Generalization of the Self-Organizing Distinctive State Abstraction

Serra Kornfilt and Rachael Mansbach

May 13, 2010

Abstract

The Self-organizing Distinctive State Abstraction (SODA) algorithm was implemented on a new task. The original SODA algorithm was used in order to learn a navigation task, but the intent of this paper was to investigate whether the SODA algorithm was generalizable to a new task and to the real world. This paper investigates whether the SODA algorithm could be applied to teaching a robot to learn to pick up a grippable object, in simulation and in the real world. The result was that several aspects of the SODA algorithm were not well described in the original paper and were not easily translatable to a new task, particularly to one taking place in the real world.

1 Introduction

One pressing concern in robotics is the idea of generalization; that is, the utility of experiments and algorithms when moved from the precisely defined conditions of one experiment to another application which differs in some way from the old. The most useful and important algorithms are those that generalize well, as they can solve a class of problems rather than just one specific issue. Therefore, this paper discusses the use of the Self-organizing Distinctive State Abstraction (SODA) algorithm to learn a new task for a new application. Specifically, experiments were performed to test whether SODA could be applied to a new task and whether SODA could be applied to said task in the real world, rather than just in simulation, as the algorithm becomes much more useful and valuable if it is generalizable from simulation to reality and from task to task.

In the original two papers on SODA [4],[5], a robot in simulation learned to move from one place in a T-shaped world to another. This task is simple and does not involve many complicated actions on the part of the robot.

In the paper From exploration to Imitation [1], Dearden and Demiris describe the use of imitation and internal object models or IOMs in order to teach a real robot to learn to move a grippable object about. Though the way the task was learned, rather than the task itself, was the focus of the paper, the task itself is a simple, elegant one which can be easily set up in simulation or in reality. Therefore, this task was seen as an appropriate one for testing the generalizability of the SODA algorithm.

Simulation versus reality has always been a concern in robotics, as the real world is generally messy and therefore things that work in simulation do not always translate well to reality. This is a distinct problem in a field which, at the end of the day, is primarily real-world-based. Several papers have addressed this issue, such as [3]. In their paper *From Simulated to Real Robots*, Lund and Miglino experiment with the transition of a simulation-built neural network control system being transferred to a real environment using the Khepera robot. Although the experiment in this paper proceeded with a different robot, nonetheless the paper illuminated ways in which simulation and reality can be compared, and the relevance of using simulation to improve performance in the real world.

The real meat of the experiment, of course, is the SODA algorithm. The algorithm was implemented using aspects from both the older [4] and newer [5]; although using just the newer paper would have presumably produced better results, some of the aspects of the older paper were included in order to heighten simplicity.

The SODA algorithm has three distinct phases. [5],[4] First, the robot performs what is known as body-babbling; that is, it spends a certain number of time-steps performing random actions drawn from predefined primitive actions. As the robot babbles, it learns a Growing Neural Gas (GNG) over the sensor space that it sees. (see section 1.1)

Once the robot has learnt a GNG on its sensors, it has essentially divided the world into different regions; so, for instance, a robot which learned a GNG on its sensor inputs in a world where it could pick up and put down an object might learn that holding an object and not holding an object correspond to states in two different categories. The idea in SODA of having the robot learn a GNG on its sensor inputs is to enable the robot to understand its world prior to learning particular options that will eventually be used in learning the specific task itself. These options are behavioral routines that allow the robot to move between distinctive states. They are separated into two types: hill-climbing and trajectory-following options. Hill-climbing options basically try to get the robot's sensors as close as possible to the center of the current category as defined by the GNG, while trajectory-following options move the robot from one category to the next [4],[5]. For example, a robot might learn a hill-climbing option that got it closer to seeing something dead center in its camera, and it might then learn a trajectory-following option that allowed it to move from there to a state in which it was holding the object (so the trajectory-following option would consist of using its primitive actions to pick up the object, in this case). After the robot has learned hill-climbing and trajectory following options, it can move within an abstracted environment for which its behaviors become tailored, increasing its performance in comparison to any attempt it might have made to learn the task based off of the raw sensor readings it received prior to categorization and learning. The robot learns hill-climbing and trajectory-following options by means of a reinforcement learning algorithm, state-action-reward-state-action (SARSA) (see section 1.2) One hill-climbing option is learned for each prototype returned for the GNG; this allows the robot to be able to get itself into the center of any prototypical state. One trajectory-following option is learned for each prototype-primitive action pair, since then the robot can continually perform an action to take itself from one prototype into another.

While learning these options in the second stage, it becomes necessary to specify a more comprehensive state representation for learning. [5] uses a method called the Top-N state representation in which they measure the distance between a particular vector and all of the remaining prototypes, take the top n n indices with the smallest distance from the prototype and pool them into a single state representation with which they learn the hill-climbing and trajectory-following options.

In the last stage of SODA, the robot learns to perform the specific task using reinforcement learning on the options it has learned in the previous stage by combining them into higher-level actions in which trajectory-following options are followed until termination and then hill-climbing actions are taken in the new state. This last stage enables the robot to learn the specific task from a higher-level strategic standpoint in which the world has been categorized and abstracted relative to its perspective, and thus more suited to maximizing its performance based on its ability to navigate and manipulate its environment.

1.1 GNG

Growing Neural Gas is an algorithm that was first introduced by [2] as a method of categorization. It assumes as its input a set of n-dimensional vectors and attempts to learn a topological structure that matches the set. The set can be discrete or continuous. In essence, the GNG tries to learn a model such that if it is later given an input it can return a prototype; that is a the vector which is closest to that input in its model. In this way it discretizes continuous spaces as well as categorizing them.

1.2 SARSA

The basic idea behind reinforcement learning is to give the robot a reward for performing specific actions. The robot then learns a policy based on what state it is in by choosing an action based on the maximum expected reward for that action. The TD-learning algorithm, a precursor to SARSA, uses the maximum expected reward over all possible actions:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t) * [r_{t+1} + \gamma max(Q(s_{t+1}, a) - Q(s_t, a_t))] \quad (1)$$

where α is the learning rate and γ is the discount factor. The SARSA algorithm, on the other hand, uses only the reward expected for the next state given a particular action; that is, instead of learning a policy of the states, it learns a policy for the state-action pairs:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max(Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))] \quad (2)$$

2 Experimental Procedure

Our project revolves around two questions: first, will SODA work when applied to a task different from the one in [5] and, second, can SODA be implemented in the real world? The success of the first question is determined by whether SODA can be used to teach a robot in simulation to learn the new task, while the success of the second experiment is determined by whether SODA can be used to teach a robot in the real world a similar task.

The task to be investigated was chosen deliberately to be a simple one. A Pioneer robot was placed in a small environment containing two grippable objects, one red and one blue, and one green immovable object, and expected to learn to move the red grippable object in front of the green immovable object. In the real world, the red and blue objects were cubes made of Legos, while the green immovable object was a bright green drawer. The robot always learned in the controllable lighting conditions of a lab with the lights on and the blackout blinds down. The robot was contained to a small, trapezoidal environment, with the green object directly ahead of it and the red and blue objects beginning halfway between the robot and the green object, one at about a forty-five degree angle from the robot, the other at about a negative forty-five degree angle from the robot (see Fig 1). In simulation, the environment was similar, except that the red and blue objects were simulated pucks, the green immovable object was a green wall, and the environment was given a cone shape in order to help direct the robot toward its goal.

The Pioneer robot was chosen for this experiment because it is equipped with both a camera and a gripper, which give it the ability to both manipulate objects in its environment and to gather information about its surroundings through vision. It also has the ability to rotate and translate which is useful in interacting in its environment.

The Self-Organizing Distinctive State Abstraction (SODA) algorithm has been implemented on a single task, staying as true as possible to the authors' outline of the algorithm in [4] and [5]. For the first stage of SODA, the body-babbling stage, the robot learns a GNG over the primitive actions, which are move forward, move backward, turn left, turn right and pick up and put down (an object). Pick up and put down were hardcoded in order to simplify the robot's learning progress. In simulation, pick up represents the robot performing a store command and put down represents the robot performing a deploy command. The GNG phase was run for 100 timesteps, 250 timesteps and 500 timesteps (in simulation).

The perceptions the GNG used to learn in the simulated world were the results of applying red, green and blue color filters to the camera, which each returned a vector of 5 numbers: x_1, y_1, x_2 ,y_1, and area; however, the GNG was fed only the average x-value

$$\frac{x_1 + x_2}{2} \tag{3}$$

and the area. The GNG was also fed the states of the inner and outer break beams of the gripper. These return 1 if the beam is broken and thus indicate that there is an object in the gripper which could potentially be picked up. The GNG was also fed information from a stall sensor. The last input into the GNG was an indicator of whether the robot was holding an object, based on whether the robot attempted to perform a pick up action and ended up with its gripper closed. This was necessary as the simulated robot would often get confused after being unable to perform and record itself as having performed them even though it was unsuccessful. The indicator of whether the robot was holding an object was added both as a safety measure and to enable the pioneer to gain a better understanding of its environment. We placed all of the above information into a single 10-unit vector for the GNG to learn from.

The original idea of body-babbling was slightly modified in that instead of letting the robot move completely randomly, the robot would automatically pick up an object that triggered its inner or outer break beam. This was an attempt to cut down on the repetition of superfluous states and increase the likelihood that the robot would see a state in which the item had been picked up, since that would be likely to be quite an important state to see.

In the real-world environment, the primitive actions remained the same, but the pick up and put down actions now represented the robot fulfilling the commands to lower its gripper, close its gripper and raise its gripper or vice versa, respectively. In the real-world environment, it was quickly discovered that in comparison to the color precision of the red, green and blue objects in the simulated environment, the colored Lego blocks and drawer that we used had colors that, unsurprisingly, were not exactly red, green and blue and therefore the blobify filter was unable to work by simply setting it to look at pure red, pure green and pure blue. This problem was solved by the addition of a match filter for each color prior to adding the blobify filters, which filtered the image for the color given by the match filter. The experiment was also modified to account for the physical limitations the real robot had relative to its simulated counterpart. For example, the robot's camera can only go low enough to visually register objects in the gripper that trigger the outer break beam and not the inner. The stall sensor and the indicator for whether the robot is holding an object in its gripper were applied to the real-world robot in identical format as in their application to the simulated robot since these challenges were faced by both robots.

In both the simulated and the real-world application of the algorithm, these assembled vectors of values were fed into the GNG algorithm, which, as mentioned previously, by finding a topological structure that matched the given distribution of values as closely as possible, returns a series of vectors that classified the world into a set of representations that the robot could interpret. These model vectors were then used in the second stage of the SODA algorithm: the learning of the hill-climbing and the trajectory-following options. To learn these options, which are defined in [5] as a formal specification of a behavioral routine that can itself be used as an action or operator in a higher-level behavior, we used the reinforcement learning algorithm SARSA (state-action-reward-state-action). (see Introduction)

When learning hill-climbing, the robot learns an option for each prototype while in trajectory-following the robot learns an option for each prototype and primitive action pair. In implementing SARSA to learn these options, the Q-table states for both hill-climbing and trajectory-following are all of the possible combinations of the top-3 lists of prototypes that begin with the chosen prototype. The top-3 representation is a specific case of applying the top-N state representation methodology in which we calculate the distances between the chosen prototype and all other model vectors. The experiment used inverse distance [4]

$$f_j(y) = \frac{1}{(\|y - w_j\|)^z} \tag{4}$$

where y is the chosen prototype and the w_j's are the other prototypes, with $z = 2$ rather than the Gaussian mentioned in [5] in order to keep the implementation simple.

The reward function for learning the hill-climbing options is the same as in the newer Provost paper,

$$R_i^{HC} = c_{R1}\Delta f_i(y) - c_{R2} \text{ if not terminal} \tag{5}$$

$$R_i^{HC} = f_i(y) \text{ if terminal} \tag{6}$$

, , where f is the inverse distance function [5]; that is, a reward of the inverse distance if the robot is in its terminal or goal state, and a reward of the change in the inverse distance otherwise, balanced by several constants. (In our experiment, both c_{R1} and c_{R2} were set to 1). The reward function for learning the trajectory-following options is (likewise from the Provost paper)

$$R_{ij}^{TF}(y) = f_i(y) \text{ if not terminal} \tag{7}$$

$$R_{ij}^{TF}(y) = 0 \text{ if terminal} \tag{8}$$

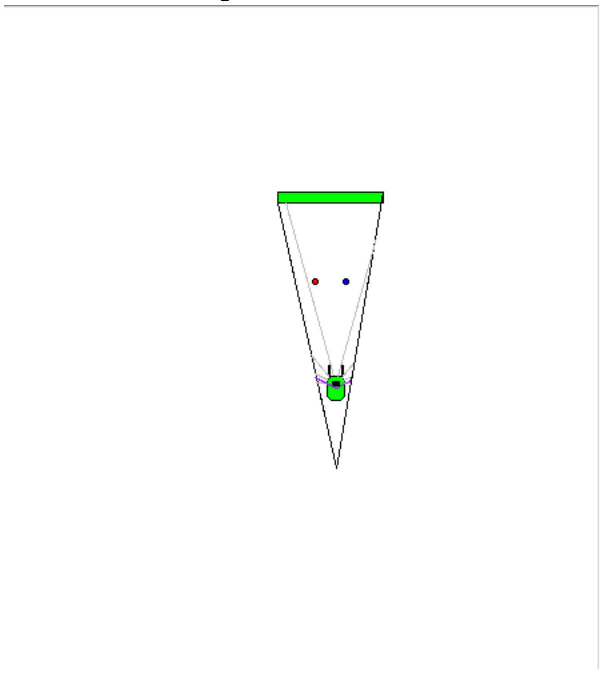
that is, a reward of the inverse distance if the robot is not in its terminal or goal state and a reward of 0 if it is. The terminal state for hill-climbing was defined to be when the current sensor readings were within a certain distance of the goal prototype. For the purposes of this experiment, the distance was defined to be two. The terminal state for trajectory-following was when the robot moved out of the current state into a new state.

The actions for the hill-climbing Q-table are move forward, move backward, turn left, turn right, pick up and put down, whereas the actions for the trajectory-following Q-table are move forward, move backward, turn left and turn right. As mentioned earlier, the robot learns a trajectory-following option for each prototype-action pair; the actions for the Q-table consist of the action plus some small corrective action taken from the other prototypes. Since one cannot have part of a pick up or put down, the primitive actions used to generate the actions for the trajectory-following Q-table could only include the aforementioned move forward, move backward, turn left and turn right.

The second phase of the algorithm, in which the robot learns the options, requires the robot to visit each prototype once for hill-climbing and four times for trajectory-following (one for each prototype-action pair, as mentioned before). Only after all of these requirements are satisfied does the robot move onto phase three, learning the specific task, in order to make sure that the robot has learned an option for all possibilities.

In order to ensure that the robot did see all the prototypes, the robot first learned hill-climbing on whatever prototype it happened to be in when the GNG finished. It then learned trajectory-following on this same prototype and ended up in another one and so on and so forth. How many times a particular prototype had been visited was kept track of in a dictionary indexed by the prototype number, for both hill-climbing and trajectory-following; so, for instance, if the robot that had learned a hill-climbing option and two trajectory-following options for a prototype, then the dictionaries would have stored a one in the hill-climbing visited dictionary and a two in the trajectory-following visited dictionary. If the robot entered a prototype which had nothing left to be learned (a one in the hill-climbing visited dictionary and a four in the trajectory-following dictionary), it would then simply take a random move to get into another

Figure 1: The simulated world in its initial set-up.



prototype. If the robot stalled, then it would be reset into a random location in the environment.

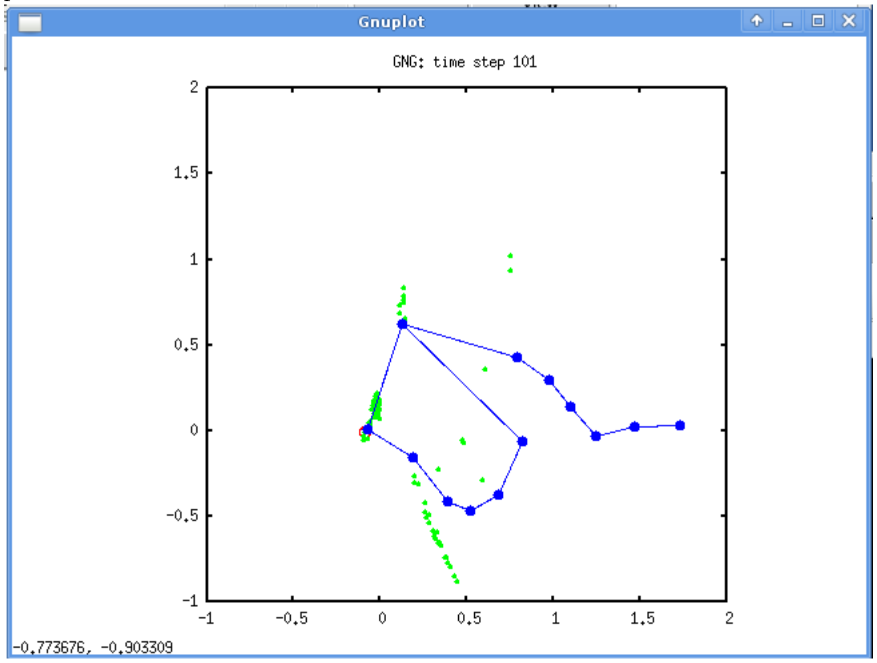
3 Results

The current set of results indicates that, when the GNG is run for 100 time-steps, although the algorithm can successfully proceed through the three stages outlined in the previous section, and can learn a GNG on the sensors, its limited number of prototypes prevent it from successfully learning a policy over which it can learn the specific task that we require. The reason is because with a small number of prototypes, the robot cannot distinguish from its starting state and its goal state. Currently, therefore, the only visualizations of the experiment are the 2D representations of the GNGs that it is learning (See Fig 2). There are no other results because there was no time to run them and debug longer runs.

4 Discussion

The most significant result from this investigation lies more in the attempts to implement SODA than in any of the results of running the algorithm. These attempts highlight several major flaws in the SODA algorithm. First of all is the problem of the

Figure 2: A two-dimensional visualization of the GNG returned by the body-babbling phase.



necessity of visiting all the prototypes in order to learn the options. [5] never clearly explains how this problem was addressed, and it seems difficult to come up with a really elegant solution. In simulation, in this experiment, the problem was addressed with a combination of random moves and hand-of-god teleportations, which, of course, would not work in the real world. Further, it leads to a very haphazard, hard-to-understand algorithm, which can easily lead to infinite loops, particularly if the GNG is not pruned properly, since the GNG starts with two random prototypes which may, for instance, never be visited again in the course of learning. During the initial phases of the GNG, other spurious prototypes may be generated that may actually be visited several times before being discarded but not erased, with the result that it is difficult to say for certain which prototypes should be pruned in order to ensure that no unphysical prototypes are passed into the second phase of SODA.

Another major problem was that while learning hill-climbing options, it was easy for the robot to accidentally overshoot and wind up in a new state, which threw off the learning algorithm, since such a state would not be in the Q-table. This problem was never addressed in [cite paper] and it did cause some real problems in terms of throwing off visited counts and such and deciding whether an option could be considered learned if it overshot at some point. Eventually for the purposes of expediency, the option was defined to be learned even if the learning algorithm exited due to an overshoot, because the alternative could again easily lead to infinite or semi-infinite loops if the robot always overshot in a particular hill-climbing option.

Clearly, from observations of the GNG (see Fig 2), the algorithm needs to be run for a larger number of time-steps than one hundred. The GNG has not entirely fitted to the data, and it can be seen that the robot is seeing perhaps two or three broad groups of prototypes spread out over a spectrum. Probably the spreading on the spectrum is due to the continuous nature of the changing sizes of the blobs of different colors, whereas the broadly different categories probably correspond to the discrete variables; in all likelihood stalled and not stalled would be most likely to be seen most often and therefore to create broadly different groups. Also, possibly, holding object and not holding object might conceivably create discrete groups.

4.1 Future Work

The main focus of any future work would be improving the accuracy of the results and manipulating the algorithm so that its individual components could be better specified. The primary concern that arose was that due to the small number of prototypes produced by the GNG for use in the second and third stages of the algorithm, the robot could not distinguish its starting state from its goal state, and could not learn a policy by which to complete the task. To increase the number of prototypes returned by the GNG in future work, the number of steps specified in self.limit could be increased so that that the first phase of the algorithm would have more time and more data on which to build topologies. Alternatively, the simulated world could be adjusted to encourage the discovery of prototypes when exploring, and have larger distinctions between states, thus making it easier for the robot to distinguish between them.

Due to the difficulties we had with implementing SODA in simulation, the second and third phases of the algorithm were not tested on the pioneer. As mentioned before, the paper did not address the issue of the robot visiting all the prototypes leading to a solution that was specific to the conditions allowed by the simulated environment. Future work with the algorithm with regards to the pioneer would include adapting the algorithm to enable the real-world robot to visit all of the states, thus allowing the algorithm to progress to the point of learning a policy and providing data to compare with the results of running the algorithm in simulation.

Another step, once the algorithm is improved, might be to change the body-babbling step to a hard-coded routine to allow the robot to see all the possible necessary states. On the one hand, this might allow the robot to learn more accurately the states that are actually important, but on the other hand, such supervision can be detrimental to a robot's learning capability.

References

[1] Anthony Dearden and Yiannis Demiris. From Exploration to Imitation: Using Learnt Internal Models to Imitate Others. *Proceedings of the Society for the Study of Artificial Intelligence and Simulation of Behavior*, 2007.

[2] Bernd Fritzke. A Growing Neural Gas Learns Topologies. *Advances in Neural Information Processing Systems*, (7), 1995.

[3] Henrik Hautop Lund and Orazio Miglino. From Simulated to Real Robots. *Proceedings of IEEE, Third International Conference on Evolutionary Computation*, 1996.

[4] Jefferson Provost, Benjamin J. Kuipers, and Risto Miikkulainen. Self-organizing Distinctive State Abtstraction for Learning Robot Navigation. *University of Texas Artificial Intelligence Lab Technical Report*, 2005.

[5] Jefferson Provost, Benjamin J. Kuipers, and Risto Miikkulainen. Self-organizing Distinctive State Abstraction Using Options. *Proceedings of the Seventh International Conference on Epigenetic Robotics*, 2007.