Tool for Evaluating Human Assisted Swarms

Jason E. Dengler
Department of Computer Science John Carroll University
20700 North Park Blvd.
University Heights, OH 44118
jdengler10@jcu.edu

Faculty Advisors: Daniel W. Palmer and Marc Kirschenbaum

ABSTRACT
Swarm algorithms - a decentralized problem solving approach - can sometimes succeed where deterministic techniques cannot, but they have small application niches and potential thrashing problems. In human assisted swarms (HAS), humans provide high-level information to a swarm to offset these limitations. We present a Flash-based tool to explore HAS using jigsaw puzzles. The program slices up high-resolution Jpeg images, randomizes them, and then animates a swarm attempting to put them back together. Humans watching the swarm specify strategies for the swarm to use. This tool will help determine the best ways for humans to assist swarm algorithms.

1 INTRODUCTION

The idea of swarm algorithms in computer programming is modeled after the collective behaviors of social insects such as some species of ants. Swarms of social insects use a large number of individual decisions that are based upon local information to solve a problem - in the case of social insects the problem is survival. Each "agent" in the swarm can only use information locally available to them because there is no central knowledge or leadership, the "queen" ant is a misnomer, ant societies have no hierarchical decision making process. Every ant in the swarm acts on only the knowledge of their local surroundings but, even with this small amount of information they are able to solve even extremely complicated problems. This idea of using swarm algorithms in computer science can lead to finding faster and better solutions to problems.

Through the combination of the small decisions by each member of the swarm large scale trends can show. These trends are called emergence which is the production of high level properties from low level interactions. Randomness is the key concept underlying emergence. Through the random actions of many "agents" attempting to achieve goal very complex solutions can arise.

A swarm algorithm is very good at solving a problem from the bottom up, until the problem is about 80% solved. At this point the randomness works against the swarm and local information can cause the swarm to thrash. The swarm ends up fighting against itself causing negative performance and not getting any closer to the solution. The swarm has no concept of the large picture of the task, and therefore can be unaware of self-undermining behavior. It is unable to see things such as patterns or trends. Different portions of the swarm might think that they have the problem solved and fight to make their part of the solution permanent. Often, to finish off the last 20% of the problem without

having the swarm thrash, some high level information and direction is needed. [1]

Productivity of a swarm can range from being efficient to self-destructive. If the local information that the swarm gets is useful it can lead to a very fast solution while if the local information is misleading this information can end up being passed throughout the swarm. The self-destructive activity happens because the swarm can't see the big picture. An example of this efficiency problem from a real biological swarm is when ants get into a "Circular Mill" of Death. This phenomenon is depicted in Figure 1. When ants move as a group the ants in the front are the searcher ants that lay down a pheromone for the other ants to follow. All ants that are behind these front ants just follow the scent. If the ants in the front run into another group of ants they will begin to follow this new group. However, if the ants in the front walk in a circle and end up finding the ants in the back of their own group they will end up just following themselves causing this "Circular Mill" of Death. The ants can't see the big picture so they think they just joined a different group while instead they are trapped in an infinite loop. [2]

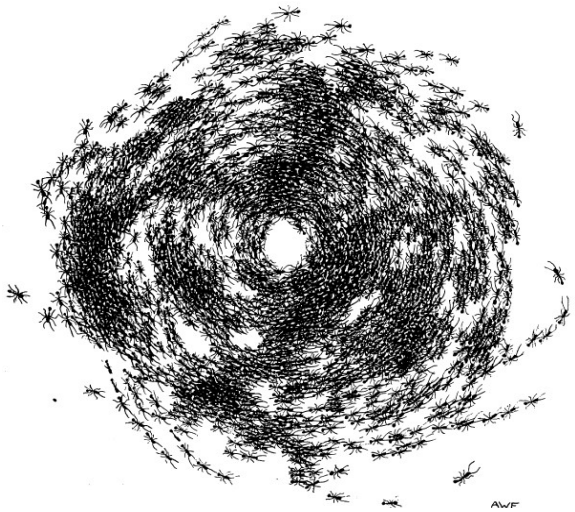

Figure 1. Ants in a "Circular Mill" of Death [2]

Unlike swarms, humans can see the big picture when attempting to solve a problem. Usually they are able to pick out patterns or trends and can use a combination of these patterns and other strategies that help them solve the problem faster. Humans get bored easily and look for different (better) ways to accomplish a task faster.

2 HUNAN ASSISTED SWARMS

Using these ideas, that swarms are very good at solving certain problems from a low level and human logic is good at solving a problem from a high level, we can combine these ideas into a hybrid attempt to solve a problem. By using the best of both approaches we hope to provide a better solution than a swarm alone.

So how would a human assisted swarm work? It would work the same way a normal swarm does, but with a human providing high-level information to help steer the swarm to a solution. The swarm then reacts to this information doing things that it would not have normally done, yet continuing to otherwise act as before.

An example of a swarm problem that could benefit from human assistance is the application of unmanned aerial vehicles (UAV). If you have a swarm of UAVs out scouting an area and one of them goes by an area with a problem a human can quickly notice that there is a problem and converge many other UAVs on this area. The swarm, without any human assistance might not be able to see the significance due to its lack of understanding of the big picture. This might cause the swarm to not converge on the problem area as fast.

Our research group has previously built an interactive tool to investigate swarm strategies. The Emergence Behavior Simulation Tool (EBST) is a bottom up approach to demonstrate the concept of emergent behavior. The user chooses all the small actions that a swarm will perform and the program shows a simulation of how the swarm behaves with the parameters that the user gave. The driving problem of one component of the tool is to get as many ants from one side of the river to the other by making bridges made out of ants. An example of the EBST program is shown in Figure 2. [3]

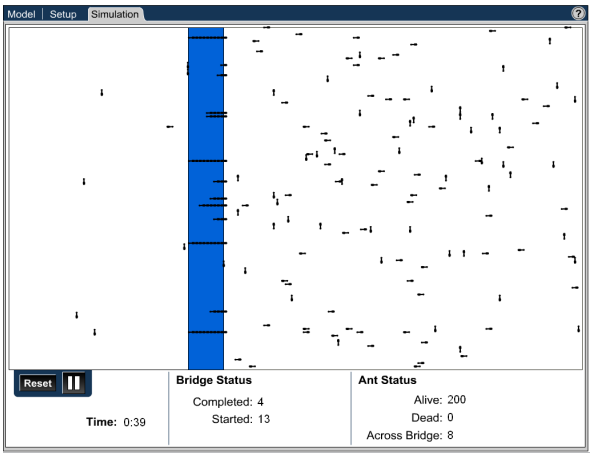

Figure 2. EBST tool, ants crossing the river

In this paper we describe building a tool that compliments the EBST by showing a human assisted swarm from a top-down perspective rather than the bottom-up perspective. This new tool is being developed to provide insight on how to use humans to help improve swarm based algorithms from a top-down perspective.

3 CHOOSING A PROBLEM

The tool that is being built is a jigsaw puzzle application that uses a swarm to solve the puzzle, with the ability for a human to assist the swarm. The human is able to give multiple high level directions to the swarm.

We chose a jigsaw puzzle as our primary application because humans solve jigsaw puzzles by looking at patterns and trends, which is what swarms have a very hard time doing. Some examples of strategies humans use when trying to solve a puzzle are: putting the edge pieces together, trying to connect similar colors together, and trying to fit similar connectors together. Humans have the ability to look at a jumble of puzzle pieces and quickly decide on an appropriate strategy. Someone would not start sorting a monochrome puzzle by color or look for corner pieces of a round puzzle. Humans can also change strategies dynamically, For example, they could switch to sorting by colors when the edges have all been assembled.

A picture that is mostly all one color is an example of a puzzle that may cause the swarm to thrash a lot before it will be able to find a solution. The swarm might choose to try and solve the puzzle by sorting it by color, which would not quickly lead to a solution. A human could see that the puzzle was mostly one color and could pass on this high level information to the swarm. This means that the swarm will not waste its time on a strategy that will not lead to a solution in a reasonable time frame.

4 HOW THE TOOL WORKS

4.1 Basics of the program

The tool is written in flash and designed so its functionality can be easily changed to run tests in multiple ways. When the program is loaded the user has the ability to choose what image they would like to load in, how many

agents they would like to run, and how fast the agents will move. The program has the ability to run with any image that has been previously split and placed into the images folder of the project. The program doesn't have the ability to split the images itself, but a program called TileMage Image Splitter splits the images into small jpeg files.[4] The images can be split into any number of pieces of any size where all the heights are the same and all the widths are the same. All images were also edited to have a red border around the outside of the picture to show the piece is an edge piece.

Figure 3. Full picture before split

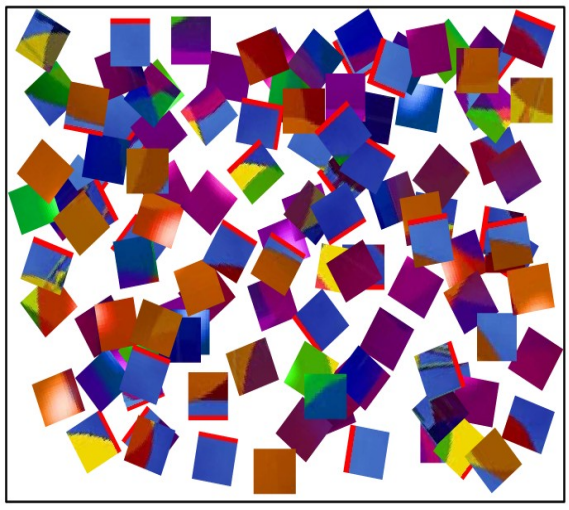

Figure 4. A split and randomized image

After all the sub-images are loaded and the user starts the program, the images randomize themselves on the screen within the area and choose a random orientation as shown in Figures 3 and 4. Each piece is then assigned a number for each of its four sides. For sides that are edges they are each assigned a -1 so that they cannot connect to any other piece. A side of a piece that can connect with another piece is given a unique random number from 1 to 999 that correspond with the side it connects to. An example of how the pieces are numbered is shown in Figure 5. The reason for this numbering system is to make the puzzle similar to a real jigsaw puzzle, in that there are sides that are similar to each other but can not fit together. For instance sides that have the first two digits being the same would be considered similar. These similarities will be used latter by the agents to find and sort pieces.

-1 -1 155 607	-1 155 132 432	-1 132 152 905	-1 152 -1 463
607 -1 394 401	432 394 841 238	905 841 247 198	463 247 -1 817
401 -1 854 59	238 854 489 641	198 489 10 397	817 10 -1 501
59 -1 142 87	641 142 970 123	397 970 62 285	501 62 -1 679
87 -1 746 -1	123 746 301 -1	285 301 91 -1	679 91 -1 -1

Figure 5. Demonstration of numbering system of pieces

Pieces of the puzzle will snap together when they are within 10° orientation of each other and within a small distance of where they will be snapped to. Once pieces are snapped together they will move as a group with other pieces it is connected to. The same is true when a connected piece is rotated. All the

pieces connected to the rotated piece will be rotated and be moved to the correct location in relation to its connected pieces. An example is shown in Figure 6.

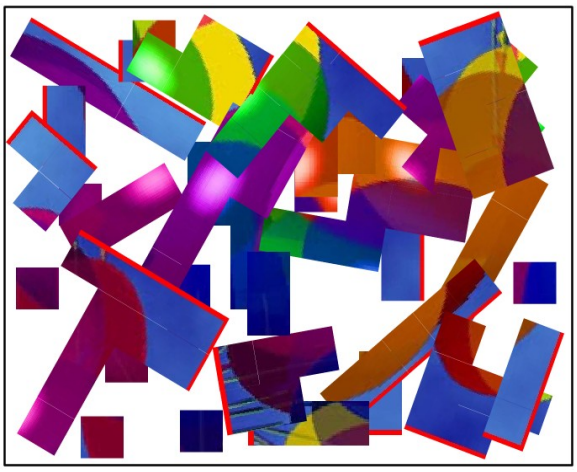

Figure 6. A partially solved puzzled

4.2 Agents

When agents are loaded into the program they are represented by yellow circles. The program can have any number of agents, each running independently of one another. This is shown in Figures 7 and 8. Each agent has actions and sensors that are used to solve the problem. The actions that the agents have include the ability to move randomly, start a pile of pieces, remember its location, and make its way back to its pile. The sensors an agent has are: the ability to see the "tags" of a piece it is holding or near, see the number given to each side of the piece it is holding or near, and ask if two pieces are a match to connect them.

Every agent also has strategies that it can use to work towards solving the puzzle. The current strategies that are available are: random movement, sort by edges, sort by tag, sort by sides, and a solver strategy. Multiple agents can exercise each strategy simultaneously, each will have their own pile depending on which strategy the agent is currently using. Some agents are tasked with sorting while others have the job of solving. Sorting the pieces into piles, that have similar attributes, helps the solver agents put pieces together.

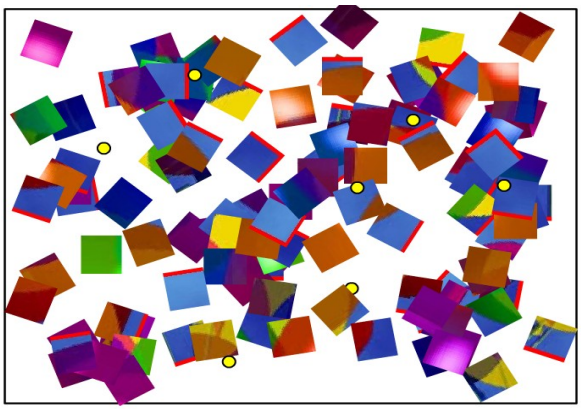

Figure 7. Demonstrating the program running with a small number of agents.

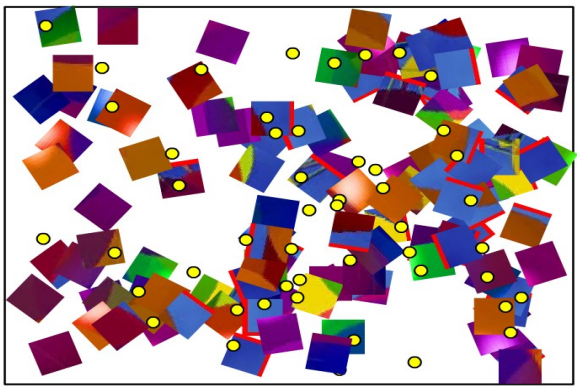

Figure 8. Demonstrating the program running with a large number of agents

The random movement strategy is the default strategy for every agent. Under this strategy the agent wanders around in a random direction for a random distance. How far an agent moves each time is based upon the speed that was initially set by the user before everything was loaded.

The edge sorting strategy has an agent make a pile of pieces that are all edge pieces. The agent starts off by moving randomly until it

finds a piece that has a -1 side (representative of a edge). It picks up this piece and immediately drops it and records the location. This location will be the place the agent makes its pile. The agent then goes back into a random movement until it finds another edge piece. When it does find another piece it picks it up and drags it back to its pile location. This process then repeats.

The tag sorting strategy has an agent make a pile of pieces that all have the same tag. A tag is an attribute of a piece such as its color. The tag of a piece needs to be preset before the program is started just like the splitting of an image. The tags are set by a text file that lists the attributes of each piece. This file is loaded into the program when the images are being loaded. A piece can have multiple tags associated with it. The agent starts off by moving randomly until it finds a piece. When it finds this piece it records its tag and location and becomes the start of the pile. The agent then returns to moving randomly until it finds another piece with the same tag as its pile and drags it back to its pile location. This then repeats. Figure 9 shows ten agents sorting by tag; in this case the tag that is set for each piece is the color.

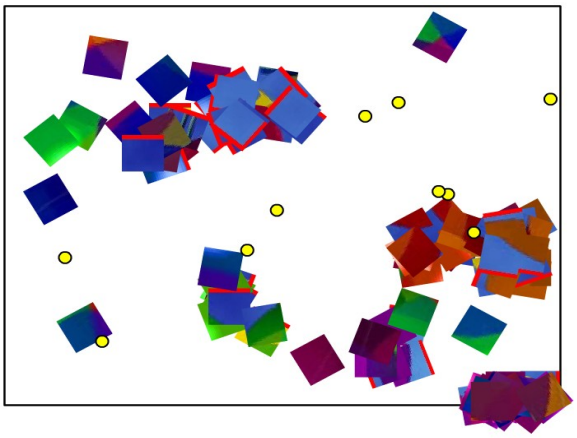

Figure 9. A partially solved puzzle where the agents are sorting the pieces by color.

The sides sorting strategy has the agents sort pieces into piles by edges that are similar to each other. Since the piece's sides are designated by numbers instead of random shapes like a traditional jigsaw puzzle, a way is needed to look at sides that are similar but not equal. The way this is done is to look at the first two digits of the sides of a piece but not the last. As in the Figure 5 showing the numbered sides, the sides 152 and 155 would show up as similar pieces. They would both be representative of a pile that would be based on the number 15. In the same way as the edge strategy and the tag strategy the piece wanders until it finds it first piece that starts the pile, and then continues to randomly walk until it finds a similar piece to drag back to its pile.

The solver strategy is where the agent's main task is to connect pieces together. When an agent has the solver strategy and picks up a piece it will try to find a match for this piece. It asks all the pieces within a communication radius, which we call a "yell," if any piece has a similar side, just like real jigsaw puzzles determining exact matching edges requires putting the pieces in the same location. The first agent that responds to the solver is the one the agent moves towards. When it reaches the location it checks to see if any sides of the piece the agent is holding are exact matches. If any are the agent sets the found piece to the correct spot and orientation and the two pieces are connected and dropped by the agent. However, if the agent makes it to the location and does not find a match it does another yell and moves to its new location. The agent keeps track of recently visited pieces also so it does not get stuck checking the same piece over and over. If a yell is done and the agent gets no responses, the agent will go into random movement and do a new yell each move.

4.3 Human Interaction

The human user will be able to interact with the swarm as it is running in many ways. The purpose of these interactions is to give high level directions to the swarm, since the swarm bases what it does on all low level information. The combination of the swarm's low level directions combined with the humans high level directions should make the puzzle solve faster than if the swarm was acting alone.

The first way the human can interact is by changing the global default strategy. The human can change it to any of the strategies that were listed in the agent section. There can be multiple global strategies set at the same time. The user gives a weight of what percentage of the swarm has each strategy. The next interaction a user can give is a strategy to a single agent. The final interaction a user can do is giving directions where pieces should be moved to. An example of giving directions would be if the user realizes that the picture has green grass and a blue sky. The user would realize that the sky should be at the top and the ground should be at the bottom while the swarm cannot Figure this out. The user would inform the swarm that they should move these pieces to the correct area of the puzzle.

5 STATUS

The project implementation is not yet complete. Currently any image that has been split up can be loaded, the randomizing of the pieces is implemented, the connecting of pieces is implemented, pieces moving and rotating as a group is implemented, and all agents are implemented. Each strategy that was listed is complete and the program can be run with each agent having a hard coded strategy. It is not required for agents to all use the same strategy.

The GUI interface for the human to interact with the swarm is not yet implemented. However, we are able to run the swarm with different strategies currently by hard coding the strategies of each agent in the swarm. We can also show how if the swarm uses some strategies it will not help it solve the puzzle. An example of this is attempting to solve a monochrome puzzle where the agents sort the puzzle by color. Also we can show that if the swarm were to choose to sort by edge they would endlessly do this due to the fact that they do not know when they are done sorting. The GUI interface needs to be implemented so that the strategies of the agents can be changed in real time.

A current version of the program, where all the agents are running the sort by tag (color is the current tag) algorithm, can be found at: http://stuweb.jcu.edu/jdengler10/jigsawMCURCSM.html.

6 FUTURE PLANS

The most important future plan for this project is to complete the program. We just need to implement the human interface that allows the human to interact in real time with the swarm.

Once this is complete, data collection on how much human interactions help the swarm to perform its task must be done. We will set up multiple different types of tests to show this, such as running the swarm by itself using different combinations of strategies. Then we will run tests using the same combinations but with a human interacting with the swarm.

7 Special Thanks

Thank you to the Huntington Foundation for providing the funding so this research could be done.

REFERENCES

[1] Palmer, D., Hantak, C., Kovacina, M., *The Impact of Behavior Influence on Decentralized Control Strategies for Swarms of Simple, Autonomous Mobile Agents*. In Workshop Proceedings of Biomechanics Meets Robotics, Heidelberg, Germany, 1999.

[2] Schneirla, T. C., *A unique case of circular milling in ants, considered in relation to trail following and the general problem of orientation*. American Museum Novitiates. The American Museum of Natural History, New York City, 1944. http://digitallibrary.amnh.org/dspace/bitstream/2246/3733/1/N1253.pdf.

[3] Kirshenbaum, M., Palmer, D., McCullick, P., Vaidyanathan, R., *Explaining Swarm Design Concepts Using an Interactive, Bottom-Up Simulation Tool*. Proceedings of Innovating Techniques in Instruction Technology, E- learning, E-assessment and Education, 2008.

[4] TileMage ImageSplitter. OrangeBright. http://tilemage.50webs.com/.