a clustering of agents at local maximas in figure (c).

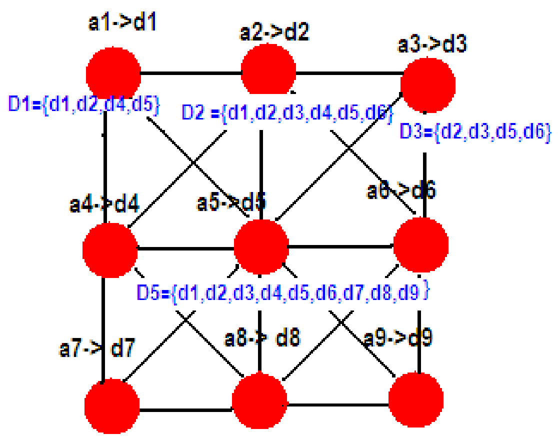


Fig. 7. In this figure we have represented nine agents as nine red circles. The interconnection between them are shown as black lines. Here, at any time instant $t > t_0$ the agents a1 to a9 found decision metrics d1 to d9. They added their corresponding decision as well as the decision found by their neighbors to their decision set. We have shown how the decision matrix of a1, a2, a3, a5 will look like. These decision matrix are represented as D1, D2, D3, D5. Others decision matrix are not shown due to lack of space but they will follow the same rule.

rule. Here we have considered that all agents have found some attractive locations, which is not a necessary state of affairs on every occasion. We shall discuss some aspects of this inter agent communication scheme in the next subsection.

At this point the first iteration is complete. If, at the first iteration, some of the agents have found some attractive locations then those who have found attractive locations have already changed their locations towards the most attractive area in their initial local search space. So, accordingly, the centers of their local search spaces are also changed and hence the local search spaces no longer remain non overlapping to each other. This phenomena is shown in the following equation.

$$\mathcal{S} \supset \bigcup_{i=0}^{K-1} S_i^t, t > t_0 \tag{5}$$

Even after the information exchange some of the agents may have an empty decision set. Those who still have an empty decision set will have a small unbiased random walk and those who have a non empty decision set will have a biased random walk in the next iteration.

C. Interagent Communication

As we have stated earlier, each agent communicates to its neighbors after it finds some attractive location in order to inform their neighbor about the attractiveness of the location it has revealed. A question may arise at this point, as to why they should communicate to only their neighbors - and not to *every* other agent? To answer this question let us consider that we have 200 agents in some application searching for an optimal solution of a given problem. Each of these agents have 5 neighbors. Now if they had to comunicate with all the remaining agents then at each iteration each agent had to go through 199 computations in order to make the commuication only. After the communication is complete they need to take decisions. If all the agents communicate to all the remaining agents then the decision set will also be proportionately larger. This would put severe computational burden on the system. Rather if we take the scheme of neighborhood communication then at each iteration each agent has to go through 5 computations only in order to communicate and the decision set will also be small in size. So the neighborhood communication scheme is more realistic from the computational point of view.

After the communication is complete each agent has to decide about their future movements. This procedure is described in the following subsections.

D. Unbiased Random Walk

When an agent is attempting to make some decision it will search its own decision set for the available decision metrics at that time instant. But if it finds that its decision set is empty then it will execute a small unbiased random walk around its current location.

$$\begin{aligned} &X_{\mathrm{a_i}}^{\mathrm{t+1}} = X_{\mathrm{a_i}}^{\mathrm{t}} + \xi, t \geq t_0 \\ &\xi \text{ is a small random vector such that } \|\xi\| << \tau_i \\ &\tau_i \text{ is the radius of local search space of } i^{\text{th}} \text{ agent .} \end{aligned} \tag{6}$$

E. Biased Random Walk

Any agent which has a nonempty decision set will execute biased random walk biased by the probability factors associated with each available decision metric in their decision set. The agent a_i will calculate n_i probability factors P_j^t where $j = 0, 1, ...n_i - 1$ depending upon the rewards and punishments associated with each available decision in the following way.

$$\begin{aligned} e_j^t &= \frac{\mathcal{F}(r_j^t)}{\mathcal{G}(p_j^t)} \\ P_j^t &= \frac{e_j^t}{\sum_{k=0}^{n_i-1} e_k^t}, t > t_0, j = 0, 1, ...n_i - 1. \end{aligned} \tag{7}$$

Here $\mathcal{F}(r_j^t)$ and $\mathcal{G}(p_j^t)$ are the functions of reward and punishment of the j^{th} decision at the time instant $t > t_0$. After the calculations of the probability factor the agent a_i will take decision d_j with a probability P_j^t. If the agent a_i has chosen the decision d_j^t then it will update its location according to the following equation.

$$\begin{aligned} &X_{\mathrm{a_i}}^{\mathrm{t+1}} = X_{\mathrm{a_i}}^t + \beta(X_j - X_{\mathrm{a_i}}^t), t \geq t_0 \\ &\text{Generally } 0 < \beta \leq 1 \end{aligned} \tag{8}$$

Here β is the rate of adaptation. If we choose $0 < \beta << 1$, typically in the range $(0.001, 0.1)$, then the algorithm will yield better solution but at the same time it will take more time to converge. On the other hand if we take β in the range of $(0.5, 0.9)$ the convergence will be faster but the algorithm may not yield a good solution. There is a need to optimize the performance of the algorithm in terms of quality of the solution and the time taken to converge. This can be done by using Genetic Algorithm. However we did not use G.A. In our applications we have manually assigned β a value of 0.5.

In the equations above we have used a function $\mathcal{F}$ of reward and a function $\mathcal{G}$ of punishment. A question may arise regarding their exact forms. There are no generalized forms for these two functions. Their form will depend on the type of problem. But one thing must be remembered. Though it looks like the punishment factor is unnecessary and can be avoided this is not practically so. Because if we make the punishment factor a constant the algorithm may take too much time to converge. This punishment factors help the algorithm to converge quicker.

Here we shall discuss some examples of the forms these functions may take. Most oftenly the function $\mathcal{F}$ is a linear function of the reward r_j^t because attractiveness is the prime concern in most of the problems. So the function $\mathcal{F}(r_j^t) = \theta r_j^t + \omega$ where both θ and ω are constants. The value of these constants may vary from application to application. Even the values of these constants can be tuned by means of some other meta heuristic algorithms like Genetic Algorithm to get most effective results. In our applications we have simply taken $\mathcal{F}(r_j^t) = r_j^t$. Now the function $\mathcal{G}$ may have different forms. If it's as important as the attractiveness we may use a linear form of the kind we have used for $\mathcal{F}$. But in some cases we may have to prioritize the punishment factor against the reward factor. If the priority of punishment factor is less compared to the reward factor then it's better to use some nonlinear form for $\mathcal{G}$. For example let us define $\mathcal{G}(p_j^t) = \delta^{\mathrm{p_j^t}}$ where δ is a constant and $\delta = 1 + \nu$ and $0 < \nu << 1$, generally $0.001 < \nu < 0.01$. From this function it's clear that even for large variation of p_j^t the variation of $\mathcal{G}(p_j^t)$ will remain small. We have used this function in one of our applications discussed later.

F. Population Control

Let us consider that at any time instant 't', R $(R << K)$ number of agents have found some attractive locations. Now, suppose a total of M $(M \geq R)$ number of agents have got the informations about the R attractive locations. So at the instant t a total of M number of agents have to select their moves from 'R' available locations to decide their position at the $t + 1^{th}$ time instant. If $M >> R$ then some attractive locations may become crowdy due to over gathering of agents. This may lead to some undesired results because due to this overcrowding some attractive locations may remain unrevealed for ever. So to avoid this we have to take necessary measures.

Let's consider that at any time instant t, an agent a_i is about to make a move towards, say , attractive location X_j. So it will first see at the same time instant t how many agents have already made the same choice prior to itself. If the total number of such agents goes beyond the crowd factor υ then it will make the chosen move + a small random vector. Precisely,

$$X_{\mathrm{a_i}}^{t+1} = X_{\mathrm{a_i}}^{\mathrm{t}} + \beta * (X_j - X_{\mathrm{a_i}}^{\mathrm{t}}) + \xi, t \geq t_0$$
ξ is a small random number such that $\|\xi\| << \tau_i$,
Where τ_i is the radius of local search space of the i^{th} agent. (9)

To inmplement this population (density) control mechanism we maintain a global dynamic state vector. Each element of this vector will correspond to the current state of the corresponding agent. Let us denote this state vector by Υ. So Υ will have K elements. At t the ith element of Υ will correspond to the state of a_i. At the start of each iteration, the value of all the elements of this state vector will be reset to zero. Now suppose that at any t the first agent a_1 has decided to move towards the location of the agent a_2. We shall then increase the second element of Υ by 1 to indicate that at the location of a_2 has a population excess of one agent. Applying this procedure for every agent move in the current iteration, we shall be able to track the population at each step at each current home location. After finishing each iteration we shall again reset all the elements of the state vector to zero.

G. Convergence

Once all the agents have made their decisions they will start the next iteration with a local search and go through the whole procedure again. This is repeated until the convergence criterion occurs. We shall say our algorithm has converged when the change in average movements of the agents with respect to time is almost zero. However we cannot presently give a theoretical proof of the convergence of our algorithm. But we have observed that there is an important role of the punishment factor in the convergence of our algorithm. Without a punishment factor there is a high chance for the algorithm to oscillate for ever. This phenomena occurs when some of the agents got some equally attractive locations. Then they jump around from one location to another endlessly and the system starts oscillating. This can be avoided by introducing a proper punishment factor.

A local search in every iteration will produce heavy computational burden. In the next section we shall discuss about some techniques to avoid this computational burden.

H. A procedure to avoid local search at each iteration

At the very first iteration of our algorithm, we divided the whole global search space in K nonoverlapping regions S_i and assigned each region to a unique agent as its exclusive zone that contained initially(and even equalled, if $\tau_i = \pi_i$) its local search space. So, as we have covered the whole global search space in the first iteration, we will have a gross estimate of the most attractive locations after the first iteration or rather after the iteration when for the first time a considerable number of the agents have found some attractive locations. At the subsequent iterations the agents having non empty decision sets will move towards one of the attractive locations found in earlier iterations. So their modified location will be very near to one of the attractive locations found previously. Thus, rather than carrying out a thorough local search we can formulate an estimation procedure which will estimate the attractiveness of the modified locations. This estimation may be done by a function of distance of the modified locations from the previous locations of the topological neighbors as well as the previous location of the concerned agent itself. This procedure may be formulated mathematically in the following manner.

$$\begin{aligned}
&r(X^{\mathrm{t+1}}_{\mathrm{a_i}}) = \mathcal{E}(X^{\mathrm{t}}_{\mathrm{a_k}}, r(X^{\mathrm{t}}_{\mathrm{a_k}})), k = 0, ..n_i - 1.\\
&X^{\mathrm{t}}_{\mathrm{a_k}} = L(a_k) \text{ at time instant } t > t_0\\
&a_k \in N_i \cup a_i, N_i \text{ is the neighbor set of agent } a_i.\\
&r(X^{\mathrm{t+1}}_{\mathrm{a_i}}) \text{ is the estimated attractiveness}\\
&\ \text{of the new location of agent } a_i\\
&\mathcal{E} \text{ is the estimation function.}
\end{aligned} \tag{10}$$

Here $r(X^{\mathrm{t+1}}_{\mathrm{a_i}})$ is the attractiveness of the location of the i^{th} agent a_i at the time instant $t+1$. We know that $X^{\mathrm{t+1}}_{\mathrm{a_i}}$ will be somewhere near the home locations of the agents which are neighbors of the agent a_i or it will be very near to the home location of the agent a_i itself. So here we have defined the attractiveness of the new location $X^{t+1}_{\mathrm{a_i}}$ as a function of the attractiveness of its neighbors, attractiveness of itself, its neighbors locations and its own location. For the simplest possible approach let us consider that the attractiveness decays exponentially with distance. So if X_j is an attractive location and $r(X_j)$ is the attractiveness of the location X_j then we can say that at a location X the attractiveness will be estimated as $r(X_j)\exp(\frac{-\|X_j - X\|}{\nu})$. Using this technique we estimate the attractiveness of the location of the agent a_i at the time instant $t+1$ as follows.

$$\begin{aligned}
&r(X^{\mathrm{t+1}}_{\mathrm{a_i}}) = \sum_{k=0}^{n_i} \kappa_k r(X^t_{\mathrm{a_k}}) \exp(\frac{-\|X^t_{\mathrm{ak}} - X^{t+1}_{\mathrm{a_i}}\|}{\nu})\\
&a_k \in N_i \cup a_i, \kappa_k \text{ is a constant such that } 0 < \kappa_k < 1.\\
&\nu \text{ is a constant such that } \nu > 1.
\end{aligned} \tag{11}$$

IV. Some Simple Applications Of Our Algorithm

We applied our algorithm to some simple image processing applications. In this section we will discuss some of the simplest applications and analyze their results.

A. *Finding High Density Clusters Of White Pixels*

In Fig. 8 we have a binary image which contains some white pixels as well as some black pixels. Our objective is to find out the areas with high density of white pixels. For this we set up a population of agents and leave them on their own to find out the region where the density of white pixels is considerably high. We formulate the whole problem in the following way.

The reward is the probability of white pixels at the neighboring region of the location of interest. So the attractiveness of the point $X_{\mathrm{a_i}}$ will be $\frac{n^w_i}{n^t_i}$ where n^w_i is the total number of white pixels and n^t_i is the total number of pixels picked up randomly from the local search region centered at $X_{\mathrm{a_i}}$. The punishment factor is the distance to be travelled to reach

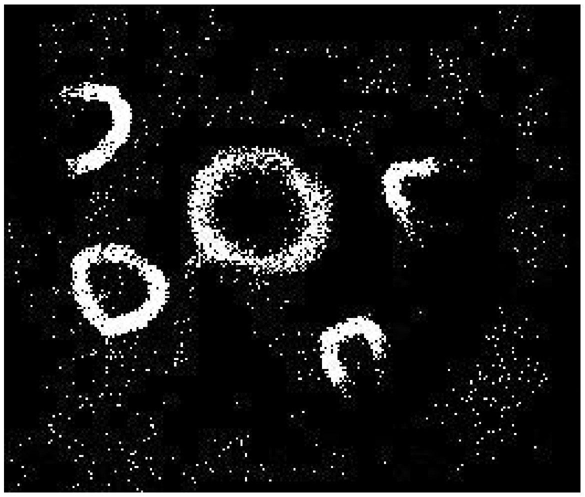


Fig. 8. In this figure we can see some white pixels are spread over a black background. In some areas those white pixels are forming much denser clusters than the remaining area. Our goal is to find out the denser clusters in the image in a completely heuristic way.

the attractive location divided by some constant factor. The formulations are shown in the following way.

1) First we generate a set of population and then we populate the neighbor set of each agents.

$$\begin{aligned}
&\Lambda = \{a_i : i = 0, 1, ...K - 1\}\\
&a_i = \langle X^{\mathrm{t}}_{\mathrm{a_i}}, D^t_i, N_i\rangle\\
&D^t_i = \{d^t_j : j = 0, 1, ...n_i - 1\}\\
&d^t_j = \langle X^t_j, r^t_j\rangle\\
&X^t_j = \{x_1, x_2, ...x_m\}, m > 0\\
&N_i \subset \Lambda
\end{aligned} \tag{12}$$

The radius of local search region is determined afterwards. Let the radius be r.

2) Every agent estimates the probability of white pixels and try to update their locations towards the center of the most dense cluster in their local search space. This probability is the reward at the newly updated location. Updating the home locations towards the most dense region within the immediate neighborhood of an agent a_i follows the rule stated in equation 4. The calculation of the reward factor goes as follows.

$$\begin{aligned}
&X^w_{\mathrm{a_i}} \subseteq X^a_{\mathrm{a_i}} \subset S^t_i\\
&S^t_i \text{ is the local search space of the agent } a_i\\
&\text{at a time instant t. } r^t_i = \frac{|X^w_{\mathrm{a_i}}|}{|X^a_{\mathrm{a_i}}|}
\end{aligned} \tag{13}$$

Here $X^w_{\mathrm{a_i}}$ is the set of white pixels found and $X^a_{\mathrm{a_i}}$ is the set of total pixels counted. Here $n^w_i = |X^w_{\mathrm{a_i}}|$ and $n^t_i = |X^a_{\mathrm{a_i}}|$.

3) Now, those who have a nonzero reward will populate their own decision set as well as their neighbors' decision set.

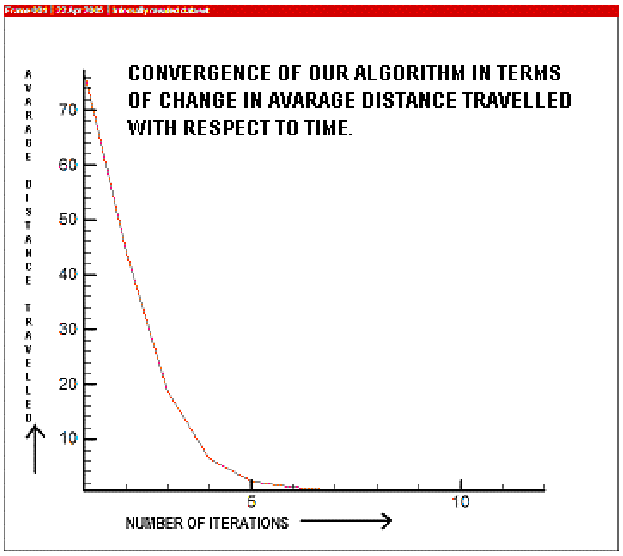


Fig. 10. In this figure we can see average movement of every agent is being reduced drastically at each iteration and at iteration eight the algorithm stabilizes.

4) At this point, when all agents finish inter agent communication, those who have empty decision set will make an unbiased random walk as described in equation 6.
5) The agents who have a nonempty decision set will calculate the probability factor associated with each decision metric available in its decision set in the following way. For every decision metric $d_j^t \in D_i^t$ the probability factor P_j^t will be calculated as follows.

$$e_j^t = \frac{r_j^t + \varepsilon}{\frac{\|X_j^t - X_{\mathrm{a_i}}^t\| + \epsilon}{\lambda}}$$

$$P_j^t = \frac{e_j^t}{\sum_{k=0}^{\mathrm{n_i}-1} e_k^t}, j = 0, 1, ...n_i - 1. \tag{14}$$

Here $\varepsilon, \epsilon, \lambda$ are constants.

Then the agent a_i will take its j^{th} decision with probability P_j^t and adapt their locations as shown in equation 7 and 8.
6) After every agent finish updating their locations they have to determine the attractiveness of their current locations. Here instead of going through a thorough local search we used simple estimation procedure to estimate the attractiveness of the new location. This estimation process follows the rule of equation 10 and 11. Then the whole procedure repeats from step 3 until convergence.

B. Experimental Results

The experimental results are shown in Fig. 9. Here the green circles indicate the position of each agent and the blue lines are drawn to visualize the neighbor set of each agent. In Fig. 10 we have plotted the average distance travelled per agent at each iteration.From this figure it can be seen that our algorithm converges at approximately eighth iteration in this case.

C. Automatic Eight Point Matching For Stereo Images

In this application we have used our algorithm to find out a number of pairs of matched points from a pair of stereo images. To do this, first we have found out 20 feature points in the left image using Gabor wavelet feature detection technique. Here the left image is nothing but image taken by the left camera and similarly the right image is the image taken by the right camera. However after finding out the feature points we shall search through the right image for their matches. The matching is done by using weighted cross correlation over a local window centered at the point of interest. We want to find out the match between two points, one from the left image ,say $(x_{\mathrm{L1}}, y_{\mathrm{L1}})$and the other from the right image, say $(x_{\mathrm{R1}}, y_{\mathrm{R1}})$ and to do so we shall use the following equation to evaluate the similarity measure [14] $\Psi_{\mathrm{I_1,I_2}}(x_{\mathrm{L1}}, y_{\mathrm{L1}}, x_{\mathrm{R1}}, y_{\mathrm{R1}})$.

$$\Psi_{\mathrm{I_1,I_2}}(x_{\mathrm{L1}}, y_{\mathrm{L1}}, x_{\mathrm{R1}}, y_{\mathrm{R1}}) = \frac{\sum_{\mathrm{i,j}} \gamma_{\mathrm{ij}}(I_1(i,j) - \mu_1)(I_2(i,j) - \mu_2)}{\sqrt{\sum_{\mathrm{i,j}} \gamma_{\mathrm{ij}}(I_1(i,j) - \mu_1)^2}\sqrt{\sum_{\mathrm{i,j}} \gamma_{\mathrm{ij}}(I_2(i,j) - \mu_2)^2}}$$

$$\mu_1 = \frac{\sum_{\mathrm{i,j}} I_1(i,j)}{(2\omega_d + 1)^2}, \mu_2 = \frac{\sum_{\mathrm{i,j}} I_2(i,j)}{(2\omega_d + 1)^2} \tag{15}$$

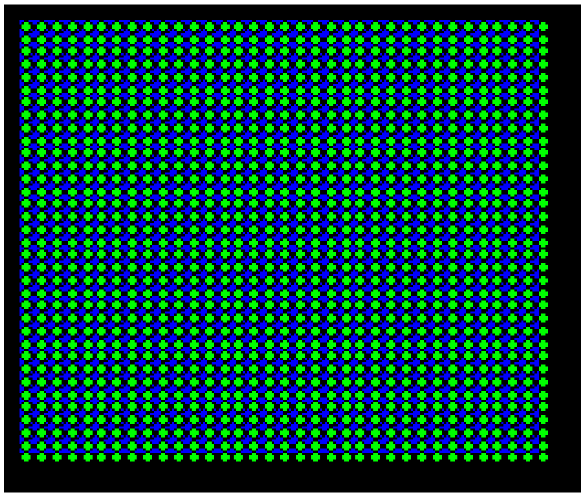
$$\gamma_{\mathrm{i,j}} = exp(-\frac{i^2 + j^2}{\sigma^2})\text{is the weighting function}$$

$$\sigma\text{is a constant typically } \sqrt{2}\omega_d.$$

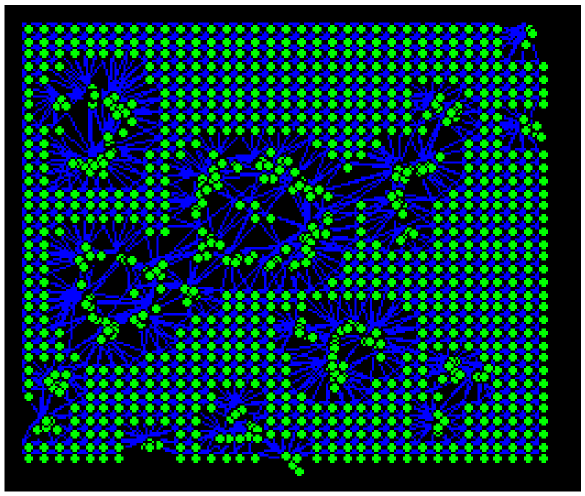
Here i and j are integers. i varies in the range $(x_{\mathrm{L1}} - \omega_d, x_{\mathrm{L1}} + \omega_d)$ and $(x_{\mathrm{R1}} - \omega_d, x_{\mathrm{R1}} + \omega_d)$ in the left and right image successively and j varies in the range $(y_{\mathrm{L1}} - \omega_d, y_{\mathrm{L1}} + \omega_d)$ and $(y_{\mathrm{R1}} - \omega_d, y_{\mathrm{R1}} + \omega_d)$ in the left and right image successively.

We shall use this similarity measure as attractiveness function. First we shall take one feature point in the left image I_1, say $(x_{\mathrm{L1}}, y_{\mathrm{L1}})$, and then define a rectangular window having sides of length $(2\omega_d + 1)$ around this point and then match the windows around every point within an estimated region in the right image. This estimated region in the right image is that region within which we predict the matching point of $(x_{\mathrm{L1}}, y_{\mathrm{L1}})$ is present. We shall not discuss the region estimation procedure here as this is out of the scope of this paper. Suppose this estimated region is $\mathcal{R}_{\mathrm{est,right}}$. We have generated a population of sixteen agents which are then uniformly distributed over this region. Each agent use the above similarity measure to estimate the attractiveness of its current location. In this problem a local search is not necessary. Each agent does not look for the scores in its neighboring regions. The punishment factor is the euclidian distance between the coordinate of the feature point $(x_{\mathrm{L1}}, y_{\mathrm{L1}})$ and the the coordinate of the attractive location towards which it may attempt to make a move. So if the k^{th} attractive location of an agent a_i is $X_{\mathrm{a_i}}^k$ then the punishment factor is the distance $\|X_{\mathrm{L1}} - X_{\mathrm{a_i}}^k\|$. So for the k^{th} attractive location we have calculated the factor e_j^t as

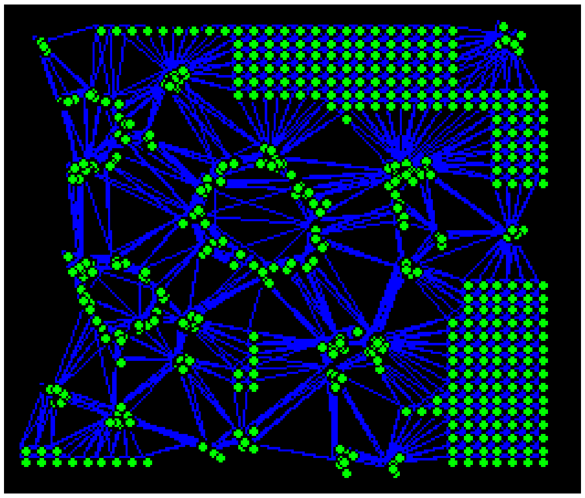
$$e_j^t = \frac{\Psi_{\mathrm{I_1,I_2}}}{(1.01)^{\|X_{\mathrm{L1}} - X_{\mathrm{a_i}}^k\|}} \tag{16}$$



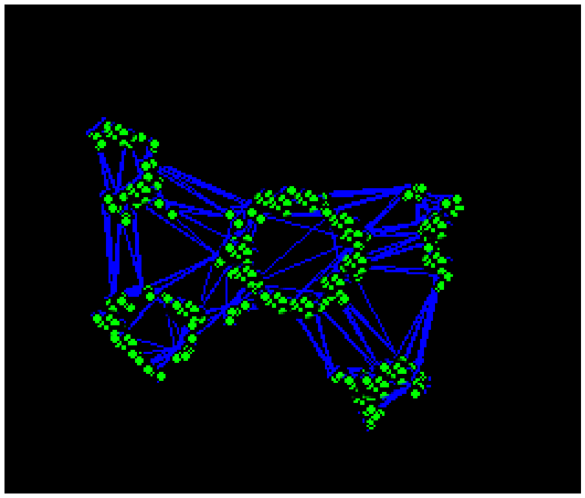
(a) Initial Population



(b) The location of agents after two iterations



(c) The location of agents after four iterations



(d) The location of agents after eight iterations

Fig. 9. In this figure we have shown the movement of the agents in a step by step manner. From the above pictures it can be seen that the algorithm converges at the eighth iteration. In (a) we can see the starting locations of all the agents. Then in (b), (c), and (d) they arrange themselves to take the shape of the denser clusters in the image shown in Fig.8.

Then the probability factor is calculated as

$$P_j^t = \frac{e_j^t}{\sum_j e_j^t} \tag{17}$$

Then everything will follow the steps stated earlier.

D. Experimental Results

The results of the above experiment is shown in figure 11. In the left images the feature points are the upper left corners of the blue rectangles. The rectangles are numbered. In the right image each green ball represents one agent. We have twenty feature points and we have generated twenty populations to find out the matches for the corresponding feature points. The feature points which are very near to the edge of the left image are ignored. The upper left corner of the bounding box of each green circle represents the location of the corresponding agent. After convergence we can see that a few populations have found more than one matches for its corresponding feature point. This is due to inefficiency of the similarity measure we have used here as attractiveness function. However most of the populations have found the appropriate match in the right image for the corresponding feature points in the left image. The most intersting thing is that we have considered the estimated region (hich is the global search space for the corresponding population) as a rectangular region of area $41X41$ pixels. So if we had to go through the whole search space in order to find out the perfect match for the corresponding feature point then we had to search for matches in all of the $41X41$ pixels. However our algorithm has converged at the eighth iteration. So the total number of similarity measures (that we have used for feature matching) we have done is $16X8$ which is less than one tenth of the earlier. But if we use populations of nine agents then the computational cost will be reduced drastically but the result remains almost the same. However we have shown here the results with populations having sixteen agents each.

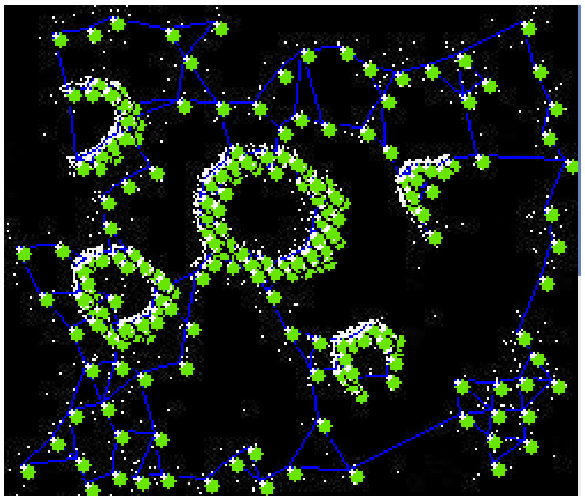


Fig. 12. In this figure we have shown the result of GNG algorithm on the image shown in figure 8. Here the green circles represents the position of the neurons. The neurons have formed clusters at every location wherever they found white pixels. They have not used the density information. They have blindly followed the locations of all the white pixels.

E. A Comparative Study

We have used another very popular self managing algorithm on the image shown in figure 8 to achieve the same result. The results are shown in Fig 12. From fig.12 it can easily be seen that the neurons do not only follow the shape of the areas where the density of white pixels is very high they also form clusters in the area where the density of white pixels is low. These kinds of algorithms do not use the density information; rather, they blindly follow the locations of the white pixels. Not only that, to achieve the result shown in the figure the GNG algorithm took (10 X No. of white pixels) numbers of iterations which is very large compared to that of our algorithm.

Let us now consider a little different problem. Suppose, in the image shown in fig.6(a), instead of finding the area with high density of white pixels we need to do the reverse. That is, we need to find out the areas where white pixel density is lower. In this case we shall not consider the areas which are having no white pixel. Rather we are interested in those areas where very few white pixels are present. Other algorithms like Self Organizing Map or Growing Neural Gas will fail to serve the purpose. But our algorithm can efficiently serves this purpose. In our algorithm we only need to change the attractiveness function. In the application shown above we used the probability of white pixels in a local search region as the measure of attractiveness. Let us denote this attractiveness measure by r_i^t. In the present case we shall use $1 - r_i^t$ as attractiveness measure when r_i^t is not equals to zero. All the other steps remain same. The result is shown in figure 13. From this figure it can readily be observed that the agents have successfully found the areas with lower density of white pixels. So from this experiment we can infer that our algorithm is more flexible than other self organizing systems like growing neural gas.

V. Future Research

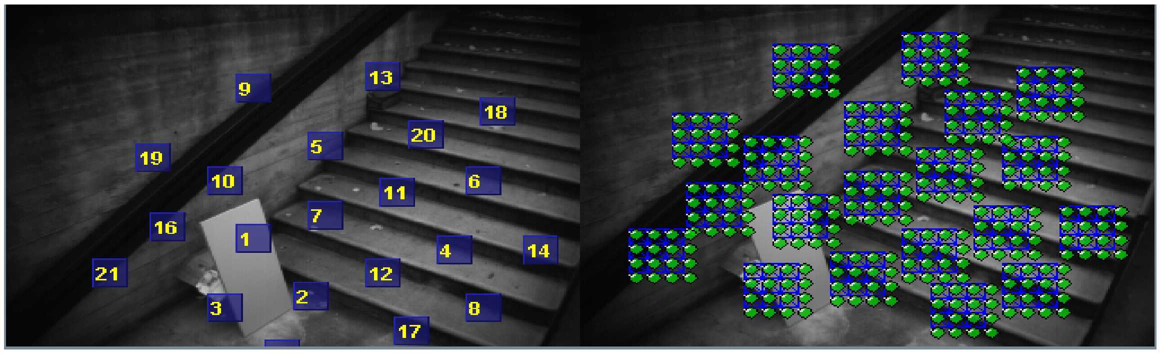
We have successfully applied our algorithm in some image processing and computer vision applications. Now we are trying to solve some problems related to finance and economics using our algorithm. We are now focusing especially on portfolio management systems and vehicle routing problems.

To improve the robustness of our algorithm in specific applications we are also trying to combine our algorithm with some other existing learning and meta heuristic algorithms like tabu search, Q-learning, ACO etc. In this paper we have considered that all the agents have similar characteristics. Now we are trying to create a population where all agent will have some common global characteristics as well as some agent specific characteristics which are not common to all. This is a more realistic approach in order to simulate the behavior of a group of people working in particular situations.

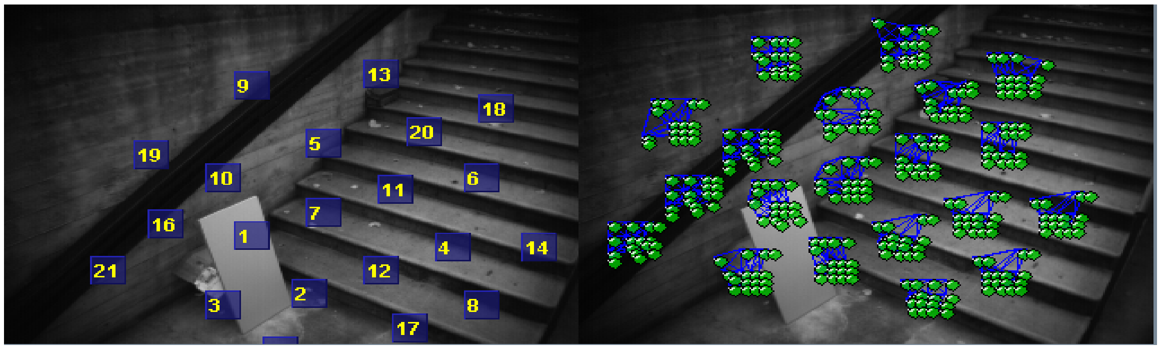
We are also working on a formal logic approach to design the behavior of each agent and then print it in the corresponding agent's gene. Thus we will be able to use genetic programming techniques to evolve the behaviors of each agent. What exactly we are trying is to generate the population in such a way that some agents are weaker is making decision than the others and then we are trying to find whether there is a possibility for the weaker agents to evolve themselves to cope with the stronger agents while working with them. And we are also trying to find whether this kind of population can help in solving more difficult situations.

VI. Conclusion

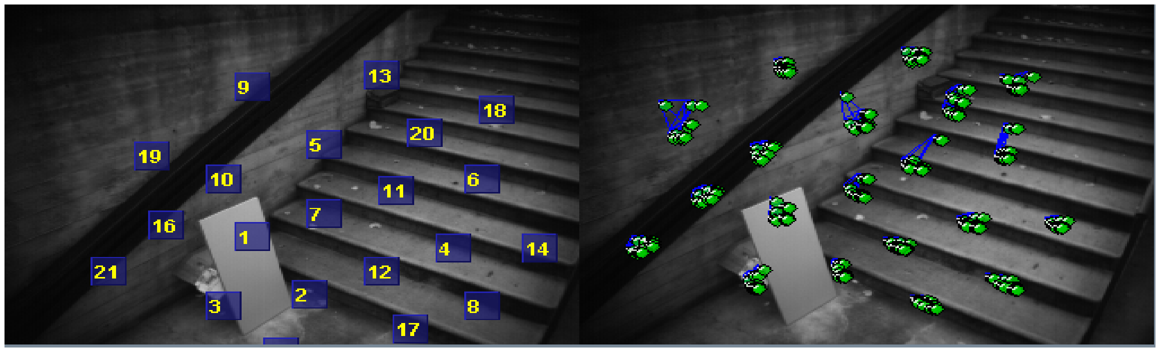
Though we have discussed only a few applications of our algorithm, in practice, it may find more and more implementation areas. So far we have not dealt with dynamically varying situations. This kind of situations can be dealt with accomplishing a local search at every iteration. There are some situations where we have to work in real time. For example consider the case of tracking an object in a video sequence. Though our algorithm is more autonomous and even sometimes computationally efficient than the others like self organizing maps we will have to work a lot to accomplish a good result in real time simulations. The performance of our algorithm depends a lot on the attractiveness estimation procedure, the reward function, the punishment function and the constant parameters. So we have to be careful in designing these estimation procedures, functions and other parameters. The overall performance of our algorithm is comparable with



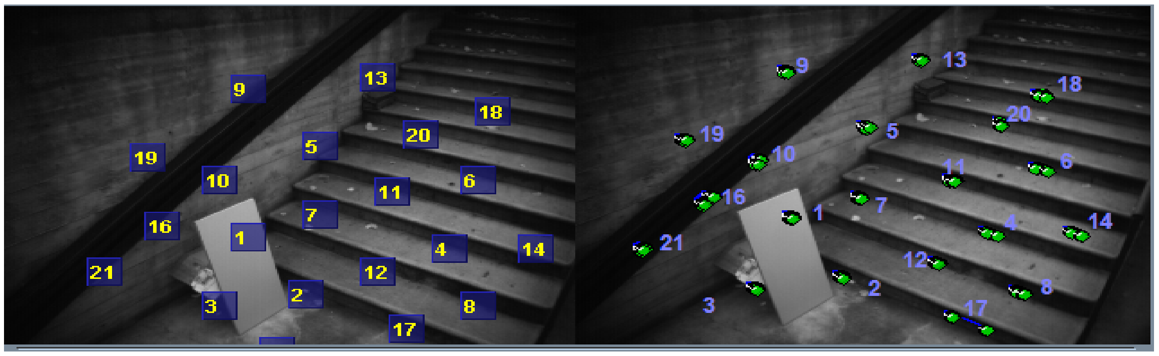
(a) Initial Population



(b) The location of agents after one iteration

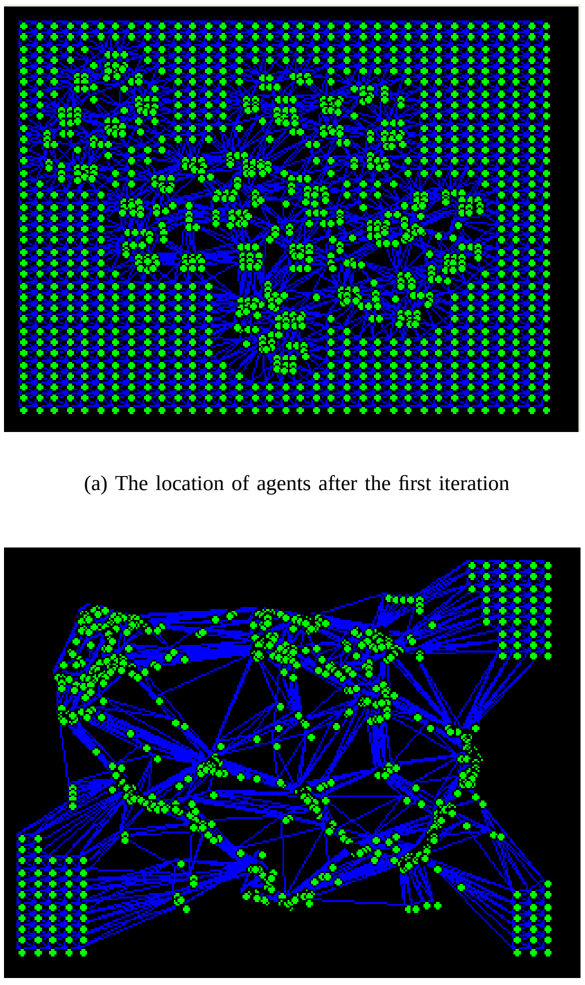


(c) The location of agents after four iterations



(d) The location of agents after eight iterations

Fig. 11. In this figure we have shown the movement of the agents in a step by step manner. From the above pictures it can be seen that the algorithm converges at the eighth iteration. In (a) we can see the starting locations of all the agents. Then in (b), (c),and (d)they arrange themselves to find the proper match for the corresponding feature points in the left image.


(a) The location of agents after the first iteration

(b) The location of agents after four iterations

(c) The location of agents after eight iterations

(d) The location of agents after sixteen iterations

Fig. 13. In this figure we have shown the movement of the agents in a step by step manner. In this case instead of finding the region having high density of white pixels we have tried to implement our algorithm to find out the region with fewer white pixels.

self organizing map but it converges at a faster rate than SOM. A SOM will need hundreds of iterations to accomplish similar results as shown in Fig. 9. where our algorithm took only eight. After all our algorithm is more robust than the Self Organizing Map because it takes into account both the reward and punishment factors and we can design these two factors to serve our purpose.

References

[1] F. Glover. Tabu Search Part I. OSRA Journal on Computing, 1(3):190-206, 1990

[2] T. Feo, M. Resende, and S. Smith. A Greedy Randomized Adaptive Search Procedure for the Maximum Independent Set. Journal of Operations Research, 42:860-878, 1994.

[3] P.M. Van Laarhoven and E.H.L. Aarts. Simulated Annealing: Theory and Applications. Kluwer Academic Publishers, Boston, 1996.

[4] D.E.Goldberg. A comparative analysis of selection schemes used in Genetic Algorithms. In Foundations of Genetic Algorithms, pages 69-93, San Mateo, California 1991. Morgan Kaufman Publishers.

[5] I Osman and J Kelly. Meta-Heuristics: Theory and Applications. Kluwer Academic Publishers, Boston, 1996.

[6] Marco Dorigo, Thomas Sttzle, Ant Colony Optimization. Bradford Books. July 1st 2004

[7] M. Dorigo, V. Maniezzo, and A. Colorni. An autocatalytic optimization process. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milano, Italy, 1991.

[8] M Dorigo, and L. Ganbardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE transactions on Evolutionary Computing, 1(1):53-66,1997.

[9] Martinetz T M,1993, Competitive Hebbian learning rule forms perfectly topology preserving maps, Int. Conf. Artificial Networks , (Amsterdam: Springer) pp 42734.

[10] T. Kohonen. Self-Organizing Maps. Springer-Verlag, Berlin, 2nd edition, 1997

[11] Fritzke B 1991 , “Unsupervised clustering with growing cell structures ,” “Proc. Int. Joint Conf. on Neural Networks 1991(Seattle, WA)” , vol II pp 5316.

[12] “Supervised learning with growing cell structures , ” “Advances in Neural Information Processing Systems ,” 6 ed, J Cowan, G Tesauro & J Alspector , (San Mateo, CA: Morgan Kaufmann) pp 25562.

[13] Bernd Fritzke, A growing neural gas network learns topologies,Advances in Neural Information Processing Systems 7, pp 625–632,1995.

[14] Y.S. Yao and R. Chellappa ”Tracking a Dynamic Set of Feature Points”,IEEE Trans. Image Processing”, Vol.4,Issued 10,pp. 1382-

1395,oct-1995.