

SAN-RL: Combining Spreading Activation Networks and Reinforcement Learning to Learn Configurable Behaviors

Daniel M. Gaines

Daniel.M.Gaines@jpl.nasa.gov
Jet Propulsion Laboratory

Mitch Wilkes, Kanok Kusumalukool, Siripun Thongchai, Kazuhiko Kawamura, John White

{Mitch.Wilkes, Kanok.Kusumalukool, Siripun.Thongchai,
Kazuhiko.Kawamura, John.H.White}@vanderbilt.edu
Department of Electrical and Computer Engineering
Vanderbilt University

ABSTRACT

Reinforcement learning techniques have been successful in allowing an agent to learn a policy for achieving tasks. The overall behavior of the agent can be controlled with an appropriate reward function. However, the policy that is learned will be fixed to this reward function. If the user wishes to change his or her preference about how the task is achieved the agent must be retrained with this new reward function. We address this challenge by combining Spreading Activation Networks and Reinforcement Learning in an approach we call SAN-RL. This approach provides the agent with a causal structure, the spreading activation network, relating goals to the actions that can achieve those goals. This enables the agent to select actions relative to the goal priorities. We combine this with reinforcement learning to enable the agent to learn a policy. Together, these approaches enable the learning of a *configurable* behaviors, a policy that can be adapted to meet the current preferences. We compare the approach with Q-learning on a robot navigation task. We demonstrate that SAN-RL exhibits goal-directed behavior before learning, exploits the causal structure of the network to focus its search during learning and results in configurable behaviors after learning.

1. INTRODUCTION

We often have different preferences in how a robot should complete a task. In many cases, our preferences may change each time we ask the robot to complete the task. For example, a robot in an office environment may be responsible for delivering mail and cleaning up trash. Under normal conditions, it may be appropriate for the robot to combine these activities so that it picks up trash on its way to deliver mail. However, if an urgent package arrives, our priorities may change and we would prefer that the robot neglect its trash collecting duties in favor of promptly delivering the package.

There have been many approaches using reinforcement learning^{3,10} that allow a robot to learn a behavior to achieve some task, however, these approaches assume a fixed set of preferences. For example, if we gave equal preference to mail delivery and trash removal the robot could use one of these existing techniques to learn an optimal policy for collecting trash while delivering mail. However, if we later changed our preference, giving a higher priority to mail delivery, we would have to retrain the robot with this new reward function. This is impractical if we change our preferences frequently.

We address this challenge by developing an approach called SAN-RL which enables the learning of *configurable* behaviors. That is, we want to learn behaviors that can later be configured to the user's current preferences for completing a given task. Our solution is to combine two existing techniques. The first is the use of a Spreading Activation Network⁴ (SAN) to enable a robot to select plans to achieve goals with different values associated with them. On its own, a spreading activation network provides the capability of combining deliberative and

reactive planning and, with appropriate parameter settings, enables a robot to select actions that make trade-offs among goals with different values. However, setting these parameters can be tricky and may require a lengthy trial and error process. Therefore, we integrate spreading activation networks with a Reinforcement Learning (RL) technique that can learn the appropriate parameters based on the reward the robot receives while carrying out the tasks.

To use SAN-RL, the user must create an initial spreading activation network which describes the behaviors the robot has along with the pre-conditions for using each behavior and the effects the behavior will have on the world. Next, the SAN-RL algorithm is used to allow the robot to learn appropriate parameters for the spreading activation network for completing tasks under nominal preference conditions (i.e. all goals have equal value). At this point, the robot can be configured with new preferences by specifying different values on the reward. It will use its learned knowledge adjusted by the new goal weights to find an appropriate plan for completing the task.

We evaluate the effectiveness of the SAN-RL approach in a simulated environment comparing it with a standard reinforcement learning technique for learning robot behaviors. The results demonstrate several advantages of SAN-RL:

- goal-directed before learning: even before learning, the agent uses the causal structure of the network to make progress toward the goal.
- exploits causal structure of the spreading activation network to focus search during learning: the agent does not spend time learning the value of state, action pairs for states in which the action cannot be executed.
- configurable after learning: after learning, the agent can adapt its behavior according to the current preferences among the goals.

2. BACKGROUND

Our approach to learning configurable behaviors relies on two bodies of previous work. Spreading activation networks provide the basis for selecting appropriate behaviors for completing a given task. To perform well, parameters of the spreading activation Network must be hand tuned. Reinforcement learning techniques enable a robot to automatically learn a policy that will maximize its expected reward.

We provide a brief overview of both of the areas below. While both of these areas perform the same task, namely defining a policy for accomplishing tasks, they have distinct advantages. Spreading activation networks can select actions to achieve goals with different priorities while reinforcement learning automatically learns a policy. In the next section we will describe how these approaches were combined in the SAN-RL approach to incorporate the advantages of both.

2.1. Spreading Activation Networks

Maes introduced the concept of spreading activation networks as an action selection mechanism capable of combining deliberative and reactive planning^{1,4,5*}. A spreading activation network consists of competency modules (e.g. primitive behaviors the robot can perform) and condition nodes representing state information. Figure 1 shows an example. Each competency module is linked to the conditions that must be satisfied before the module can be executed (its pre-conditions) and the condition nodes that will become true after the module has successfully completed (its post-condition). Using this technique, an autonomous agent can use predictive models of its actions to look-ahead and plan a course of action that will achieve its goals. At the same time, it considers opportunistic actions that are possible simply because the current state satisfies the pre-conditions of some module. The values of the condition nodes are continually updated so that if things do not go as expected, the plan will be modified.

The action selection algorithm is based on activation and inhibition of competency modules based on the current state of the world and the goals the agent is suppose to achieve. A competency module receives activation from a goal condition if the goal is one of the post-conditions of the module. For example, in Figure 1, if condition

*Maes provides a complete description of the spreading activation network algorithm in.⁴

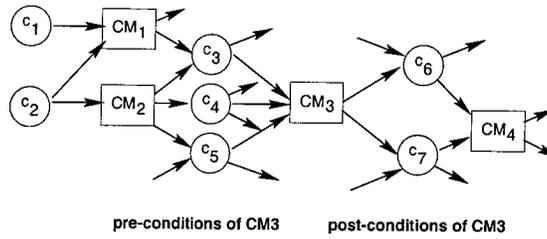


Figure 1. An example spreading activation network.

c_6 is a goal, then competency module CM_3 would receive activation. A competency module receives activation from the current state for each of its pre-condition nodes that are true in the current state. Thus, if condition c_2 is true in the current state, then both competency modules CM_1 and CM_2 would receive activation from the state.

A given competency module CM_a spreads a portion of its activation backward to some competency module CM_b if CM_b achieves one or more of CM_a 's pre-conditions. Module CM_a spreads activation forward to module CM_b if CM_a achieves pre-conditions of CM_b that are not true in the current state. In the example, competency module CM_3 would spread activation backward to competency modules CM_1 and CM_2 . Similarly, module CM_1 and CM_2 would spread activation forward to CM_3 . Modules may also inhibit one another by taking activation away. Module CM_a inhibits module CM_b if an effect of CM_b is to negate one or more of its pre-conditions. Finally, a module is also inhibited if it has an effect that would undo a goal that has already been achieved.

A spreading activation network contains several parameters that determine how it will behave. In particular, there are separate parameters that specify how much activation a competency module receives from goals, from the state and from other modules. A global activation threshold specifies how high a module's activation must be before it can become a candidate for execution.

Action selection proceeds as follows. At each time step, the activation for each module is computed as described above. If a competency module has all of its pre-conditions satisfied by the current state, then it is executable. The algorithm selects the executable competency module with the highest activation. If its activation is above the threshold, then it is selected. Otherwise, the threshold is reduced by a pre-determined amount (e.g. 10%) and the process repeats.

In its original form, spreading activation networks treat all goals as equal. We modified the algorithm slightly by adding weights to goals. The weight determines how much activation a competency module receives from a goal. A higher weight results in more activation. This allows a user to specify priorities among the goals.

With careful adjustment of the spreading activation parameters, the robot can perform well on tasks. However, it can be a tricky, trial and error process to adjust the parameters appropriately. Thus, we are interested in machine learning techniques that can assist in refining the robots behavior given an initial spreading activation network.

2.2. Reinforcement Learning

Reinforcement learning is a general class of problems in which an agent learns through interaction with the environment, typically through a (possibly delayed) reward that it receives based on the sequence of actions that it takes and/or the states through which it travels. The objective of the learner is to find a policy (a mapping from a state in the world to an action to take in that state) that will maximize the expected reward.¹⁰

Q-learning is a popular approach to reinforcement learning.¹¹ In Q-learning, the agent learns a value function, Q , that predicts the value that each of its actions will have for every state. The Q function is initialized arbitrarily (e.g. setting all values to 0). At each state s , the agent selects an action with the highest value $Q(s, a)$. It takes the action, observes the new state s' and receives a reward r . It uses this information to update the Q function as follows:

$$Q(s, a) = Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

where γ is a discount factor and α is the learning rate.¹⁰

The best policy can be found by taking the action with the highest Q value for each state the robot is in. To guarantee convergence on the optimal policy, it is necessary to include exploration of actions other than the action with the highest Q value during learning. One approach is to use an ϵ -greedy policy in which the action with the highest Q value is selected with probability $1 - \epsilon$ and a random action is selected, otherwise.

Q-learning is an effective approach to learning a policy, but the resulting policy is fixed for a particular reward function. If we change the priorities on the goals, we will have to re-train the robot in order to modify the policy. In the next section we describe an approach for combining spreading activation networks and reinforcement learning to address this limitation.

2.3. Using Reinforcement Learning in Behavior-Based Robotics

Our approach to applying reinforcement learning to robotics is based on prior work by Mataric.⁷ Mataric's approach is to create a set of *basis behaviors* that represent the robot's primitive capabilities that can be synthesized into more complex behaviors.⁶ She used reinforcement learning techniques to allow a robot to learn when it should use the basis behaviors to accomplish some task.

Spreading activation networks also make use of the concept of basis behaviors. Each competency module is a basis behavior and the spreading activation algorithm is used to select which basis behavior to activate.

3. SAN-RL: COMBINING SPREADING ACTIVATION NETWORKS AND REINFORCEMENT LEARNING

Our objective is to combine spreading activation networks with reinforcement learning to allow a robot to learn a policy for selecting among its competency modules to achieve its goals and to allow a robot to adapt its behavior when priorities among goals are changed without requiring retraining. To achieve this goal we modified the spreading activation network technique to allow a value function to be learned during execution. We also modified the spreading activation algorithm to include activation from the value function and to allow a priority to be associated with each goal.

3.1. Goal Priorities

Priorities enable a user to specify preferences on how they want the robot to complete a task. Our intention is that, whenever the robot has a choice regarding the course of action it should take, it should try to find a course of action that will maximize the sum of the priorities of the goals that will be achieved through those actions.

This was a fairly simple modification to the algorithm. In its original form, the algorithm gives competency module x at time t activation from the goals as follows:

$$\text{input_from_goals}(x, t) = \sum_j \zeta \frac{1}{|A(j)|} \frac{1}{|a_x|}$$

where ζ is the amount of activation injected by a goal[†], $A(j)$ is the set of competency modules that achieve goal j and a_x is the set of post-conditions for competency module x .

We modified this by including a weight w for each goal:

$$\text{input_from_goals}(x, t) = \sum_j w_j \zeta \frac{1}{|A(j)|} \frac{1}{|a_x|}$$

[†]Maes used γ to represent the amount of activation from a goal. We have used ζ to avoid conflict with the γ used as a discount factor in Q-learning.

Thus, a module that achieves a goal with a higher priority will get a higher activation than another module that achieves a lower priority goal. In turn, the modules that achieve pre-conditions for the former module will get more activation.

3.2. Value Function

The next step is to inject activation into a competency module in such a way that the agent can take its expectation of the value that competency module will have in the current state into consideration when selecting a module to activate. This requires modification to the action selection algorithm to allow the agent to explore the space of actions and learn the expected values as well as modification to the spreading activation algorithm to include activation from the learned value function.

The original action selection algorithm of a spreading activation network selects as winner the executable competency module with the highest activation level exceeding some threshold. To enable the agent to explore the possible actions, we modified this algorithm along the lines of an ϵ -greedy policy. With probability $1 - \epsilon$ the winning module is activated. With probability ϵ some other executable action model is selected at random. This is a slight variation on a typical ϵ -greedy policy as it takes the executability of a competency module into account and will never select an unexecutable module.

For similar reasons, we were not able to use the typical Q-learning update to learn a value function as the update includes the term: $\max_{a'} Q(s', a')$. This type of update is referred to as an *off-policy* update because the update is independent of the actual actions selected by the policy. In this case the update is based on the best action that could be taken, not necessarily the action that was actually taken. While an off-policy update has its advantages, it is not appropriate for SAN-RL. The problem is that not all actions are *executable* in a given state. A spreading activation network will never attempt to execute an action in a state in which its pre-conditions are not satisfied. Thus, it will never update the Q function for these state, action pairs and, as a result, the values for these entries will not be valid.

Instead, we use an on-policy update which updates the value function based only on the actions that were actually used during an episode[‡]. The approach we take is similar to that used in eligibility traces.^{2,9} During an episode, we keep track of each state, action pair that was observed. At the conclusion of an episode we receive a reward r . For each state, action pair (a, s) in the history we update its value with:

$$Q(s, a) = Q(s, a) + \alpha(\gamma^i r - Q(s, a))$$

where α is the learning rate, γ is a discount factor and i is the position of this station, action pair in the history relative to the end of the history (in other words, early decisions in the history receive a higher discount than later decisions).

The final change was to modify the spreading activation to include activation from the value function. The original algorithm has constants to indicate how much activation a module can receive from the state, from the goals and from other modules. We added a new constant, ω , which is the amount of activation a module can receive from the value function. At time t , each module, x , will receive a percentage of this value based on the reward the agent expects to receive by using that module in the current state, s :

$$\text{input_from_value_function}(x, t) = \omega Q(s, x)$$

4. APPLYING SAN-RL TO A NAVIGATION TASK

We evaluated the performance of SAN-RL by applying it to a navigation task. We compared SAN-RL's ability to learn a policy with a typical reinforcement learner. We then evaluated the ability of SAN-RL to adapt its policy to different preferences among the goals.

[‡]Another possibility would be to modify the Q value update to take into account the executability of actions.

4.1. The Learning Task

Figure 2 shows the learning task used to evaluate SAN-RL. The task includes a robot, a target location and an enemy. The enemy is stationary. The robot must get to the target location without being killed by the enemy. The enemy is stationary and fires at the robot at 2 second intervals whenever the robot is within range. The probability of the enemy hitting the robot is inversely proportional to the robot's distance from the center of the enemy.

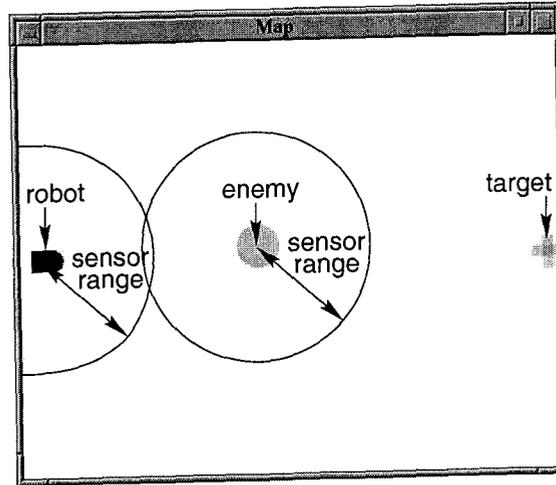


Figure 2. A navigation task.

The robot begins with a health of 1 and loses 0.6 from health when it is hit. A hit causes the robot to move more slowly. If the robot's health falls below 0 (i.e. it is hit twice) then it is dead and the episode ends.

The robot always knows its current heading and position in the map (e.g. it has a compass and GPS receiver), the target location and the position of the enemy whenever the enemy is within its sensor range.

Figure 3 shows the spreading activation network created for the robot. The network consists of two competency modules (or basis behaviors):

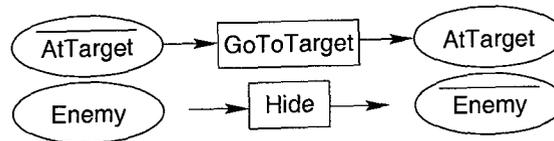


Figure 3. Spreading activation network for navigation task.

GoToTarget: This behavior is applicable when the robot is not currently at the target. It turns the robot so that it faces the target and then moves the robot forward until the target is reached. The effect is to get the robot to the target.

Hide: This behavior is applicable when the enemy is within range. The behavior turns the robot in a direction that will take it away from the enemy but is close to its current heading. The effects is to move the robot so that the enemy is no longer in range.

The network uses the following condition nodes:

AtTarget: true if the robot is at the target location.

Enemy: true if an enemy is within the sensor range of the robot.

Thus, there are four possible states for the robot. The goal of the learning task is to learn which behavior to use in each of these states. While this is a fairly simple learning problem for reinforcement learning given the small search space, it is sufficient to demonstrate the advantages of SAN-RL. The robot in the task has two objectives, to get to the goal quickly and to stay clear of enemies. Our objective with SAN-RL is for the robot to learn to use its behaviors to accomplish this task when these two goals are given equal priority. We then wish to show that, if these priorities are modified (and thus we change the reward function to favor one goal over another), the robot will change its behaviors, without retraining, to achieve the higher reward. It is also important to note that the behavior-based approach to reinforcement learning plays an important role in reducing the search space.⁶ Had we used a more traditional representation of the problem where each state consists of x, y coordinates, this would be a much more difficult problem.

4.2. Evaluating SAN-RL’s Ability to Learn a Policy

To evaluate the approach, we ran the SAN-RL algorithm and a Q-learning algorithm on the above problem. Both versions used the same set of state conditions and behaviors, although the Q-learner was not given the network structure indicating the pre-conditions and post-conditions of the behaviors. If the Q-learning algorithm selected a behavior that was not executable for the current state, the behavior does nothing for 1 second and then halts, signaling the Q-learner that it must select another action. For SAN-RL, we set the priority for both goals to the same value (1.0). For both SAN-RL and Q-learning we used: $\gamma = 0.9, \alpha = 0.1$ and $\epsilon = 0.1$.

The reward function used was:

$$\begin{aligned} \text{healthReward} &= \max(-1, 0 - \text{health}) \\ \text{timeReward} &= \max\left(-1, 1 - \frac{\text{time}}{\text{min time}}\right) \\ \text{reward} &= \text{healthReward}^{0.5} + \text{timeReward}^{0.5} \end{aligned}$$

where health refers to the robot’s health at the end of the episode and time is the time in milliseconds it took to reach the target. We calculated min time by running the robot without an enemy on a path directly from the start to the target. We found that 8 seconds was a good lower bound on the time to reach the target.

We ran both algorithms for 200 training episodes. At the end of each episode, when either the robot reached the target or was killed, we recorded the reward received, reset the robot’s health to 1 and put the robot back at the starting point. After each training episode we fixed the policy and weights and ran the robot on 10 test episodes to calculate the average reward received.

Figure 4 shows the results. Each data point in the figure represents the average reward received during testing over 10 training episodes (and thus, over 100 testing episodes). SAN-RL and Q-learning have similar performance. Both find the optimal policy in about the same time.

The learning task essentially came down to deciding which behavior to select when an enemy is present. If an enemy is not present, the Hide behavior is not executable, and SAN-RL never considers its use. Q-Learning tried it, but quickly learned that it was not valuable for achieving a reward. Thus, both systems experimented with the two policies shown in Figure 5. In Figure 5 (a), the robot runs straight past the enemy toward the goal. This policy will maximize the time reward but puts the robot at risk of getting shot. In contrast, the policy in Figure 5 (b) uses the Hide behavior whenever the enemy is present. While this policy receives less reward for time, it keeps the robot safe, and therefore gets a high health reward.

Although the learning task was fairly simple, Figure 4 shows that both systems required about 50 training episodes before they found the best policy. This is because, under the reward function stated above, both policies are very close in the reward that they will earn. If the robot is lucky, it may go several episodes with the policy in Figure 5 (a) without getting shot. This gives the robot the highest reward and accounts for the peaks in the

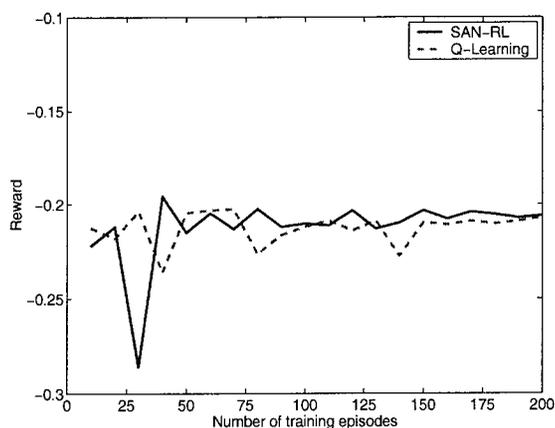
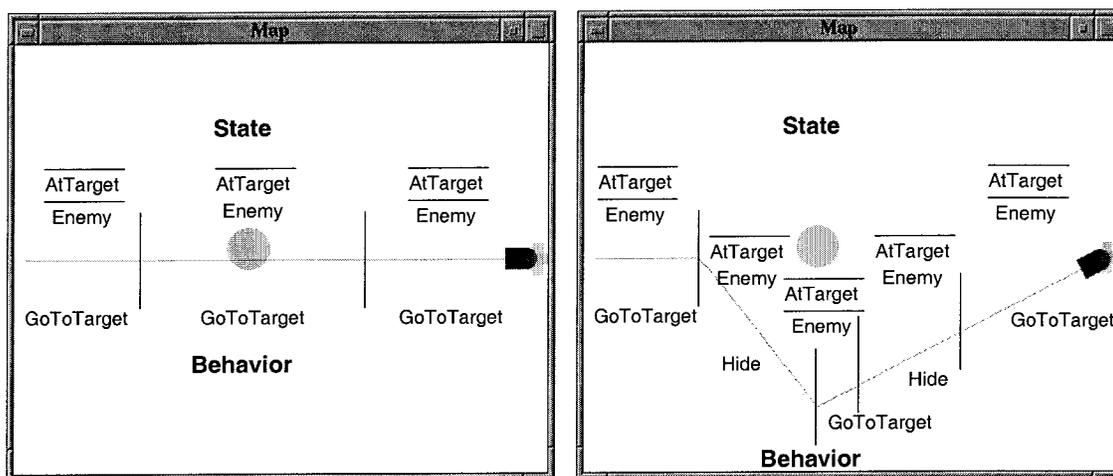


Figure 4. Results comparing SAN-RL and Q-learning.



(a) A policy using only GotToPoint

(b) A policy using GotToTarget and Hide

Figure 5. Policies for the learning task.

early episodes in Figure 4. However, this policy is eventually proven dangerous as the robot is often killed by the enemy, which returns the lowest reward and accounts for the sharp dips in Figure 4.

Overall, SAN-RL and Q-learning appear to perform equivalently on this task. We did note that Q-learning spends time considering the value of unexecutable actions (e.g. using Hide when an enemy is not present). This did not adversely affect its performance in this example, but we anticipate that it would detract from its performance in more complicated domains.

Another advantage of SAN-RL is that is goal-directed, even before learning. It can use the causal structure of the spreading activation network to select behaviors that will achieve the goal. This will not necessarily be the best action, but it does lead the planner toward the goal.

4.3. Evaluating SAN-RL's Ability to Transfer Learned Knowledge to New Problems

We would like the robot to be able to transfer the knowledge that it has learned to similar problems. One of the advantages of taking a behavior-based approach to reinforcement learning is that the states are represented relative to the robot rather than a global reference point. For example, we represent whether or not the robot is near an enemy rather than the robot's specific x, y position. Therefore, the policy based on this representation can be applied to new problems.

Figure 6 shows an example. We took the policy obtained from the final learning episode and used it on this map. The robot was able to get to the target point while avoiding the enemies.

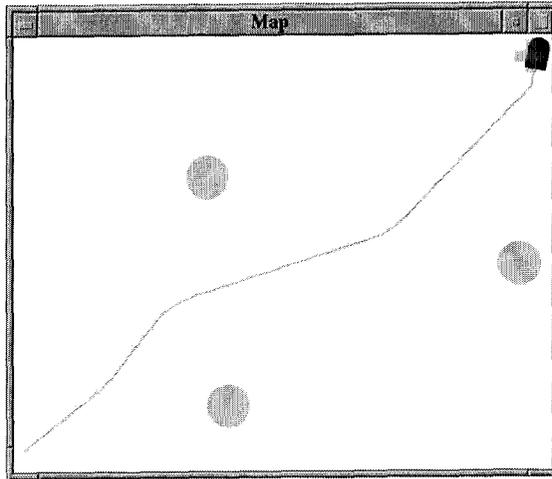


Figure 6. Applying previously learned policy to a new problem.

This ability to transfer learned knowledge comes from the representation of the state and not from a specific property of SAN-RL. Thus, the policy obtained from Q-learning results in the same behavior shown in Figure 6.

4.4. Evaluating SAN-RL's Ability to Adapt a Policy

Our final, and most important, evaluation was to see what would happen if we change the priorities of the goals. With equal priority given to getting to the target and keeping clear of enemies, the SAN-RL algorithm exhibits the behavior shown in Figure 5 (b). In this case, it has learned that it is best to avoid the enemies, using the hide behavior whenever an enemy is in range. If we give a higher priority to getting to the target (a 0.1 difference for example) the robot exhibits the behavior in Figure 5 (a). It runs straight to the goal, risking fire from the enemy.

This demonstrates that the robot is able to adapt its behavior as the user preferences alter. It did this without needing to be retrained with the new reward function.

5. IMPLEMENTATION OF SAN-RL ON A MOBILE ROBOT

We are in the process of implementing SAN-RL on an ATRV-Jr mobile robot. Our robot is equipped with Sonar, Lida DGPS, an inertial navigation system and a camera. We have also added sensors from a laser tag game so that we can simulate the robot being shot by enemies as shown in Figure 7.

We have developed a user interface to enable the design of spreading activation networks. Figure 8 shows the interface as well as the network we are creating to evaluate SAN-RL. Competency modules for the network have been implemented using our Intelligent Machine Architecture.⁸

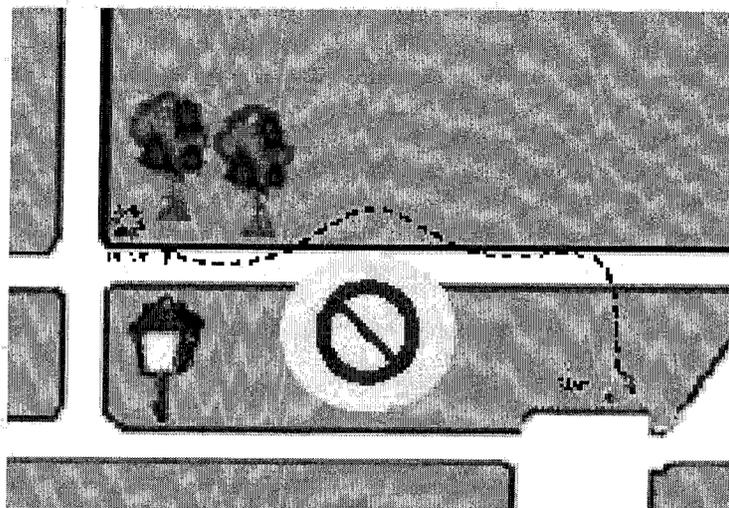


Figure 9. Behavior exhibited by the robot using the SAN from Figure 8.

will be to implement RL and evaluate its ability to enable the robot to learn skills and adapt them to changing priorities.

6. CONCLUSION

Our preferences in how a task should be carried out change over time. Therefore, the behaviors that an agent learns should be adaptable, or configurable, to the current set of preferences. We have presented a technique called SAN-RL that combines spreading activation networks and reinforcement learning to address this challenge. The spreading activation network provides the agent with a causal structure relating goals with the actions that can achieve them. This lets the agent select actions relative to the priorities of the goals. Reinforcement learning is used to find a policy for selecting among these actions when all goals are weighted evenly. The resulting behavior can be configured by setting appropriate weights to the goals without the need for retraining.

The current implementation has several limitations that will be addressed in future work. Currently, all condition nodes are boolean. We would like to use continuous variables and have the agent learn which values are appropriate for each behavior. We use a simple lookup-table to represent the value function. As the complexity of the task and state representation grows we will consider using function approximation techniques. Finally, we will apply this technique on a real mobile robot.

Acknowledgments

The work described in this paper was performed at Vanderbilt University, sponsored by the Defense Advanced Research Projects Agency grant DASG60-99-1-0005 issued by the U.S. Army Space and Missile Defense Command. The writing and publication of this paper was supported by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

1. S. Bagchi, G. Biswas, and K. Kawamura. Task planning under uncertainty using a spreading activation network. *IEEE Transactions on SMC-Part A: Systems and Humans*, 30(6):639–650, November 2000.
2. T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6, 1994.
3. Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

4. Pattie Maes. How to Do the Right Thing. *Connection Science*, 1(3):291–323, 1989.
5. Pattie Maes. A bottom-up mechanism for action selection in an artificial creature. In S. Wilson and J. Arcady-Meyer, editors, *From Animals to Animats: Proceedings of the Adaptive Behavior Conference*. MIT Press, February 1991.
6. Maja Mataric. Designing emergent behaviors: From local interactions to collective intelligence. In H. Roitblat J-A. Meyer and S. Wilson, editors, *Proceedings of From Animals to Animats 2, Second International Conference on Simulation of Adaptive Behavior*, pages 432–441. MIT Press, 1992.
7. Maja J. Mataric. Reward functions for accelerated learning. In William W. Cohen and Haym Hirsh, editors, *Proceedings of the Eleventh International Conference on Machine Learning*, pages 181–189, San Francisco, 1994. Morgan Kaufmann.
8. R. Pack, M. Wilkes, G. Biswas, and K. Kawamura. Intelligent machine architecture for object-based system integration. In *Proceedings of the 1997 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Tokyo, June 1997.
9. Richard S. Sutton. Learning to predict by the method of temporal difference. *Machine Learning*, 3:9–44, 1988.
10. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 1998.
11. C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.