Exploring Knowledge and Population Swarms via an Agent-Based Cultural Algorithms Simulation Toolkit (CAT)

Robert G. Reynolds, *Member, IEEE* and Mostafa Z. Ali, *Member, IEEE*

Abstract— **Cultural Algorithms employ a basic set of knowledge sources, each related to knowledge observed in various social species. These knowledge sources are then combined to direct the decisions of the individual agents in solving optimization problems. While many successful real-world applications of Cultural Algorithms have been produced, we are interested in studying the fundamental computational processes involved in the use of Cultural Systems as problem solvers. Here we describe a Java-based toolkit system, the Cultural Algorithm Toolkit (CAT) developed in the Repast Symphony Simulation environment. The system allows users to easily configure and visualize the problem solving process of a Cultural Algorithm. Currently the system supports predator/prey problem solving in a "Cones World" environment as well as a suite of benchmark problems in engineering design. Example runs of a predator prey example are presented to demonstrate the systems' capabilities.**

I. Introduction

Cultural Algorithms [1], [2] can provide a flexible framework in which to study the emergence of complexity in a multi-agent system (MAS) [3] [4]. Here we have embedded the Cultural Algorithms framework within the recursive porous agent simulation tool (Repast) [5], [6], producing a toolkit called Cultural Algorithms Simulation Toolkit (CAT).

In this paper we introduce this agent-based simulation toolkit for exploring knowledge, and social intelligence via Cultural Algorithms. A brief overview of the system is given in Fig. 1. The system is composed of the Cultural Algorithms Core System, the visualizer subsystem, a database (Access or MySQL) and the analyzer subsystem (geometry and statistics). The system is implemented under Repast (an Agent-Based Simulation Environment developed at Argonne National Laboratory. The visualizer sub-system is used to display the results, draw some conclusions about the data, and construct some basic real-time dynamic charts and statistics.

In section 2, we describe the Cultural Algorithm system. Section 3 discusses the base agent model. In section 4 the experimental framework will be discussed. Implementation of the overall system will be discussed in section 5. Section 6 is dedicated to test the created system, and demonstrate its capabilities. Section 7 presents our conclusions and suggests future work.

This work was supported in part by the IEEE under the Karplus Summer Research Grant, 2006.

R. G. Reynolds is a professor at Computer Science Department, Wayne State University, MI, 48202, USA, e-mail: reynolds@cs.wayne.edu

M. Z. Ali is a PhD student at Computer Science Department, Wayne State University, MI, 48202, USA, e-mail: mostafa@wayne.edu.

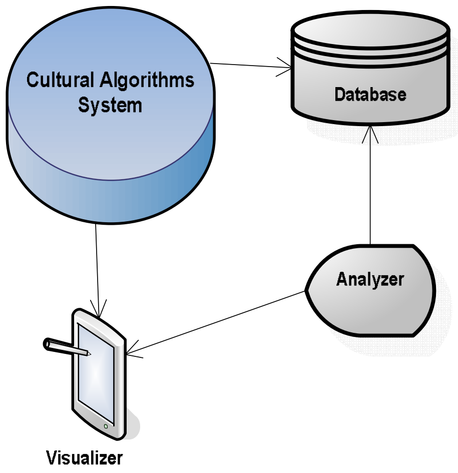


Fig. 1. The overall CAT system architecture.

II. Introduction to Cultural Algorithms

The Cultural Algorithm (CA) [1] is an evolutionary computation model derived from the cultural evolution process. The major components of a Cultural Algorithm are depicted in fig. 2. Besides the population component that traditional evolutionary computation methods have, there is an additional shared knowledge component (belief space) and a supporting communication mechanism between the belief space and the population component. The experience of individuals selected from the population space is used to generate problem solving knowledge that can reside in the belief space. The belief space stores and manipulates the knowledge and in turn influences the evolution of the population component. In this way, the population component and the belief space interact with and support each other. As such, the Cultural Algorithm is a dual inheritance system where both the population of individuals and their beliefs evolve over time in parallel. It provides a general framework within which to describe complex self-adaptive systems.

Cultural Algorithms have been studied with benchmark optimization problems [7] as well as in a variety of application areas such as modeling the evolution of agriculture, concept learning [[1], real-valued function optimization [8] and the re-engineering of knowledge bases for manufacturing assembly process [9].

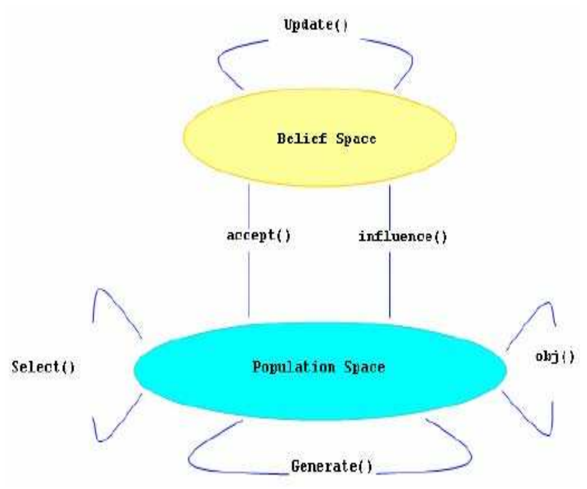


Fig. 2: Cultural Algorithms Framework

In the earliest Cultural Algorithms only one knowledge source was used in the belief space. This was typically tailored to direct the search for the problem at hand. Situational knowledge was used first, then normative, topographic, domain, and finally history knowledge in succession. Their additions reflect an evolution in the complexity of the problems to which Cultural Algorithms were applied. The current configurations in this tutorial use all of them to differing degrees. Each knowledge source has a different set of properties that affects its ability to generate new individuals in response to knowledge produced through the influence of the other knowledge sources [10], [11]. The knowledge sources are described here in terms of their ability to coordinate the spread of individuals over the landscape for a problem.

Topographical Knowledge was originally proposed to reason about region-based functional landscape patterns [7]. It can distribute individuals potentially over the entire landscape. It was motivated in conjunction with data mining problems where the problem space was so large that a systematic way of partitioning the space during the search process was needed. If we view a state as associated with a region in the functional landscape then the topographic knowledge source is looking for new states. Thus, the state space may vary dynamically as new sub-regions are discovered and added to the mix.

Normative Knowledge is a set of promising variable ranges that provide standards for individual behaviors and guidelines within which individual adjustments can be made. Normative knowledge came into play during the learning of rules for expert system applications. Normative Knowledge directs individuals to “jump into the good range” if they are not already there. In other words it produces conduits or regional landing strips that guide the population in moving from one attractive region to the next. It therefore, spreads individuals into sub-regions of the space.

Domain Knowledge uses knowledge about the problem domain in order to guide search. Domain Knowledge was first used to find the resource cone(s) of maximum height in a landscape by Saleem [12]. In this case, problem knowledge defines properties of the resource cones that make up the performance surface. For example, in a functional landscape composed of cones, knowledge about cone shape and related parameters will be useful in reasoning about them during the search process. In particular, domain knowledge can be used to generate individuals up and down slope from a location as a function of cone slope and height. Once a relatively productive area is found, the domain knowledge can guide the exploitation of that region.

Situational Knowledge provides a set of exemplary cases that are useful for the interpretation of specific individual experiences. Situational Knowledge leads individuals to “move toward the exemplars”. This was the earliest knowledge source used with Cultural Algorithms and was inspired by elitist approaches in Genetic Algorithm. This knowledge source collaborates with domain knowledge to exploit above average regions.

Historical or Temporal knowledge monitors the search process and records important events in the search. This knowledge source was first used by Saleem [12] and expanded by Peng [13]. Individuals guided by History knowledge can consult those recorded events for guidance in predicting a good move direction. There is no dynamic component in the static examples used in this tutorial, so the variance associated with the knowledge source is a function of the local topography. In more dynamic problems it can distribute individuals over the entire space akin to topographic knowledge in order to monitor regions that were highlighted in the past.

III. Basic Agent Model

The Cultural Algorithm (CA) has three major components: a population space, a belief space, and a protocol that describes how knowledge is exchanged between the two components. The population space can support any population-based computational model, such as Genetic Algorithms, Genetic Programming, Evolutionary Programming, Particle Swarms, Ant Colony Optimization, and agent-based systems among others.

Figure 3 shows how the basic Cultural Algorithm objects are defined and related to each other in the CAT system. Cones exist in certain regions, and are generated randomly by the system for each run. Each knowledge source in the Cultural Algorithms corresponds to a predator, where each predator controls a number of individual agents in the search for prey (resource cones). Each predator distributes agents under its control over the landscape.

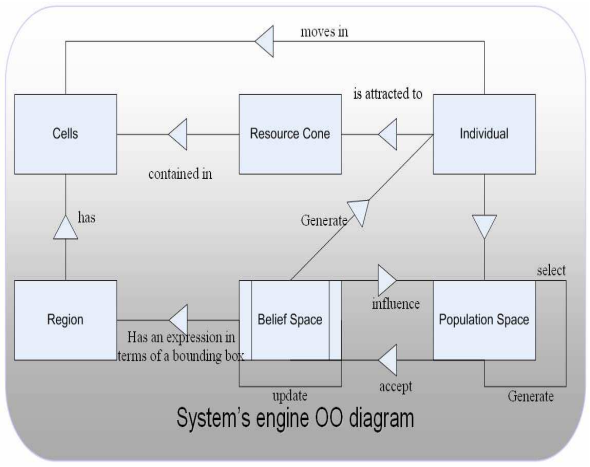


Fig. 3. Object Relationship Model for CAT

While large-scale Cultural Algorithms applications were developed using detailed domain knowledge for a specific applications [11], here we are interested in studying general-purpose problem solving behavior. From that perspective we will use the five basic types of knowledge sources within which specific knowledge for a given problem can be expressed. What is crucial here is how to assign agents to the various knowledge sources at each time step, a knowledge integration task. The answer can be revealed by imagining each knowledge source as a generator having an expression in our 2D space in terms of a bounding box characterized by a midpoint and a standard deviation along each of the two dimensions. The bounding box statistically describes the distribution of individuals generated by the knowledge source in the past time step. If we view each bounding box as analogous to a resource patch in the environment, then every knowledge source can be viewed as a predator that searches for prey in a given patch.

The performance of individuals in the population space is evaluated within a performance environment *obj()*. This performance environment can be a real or simulated social system. An acceptance function *accept()* will then determine which individuals are to influence the belief space. The experiences of those selected individuals can be used to update the knowledge/beliefs of the belief space via *update()* [10], [11]. The *update()* function represents the evolution or modification of existing beliefs as a result of individual experience over time.

In the system we support two classes of performance environments. First, there is the predator prey problem class where individuals move through an environment produced by a configuration of resource cones [14], [15], [16], looking for the optimum peak. In the second set of problem the performance environment relates to a given engineering optimization problem. A collection of such problems has been taken from the literature. We will focus on the former class of problems in this paper.

For the foraging problem class, cones are generated randomly by our system and passed to the display. The region consists of many cells, depending on how the cone in that area is extended. The prey corresponds to the resource cones while the knowledge sources in the Cultural Algorithms correspond to the predators as mentioned above. Each predator distributes agents under its control over the landscape. The Marginal Value Theorem (MVT) derived from predator/prey models [13] is used to guide the distribution of agents by predators in order to maximize the resource intake of the system over time. The gain produced by the agent there is then recorded for the predator there.

The performance of a knowledge source can then be generated via computing the average fitness value of all individuals generated by each knowledge source. The average fitness value of individuals generated using a specific knowledge source (predator) is:

$$avr_i = \frac{\sum_{j=1}^{k} f_j(x)}{k} \qquad (1).$$

Where k is the number of individuals generated via the knowledge source and $f_j(x)$ is the fitness value of individual j.

IV. Experimental Framework

A. Experimental Configurations

The landscape is a two dimensional rectangular region space, where $X \in (-1.0, 1.0)$, $Y \in (-1.0, 1.0)$ with different configuration of parameters of the number of cones (n), the range of the heights of the cones to be generated (H), the range of the slopes of the cones to be generated (R), following the optimization problem, see Fig. 4 for a sample landscape generated by the CAT for n=100, H$\in$ (1, 10), and R $\in$ (8, 20).

Each of these independently specified cones are merged together. Whenever two cones overlap at a point, the height of the point is the height of the cone with the largest value at that point.

A more detailed description of the generated landscape can be illustrated from Fig. 5, where some of the generated cones in the landscape along with the foraging agents searching for the maximum resource cone are displayed.

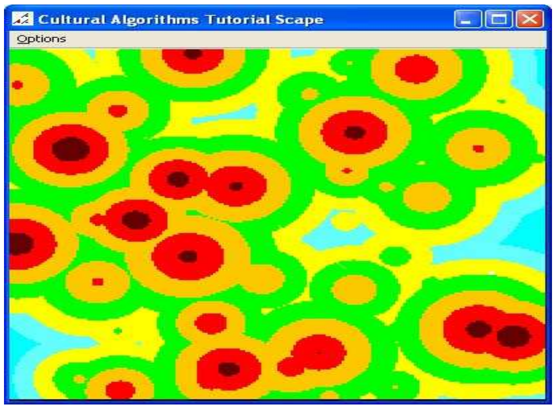
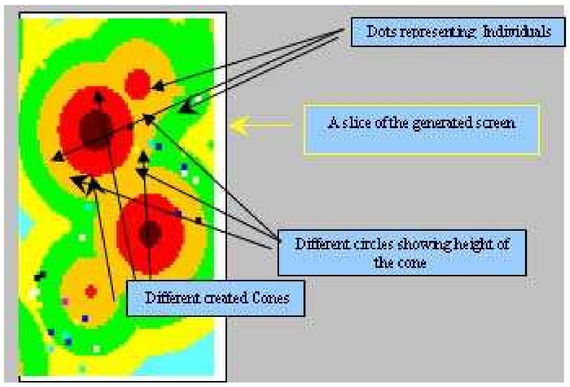


Fig.4. A landscape with the patchwork cones distribution plotted as contour maps. Different colors show different heights of the cones.

In our system we convert the previously specified ranges for *X*, and *Y* to a range between 0 and 200 respectively so we can specify more realistic ranges of values for the heights and slopes of the generated cones in the Cultural Algorithms core. Then these will be converted back to the (-1.0, 1.0) ranges that the underlying Repast system expects for its display. This transformation can be shown in the following code segment presented in Fig. 6 where:
`viewMaxX`, `viewMaxY`: the maximum X and Y values to visualize.
`viewMinX`, `viewMinY`: the minimum X and Y values to visualize.


Dots representing Individuals
A slice of the generated screen
Different circles showing height of the cone
Different created Cones
Fig.5. A slice of the generated system in CAT

```
public int[] convertLocation(double x, double y){
      int retVal [] = new int[2];
      double xMultiplier = (200)/(viewMaxX - viewMinX);
      double yMultiplier = (200)/(viewMaxY - viewMinY);

      retVal[0] = (int)((x - viewMinX) * xMultiplier);
      retVal[1] = (int)((y - viewMinY) * yMultiplier);
      return retVal;}
```
Fig.6. Scaling the coordinates and the fitness value in CAT

B. *The cones World Problem Generator*

As mentioned above, the cones world generates a problem landscape, in which a field of resource cones of different specified heights and slopes that can be randomly scattered by CAT, the two-dimensional landscape. The landscape is given by:

$$f(<x1,x2,...,x_n>) = \max_{j=1,k}(H_j - R_j * \sqrt{\sum_{i=1}^{n}(x_i - C_{j,i})^2}) \qquad (2)$$

where, k - the number of cones, n - the dimensionality, H_j – height of cone *j*, R_j – slope of cone *j*, and $C_{j,i}$ – coordinate of cone *j* in dimension *i*. Each of these parameters can be specified by the user based on the following variables:

$$H_j \in (\text{Hbase}, \text{Hbase} + \text{Hrange}) \qquad (3)$$
$$R_j \in (\text{Rbase}, \text{Rbase} + \text{Rrange}) \text{ and } C_{j,i}. \qquad (4)$$

The main reason that we selected this generator is that by changing its parameters, it generates test functions over a wide range of surface complexities. This enables us to evaluate our model in a more flexible and systematic way.

C. *Agents and Landscape*

As mentioned previously, a mapping scheme is used to map our real-valued layer to the discrete layer display in Repast. Accordingly, the individual must also be scaled to be compatible with the generated view of the landscape as in Fig. 7 below.

Once the parameters are configured, the simulation executes according to the chosen Cultural Algorithm configuration and problem environment. A sample screen for the following set of parameters, CPopSize=100, number of cones = 25, and 2-dimesions, is illustrated in Fig. 8.

```
private Individual scale(Individual agent)
{
      int x;
      for (int i=0; i<dim; i++)
      {
         x = (int) (agent.getDimensionValue(i)*100+100;
         agent.setScaledDim(i,x);
      }
      return agent;
}
```
Fig. 7. Scaling the individual to accommodate the changes in the view of CAT landscape

Individual agents are displayed on the landscape using different colors depending on the knowledge source that they are currently being controlled by. The color scheme for these individuals and the bounding boxes is given in table 1.

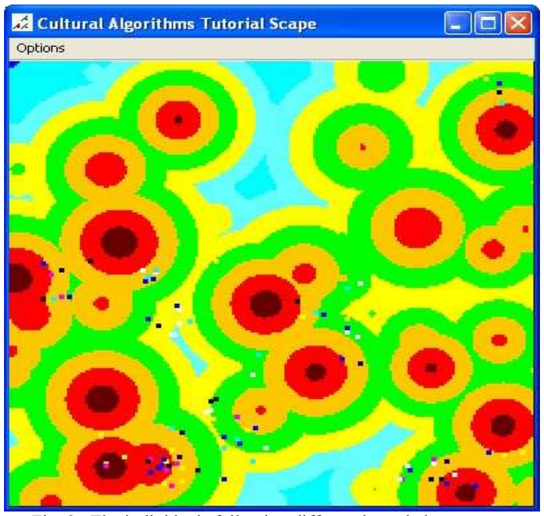


Fig. 8. The individuals following different knowledge sources are shown on the screen moving between cones on the landscape.

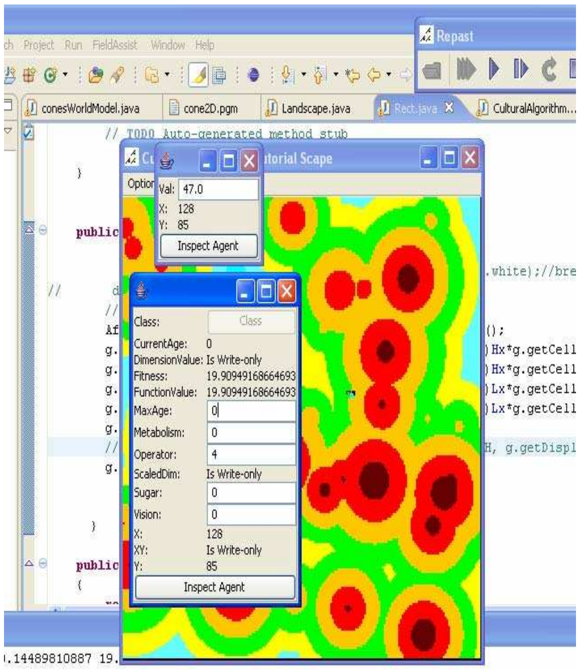
In the default mode, a cones world landscape is randomly generated based on the parameters specified by the user. The random configuration is stored in a file, Landscape.java. The system reads the landscape information of the cones from from there in order to generate the display for the problem..

We can run our model in GUI mode so that agents' properties can be displayed in a probe window. We would like these properties to be updated every model time step. This can be done via the GUI. To turn this on via the GUI, switch to the Repast actions tab panel and click on Update Probes. This is equivalent to setting the UPDATE_PROBES flag in Controller to true in the code.

Figure 9 shows the display produced when the simulation was paused, and one of our agents was probed to display its properties. These properties include location, fitness, vision and many more properties. So in order to probe an object, pause the simulation, and left click on the object in the display. The probeable state of all the objects underneath the cursor will appear in windows similar to the "Parameters" tab in the settings window, and their properties can be set by changing the property and pressing enter.

TABLE I
COLOR REPRESENTATION FOR THE PREDATOR KNOWLEDGE SOURCES AND INDIVIDUALS

Knowledge Types	**Color**
Normative	🟦 blue
Situational	O white
Domain	🟩 green
History	🟨 yellow
Topographical	🟦 cyan



```
// TODO Auto-generated method stub
```

Val: 47.0
X: 128
Y: 85

Class:	Class
CurrentAge:	0
DimensionValue:	Is Write-only
Fitness:	19.90949168664693
FunctionValue:	19.90949168664693
MaxAge:	0
Metabolism:	0
Operator:	4
ScaledDim:	Is Write-only
Sugar:	0
Vision:	0
X:	128
XY:	Is Write-only
Y:	85

Fig. 9. Working through the probing feature to display properties about the agent in our display in CAT

Generally a screen display can slow down the run substantially, so Repast [5] will not update the display if the display window is minimized, allowing for dynamic display updates. (Of course, these kinds of updates can be set explicitly through a Repast Schedule.) When the window is displayed again it will be updated in the usual manner. The scheduling activity is described below.

D. *Scheduling of Events in the Simulation*

Event scheduling in Repast consists of setting up method calls on objects so that they occur at certain times. However, these method calls must be wrapped as sub-classes of the ***BasicAction*** class. A *BasicAction* consists of some variables used by the Scheduler and an abstract public void execute() method. Any classes that sub-class a *BasicAction* must implement this method. It is in this method that the actual method will be scheduled. The user can write a sub-class of *BasicAction* yourself, or have Repast create one automatically.

In terms of the evolutionary process, there is a set of basic events associated with the execution of the Cultural Algorithm. We call upon a Cultural Algorithms object to process one generation, increment the year, and display all of that on the screen after one complete year has elapsed. The system is paused using a delay thread in order to get a complete view of the system in terms of the population (moving agents), and the belief space (bounding boxes) as well as all calculated statistics and charts. We then enter a loop to get an agent from the population, scale its parameters, and then add it to the grid. The last part of the code deals with placing bounding boxes around the certain regions for the individuals controlled by our knowledge sources (predators) as will be explained in more detail later.

Scheduling *BasicAction*s for execution is done via the Schedule object. It allows the user to schedule actions to occur related to several temporal events. Events can be scheduled at the following ways: at every time step beginning at some *specifiedsimulation* time, at some specified tick, at intervals, at a pause in the simulation, and at the end of the simulation. A tick is a single iteration over all the *BasicAction*s scheduled for execution at that time. It is important to realize that ticks are only a way to schedule actions relative to each other and they are not discrete entities themselves. Actions scheduled for tick 3 will complete execution before actions scheduled for tick 10 begin execution. But, if nothing is scheduled between times 3 and 10, the actions scheduled for 10 will begin to execute immediately after those for tick 3 complete. In this way, the ticks from 4 - 9 do not exist. Clock cycles maybe fractional values as well. For example, it is possible to schedule an action for execution at tick 1.25. This is shown in our code, in the *buildSchedule*() method in *ConesWorldModel.java*, as in Fig. 10.

V. Experiments and Results

A. Configuring parameters

Using the CAT tool, we ran several experiments, with different problem configurations to illustrate its operation. The performance function and the related parameter settings are the only parts that vary from problem to problem here. After the performance function is configured appropriately, other portions of the Cultural Algorithm subsystem will take in the changes (dimensions, range of each dimension, and so on) automatically.

B. Results In terms of belief space and other statistics

Using CAT, we ran an example of the Cones World using 100 cones, 50 individuals, and 200 years. The parameters used are shown in Fig. 11. Figures 12 and 13 illustrate the runs and the bounding boxes produced by the system at years 5, and 100.

In our system, the relative positioning of the bounding boxes will serve to generate individuals into the associated area. From the figures we can see the advantage using the bounding boxes to describe "landing strips" for new individuals. The overlapping of these boxes corresponds to the "knowledge swarming" at the meta-level.

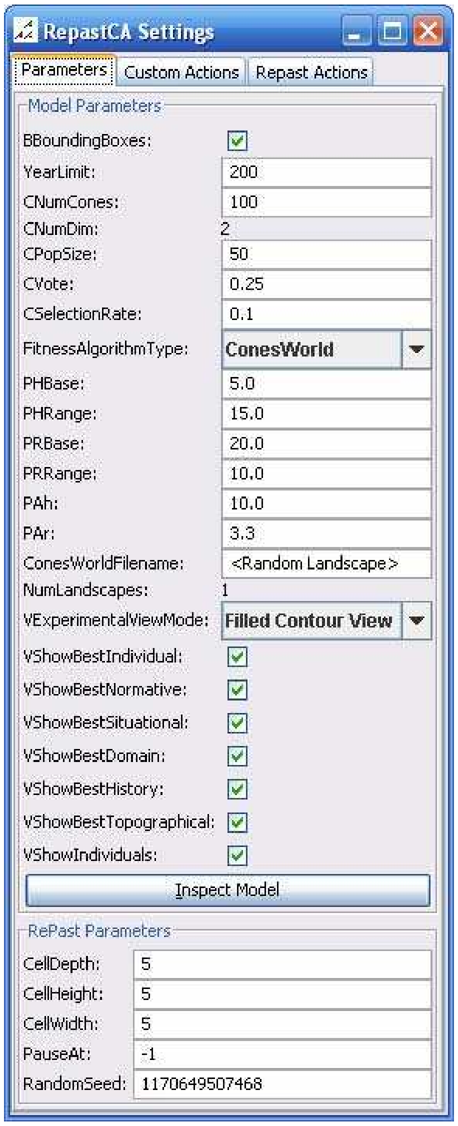
The bounding boxes represent the focus of the search process for each knowledge source. Notice that initially in Fig. 12 that the bounding boxes associated with the topographic and domain knowledge sources cover most of the space.

The tendency is that the bounding boxes for the fine-grained search process have separated from those for the coarse-grained phase and have surrounded the optimal value for this pair of dimensions. These bounding boxes are effectively channeling new individuals into this area. This process is the same for dimensions d and N.

```
int stopTick = yearLimit; // *10

schedule.scheduleActionAt(stopTick,this,    "stop",
          schedule.LAST);
```

Fig. 10. Schedule the simulation to be stopped after yearLimit years, where each year is 1 day for now.



BBoundingBoxes:	✓
YearLimit:	200
CNumCones:	100
CNumDim:	2
CPopSize:	50
CVote:	0.25
CSelectionRate:	0.1
FitnessAlgorithmType:	ConesWorld
PHBase:	5.0
PHRange:	15.0
PRBase:	20.0
PRRange:	10.0
PAh:	10.0
PAr:	3.3
ConesWorldFilename:	<Random Landscape>
NumLandscapes:	1
VExperimentalViewMode:	Filled Contour View
VShowBestIndividual:	✓
VShowBestNormative:	✓
VShowBestSituational:	✓
VShowBestDomain:	✓
VShowBestHistory:	✓
VShowBestTopographical:	✓
VShowIndividuals:	✓
CellDepth:	5
CellHeight:	5
CellWidth:	5
PauseAt:	-1
RandomSeed:	1170649507468

Figure 11: parameters used in the Cones World

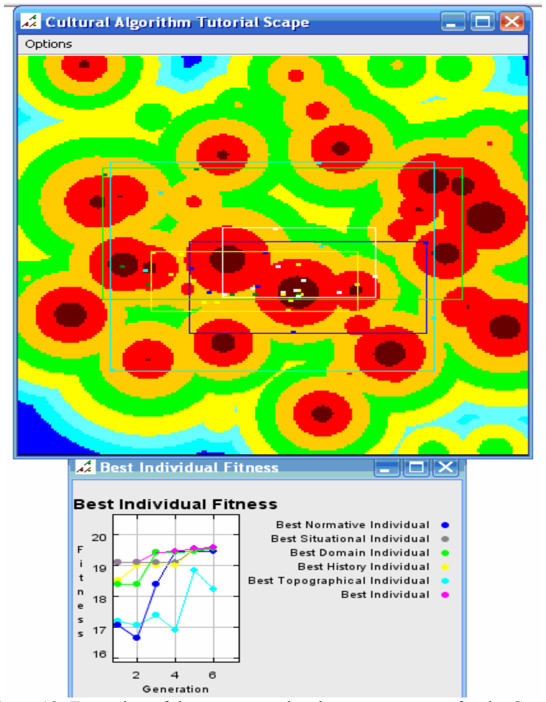


Figure 12: Execution of the system on the chosen parameters for the Cones World problem at simulated year 5.

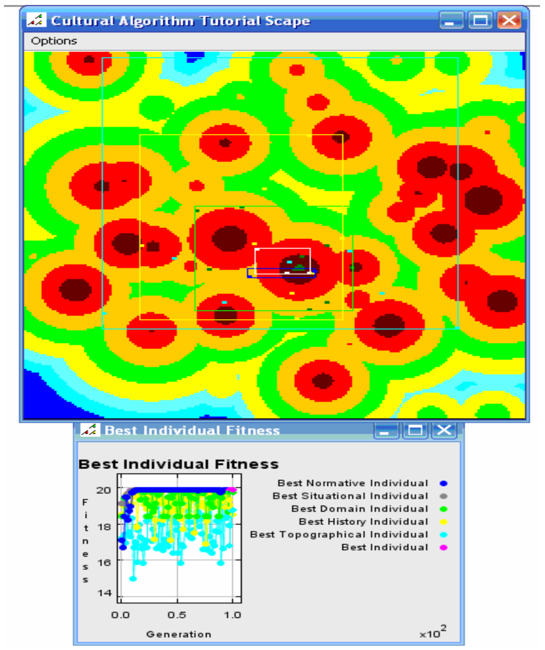


Fig. 13: Execution of the system on the chosen parameters for the Cones World problem at simulated year 100.

VI. Conclusion

The Cultural Algorithm is a stochastic optimization method that models Cultural Evolution and social behavior. Just as cultural evolution contributes to the adaptability of human society, Cultural Algorithms provide an additional degree of adaptability to evolutionary computation.

The system presented in this paper explains the basics of a Cultural Algorithm Toolkit system (CAT) and the platform it runs on, Repast. In CAT, the knowledge sources associated with cultural knowledge control the search process of the individuals in terms of the currently specified optimization problem.

The system has no feasible solution points prior to start with but the swarming of the knowledge sources in the belief level engendered by the constraints produced swarming at the population level which leads to quick convergence. Thus, the interaction of the knowledge sources produced a synchronization of the bounding boxes so as to allow the system to mimic a branch and bound process as shown in fig. 11 above. The coarse grained topographic and normative knowledge sources branch first to various potential solutions in the coarse-grained process, and then situational, domain, and history knowledge work to bound the solution region in the fine-grained process. The system therefore learned to combine the generative processes associated with the various knowledge sources in order to branch out to find other feasible regions, and then bound the solutions in the fine-tuning phase. This approach produces phases of problem solving called coarse grained and fine grained, respectively.

In future work we will extend the system to support problem solving in dynamic environments, and allow the visualization of n-dimensional environments. This will allow us to investigate the social structures and roles that emerge in a Cultural System as the result of successful or unsuccessful problem solving activities.

References

[1] R. G. Reynolds, “An Introduction to Cultural Algorithms, ” in Proceedings of the 3rd Annual Conference on Evolutionary Programming, World Scientific Publishing, pp 131-139, 1994.

[2] Reynolds, R. G., and Peng, B.*, “Cultural Algorithms: Computational Modeling of How Cultures learn to Solve Problems “, Seventeenth European Meeting on Cybernetics and Systems Research, Vienna, Austria, April13-16, 2004.

[3] Reynolds, R.G. Whallon, R. Ali, M.Z. Zadegan, B.M., "Agent-Based Modeling of Early Cultural Evolution", in: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on, pp. 1135- 1142.

[4] Reynolds, R. G., 1978, “On Modeling the Evolution of Hunter-Gatherer Decision-Making Systems.” Geographical Analysis, 10(1), 31-46.

[5] North, M.J., N.T. Collier, and J.R. Vos, "Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit," ACM Transactions on Modeling and Computer Simulation, Vol. 16, Issue 1, pp. 1-25, ACM, New York, New York, USA (January 2006).

[6] http://repast.sourceforge.net

[7] X. Jin and R. G. Reynolds, “Using Knowledge-Based Evolutionary Computation to Solve Nonlinear Constraint Optimization Problems: a Cultural Algorithm Approach, ” in Proceeding of the 1999 Congress on Evolutionary Computation, pp 1672-1678.

[8] Chung, C. & Reynolds, G. R., 1998, “CAEP: An Evolution-based Tool for Real-Valued Function Optimization using Cultural Algorithms.” International Journal on Artificial Intelligence Tools, 7(3), 239-291.

[9] N. Rychlyckyj, D. Ostrowski, G. Schleis and R. G. Reynolds, “Using Cultural Algorithms in Industry, ” in Proceeding of the 2003 IEEE Swarm Intelligence Symposium, pp 187-192, 2003.

[10] R. G. Reynolds and S. Saleem, “The Impact of Environmental Dynamics on Cultural Emergence, ” Festschrift, in Honor of John Holland, Oxford University Press, pp1-10, 2003.

[11] Reynolds. R. G., Brewster, J. 2003. “Cultural Swarms: Modeling the Impact of Culture on Social Interaction and Problem Solving, ” in Proceedings of 2003 IEEE Swarm Intelligence Symposium, IEEE Press, 205-211.

[12] Saleem, S. 2001. Knowledge-based Solution to Dynamic Optimization Problems Using Cultural Algorithms, Ph.D. Thesis, Wayne State University, 2001.

[13] Peng, B. 2005. Knowledge Swarms in Cultural Algorithms for Dynamic Environments, Ph.D. thesis, Wayne State University, 2005.

[14] Morrison, R., and De Jong, K. (1999). A test problem generator for non-stationary environments. In Proceeding of the Congress on Evolutionary Computing, 2047–2053.

[15] Hu, X., Eberhart, C. R. & Shi, Y., 2003, “Engineering Optimization with Particle Swarm.” In Proceedings of the 2003 IEEE Swarm Intelligence Symposium (Indiana, U.S.A.), IEEE Press, 53-57.

[16] Charnov, E. L. (1976). “Optimal Foraging: the Marginal Value Theorem”. Theoretical Population Biology, 9, 129-136.