7-12-2008

Scaling Ant Colony Optimization with Hierarchical Reinforcement Learning Partitioning

Erik J. Dries
Air Force Institute of Technology

Gilbert L. Peterson
Air Force Institute of Technology

Recommended Citation

Erik J. Dries and Gilbert L. Peterson. 2008. Scaling ant colony optimization with hierarchical reinforcement learning partitioning. In Proceedings of the 10th annual conference on Genetic and evolutionary computation (GECCO '08). Association for Computing Machinery, New York, NY, USA, 25–32. https://doi.org/10.1145/1389095.1389100

Scaling Ant Colony Optimization with Hierarchical Reinforcement Learning Partitioning*

Erik J. Dries and Gilbert L. Peterson
Air Force Institute of Technology
Department of Electrical and Computer Engineering
2950 Hobson Way, Bldg 640
Wright-Patterson AFB, OH 45433
{erik.dries, gilbert.peterson}@afit.edu

ABSTRACT

This paper merges hierarchical reinforcement learning (HRL) with ant colony optimization (ACO) to produce a HRL ACO algorithm capable of generating solutions for large domains. This paper describes two specific implementations of the new algorithm: the first a modification to Dietterich's MAXQ-Q HRL algorithm, the second a hierarchical ant colony system algorithm. These implementations generate faster results, with little to no significant change in the quality of solutions for the tested problem domains. The application of ACO to the MAXQ-Q algorithm replaces the reinforcement learning, Q-learning, with the modified ant colony optimization method, Ant-Q. This algorithm, MAXQ-AntQ, converges to solutions not significantly different from MAXQ-Q in 88% of the time. This paper then transfers HRL techniques to the ACO domain and traveling salesman problem (TSP). To apply HRL to ACO, a hierarchy must be created for the TSP. A data clustering algorithm creates these subtasks, with an ACO algorithm to solve the individual and complete problems. This paper tests two clustering algorithms, k-means and G-means. The results demonstrate the algorithm with data clustering produces solutions 20 times faster with 5-10% decrease in solution quality due to the effects of clustering.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms

*This work was supported in part through AFRL/SNRN Lab Task 06SN02COR from the Air Force Office of Scientific Research, Lt Col Scott Wells, program manager. The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

Keywords

Ant Colony Optimization, Swarm Intelligence, Hierarchical Reinforcement Learning

1. INTRODUCTION

Many real world problems, including scheduling, and planning, require optimized solutions. Optimization techniques provide a way to search for these solutions efficiently. Ant colony optimization (ACO) is one of these methods and the focus of this paper. ACO generates solutions efficiently and effectively but scales poorly to large problems. This paper merges the methods developed for reinforcement learning and hierarchical reinforcement learning (HRL) with ACO to produce an algorithm that scales to solve large, complex optimization problems. This hierarchical ant colony optimization algorithm focuses on two problem domains, the taxi world and traveling salesman problems.

The paper shows two results. The first demonstrates ant colony optimization learning algorithms placed into HRL algorithms reduces the overall learning time for those domains. Specifically, the MAXQ HRL algorithm introduced by Dietterich [3] and the taxi world problem. By incorporating the Ant-Q ACO learning method, the episode convergence is reduced significantly for large state space problems. The second is that transfering HRL methodologies to the ant colony optimization domain and the traveling salesman problem to decompose the TSP increases the speed of the solving algorithm with little to no loss in solution quality. To demonstrate this, this paper looks at the logical decomposition of spatial location and clusters the cities into separate problems to provide the decomposition.

2. BACKGROUND

Reinforcement learning (RL) is a method by which an agent learns near-optimal plans through interaction with the external environment [15]. This ability to interact with the environment allows the agent to learn plans without a model of state space interactions. Another benefit of reinforcement learning is the ability to use function approximations that reduce the time needed to solve problems and provides for solving larger problems [13]. RL is effected by dimensionality, with an exponential growth in memory and computational requirements as the problem size grows [15]. For this reason, HRL methods have been designed attempting

to alleviate these issues. HRL decomposes a problem into smaller subproblems and learns by environment responses for each concurrently with an overarching method that combines these subproblem solutions into a near-optimal solution for the entire problem [1].

2.1 Reinforcement Learning

A reinforcement learning algorithm attempts to learn an optimal value function for an unknown semi-Markofv decision process (SMDP). Following the model of an MDP, the agent knows the current state and the actions available. The algorithm chooses an action and observes the resultant state and reward. We focus on the following techniques, one used by Dietterich in his MAXQ HRL method, Q learning [18] and the ant colony optimization algorithm, Ant-Q [6], for adaptation into the MAXQ HRL decomposition. The state-action pair update equations are similar for these two learning methods. The value of a state-action pair is updated after every iteration of performing an action and observing the resultant state and reward.

2.1.1 Q-Learning

Q-Learning is a reinforcement learning method designed to handle a non-deterministic MDPs [18]. Originally defined as a dynamic programming method to provide a framework for learning algorithms, Watkins' Q-learning has become a standard reinforcement learning technique [18].

Q-learning relies on a value function, $Q(s, a)$, consisting of the current state, $s \in S$, and action, $a \in A$, to control the agent. In Q-learning the agent searches for the$Q(s, a)$ values that represent that state and action pairs expectation for future rewards. Q-learning uses an ε-greedy action selection policy to select an action to take at each state. An ε-greedy policy takes the action with the highest Q-value for the current state the majority of the time, but selects a random action with a probability of ε [15]. This stochastic nature ensures that if enough trials are completed, all possible actions will be selected and the optimal policy learned [15]. The $Q(s, a)$ value function update follows Equation 1.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)], \quad (1)$$

where α_t is the learning rate and γ is the future reward discount factor[18]. Q-learning is proven to converge to the optimal action-value function Q^* with probability of 1 [8].

2.1.2 Hierarchical Reinforcement Learning

HRL decomposes a complex reinforcement learning problem into manageable parts. Techniques include separating the problem across sets of machines designed to perform pre-determined tasks [10], splitting the problem into a set of temporal tasks, called *options* [16], and creating a hierarchy of tasks to solve the problem [3]. All methods decompose large problems into smaller problems that have individual solutions and combine to create a final solution.

The first step to solve a hierarchical problem is to identify the subtasks, primitive actions, and related hierarchy. The decomposition takes the MDP and separates it into a set of subtasks, these subtasks can be primitive actions or other subtasks. The hierarchy creates a dependency between the root task and the subtasks, where the solution of the root is based on the solution for the subtask [1]. An important aspect of the task graph is the arbitrary order of children, the order is determined by the policy at the root's level. The graph only limits the action choices at each subtask [3]. Each of these subtasks contains three components. First, it has a subtask policy π_i, which dictates the selection order of its children. Second, each subtask has a termination predicate, that identifies when the subtask policy is complete. Third, each subtask has a pseudo-reward function that assigns reward to all states encountered in the subtask [3].

The decomposition is the foundation for the MAXQ learning algorithm. If the agent follows the GLIE policy and is further constrained to break ties in the same order, the algorithm converges with probability 1 to the unique recursively optimal policy for the root task in the task graph [3]. A recursively optimal policy is a hierarchical policy such that for all subtasks, the subtask policy is optimal for that SMDP [3]. This differs from hierarchical optimal policy, which is optimal across all the policies learned within the hierarchical constraints [1]. Dietterich chose to pursue the *weaker* recursively optimal policy to allow for subtask reuse. By creating the optimal subtask policy, it allows the policy learned to be used regardless of the parameters passed in to the subtask. This reuse reduces the time needed to learn a subtask and thus reduces the overall problem time requirement [1].

2.2 Ant Colony Optimization

Mimicking the foraging behaviors of a colony of ants, the ACO algorithm models the ants activities and the strategy used by the colony. ACO follows the idea that each ant lays pheromone along the ant's path to mark the path taken, and as the pheremone builds up, the shortest path is found [4]. Using the concept of a metaheuristic, ACO can be used to provide solutions for many NP-complete problems [4]. Since the emerging behavior for following the ACO metaheuristic algorithm results in the ants finding the shortest path from a source to a destination, the most common problem ACO is tested against is the traveling salesman problem [2].

2.2.1 Ant-Q

Based on the ACO metaheuristic, Ant-Q was developed to merge Q-learning with the ACO concept [6]. Using the ACS algorithm to perform action selection and pheromone updates, Ant-Q matches the Q-learning value functions with its own Q-value function, $AQ(s, a)$, where s is the current state and a is the selected action. It returns the value of performing the action a from the state s. The concept is the same as Q-learning with the addition of multiple runs by multiple agents to solve the problem [6]. Ant-Q uses a pseudo Q-value function as the value function with a similar but varied learning function:

$$AQ(s,a) \leftarrow (1-\alpha)AQ(s,a) + \alpha[\Delta AQ(s,a) + \gamma \max_{a'} AQ(s',a')], \quad (2)$$

where AQ(s,a) is equivalent to Q(s,a), α is the learning rate, γ is the discount factor, and ΔAQ(s,a) is the delayed reward update. Ant-Q also provides a method to update the AQ-value function with delayed reinforcement. Gambardella and Dorigo discuss two methods, a global-best and an iteration-best delayed update [6]. The global-best update is calculated by

$$\Delta AQ(s,a) = \begin{cases} W/C_k, & \text{if (s,a) is performed by} \\ & \text{global best ant k} \\ 0, & \text{otherwise} \end{cases} . \quad (3)$$

where W is a constant set to 10, based on the ant system

value [6], and C_k is the cost of the k^{th} ant's solution. The iteration-best update is the same calculation as the global-best update, but is based upon the iteration's best ant, not the global best. Gambardella and Dorigo selected the iteration-best as the main source of delayed reinforcement update based on results comparing both methods [6].

Similar to the ε-greedy selection of Q-learning, Ant-Q uses a probability calculation to determine the action selection. An action is selected by policy with a probability greater than the constant q_0, otherwise a random possible action is selected [6]. This random action selection allows the algorithm to explore the state space more and is one of the foundations of ACO [2]. The stochastic calculation is based on the probability

$$P_a(s) = \frac{[AQ(s,a)]^\delta \cdot [HE(s,a)]^\beta}{\sum_{u \in A_s}[AQ(s,u)]^\delta \cdot [HE(s,u)]^\beta}, if a \in A_s \quad (4)$$

where δ and β are constants, u is all possible actions, $HE(s,a)$ is a heuristic evaluation for the state action pair, and A_s is the set of admissible actions in state s. [6].

2.2.2 Ant Colony System

For convenience we explain ACS as it ised in solving the TSP. The premise of the ACS is to track the tours found by a set of ants across a period of time and update the solutions created with an additional update to the best solution discovered so far. ACS is explained with respect to the traveling salesman problem in this article to show the comparisons between algorithms. Each ant finds a solution based on a series of next city selections. ACS uses a positive feedback method to reinforce the better solutions found by a set of ants [2]. The stochastic nature of the ant system allows a solution to escape from local minima and promote better exploration of the solution space. ACS also exploits a candidate list, a set of the cl closest cities to the starting city, where cl is constant. When the candidate list is empty, the ant selects the closest city from those remaining. If the candidate list is not empty, a probability is generated, if it is below a threshold, q_0, the city is selected based on

$$j = \arg\max_{u \in J_i^k}\{[\tau_{iu}(t)] \cdot [\eta_{iu}]^\beta\}, \quad (5)$$

where i is current city, k is ant, J_i^k is remaining cities, $\tau_{iu}(t)$ is the pheromone across edge (i,u), η_{iu} is the visibility of u from i, and β is a constant. If greater than q_0, the selection is based on the probability

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)] \cdot [\eta_{ij}]^\beta}{\sum_{l \in J_i^k}[\tau_{il}(t)] \cdot [\eta_{il}]^\beta}. \quad (6)$$

These three selection criteria promote exploitation and exploration of the solution space. The max argument selection allows the ant to follow the pheromone, the probability selection allows the ant to select another path away from the strongest pheromone, and the closest city selection is a local-greedy search [2]. After an ant selects the next city, the pheromone on the selected edge is updated, as given by

$$\tau_{ij}(t) \leftarrow (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \tau_0, \quad (7)$$

where ρ is a constant and τ_0 is

$$\tau_0 = (n \cdot L_{nn})^{-1}, \quad (8)$$

L_{nn} is the tour length found through the nearest neighbor heuristic. After all ants have found a complete tour, a comparison between each ant's solution and the current best solution found is made. If an ant has a better solution, this solution becomes the best solution. A global pheromone update is then made based on the best solution

$$\forall (i,j) \in T^+, \tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta\tau_{ij}(t), \quad (9)$$

where

$$\Delta\tau_{ij}(t) = 1/L^+, \quad (10)$$

and T^+ and L^+ are the shortest tour and length respectively. This process of ants finding solutions and updating pheromone is repeated for a set number of iterations. The last shortest path saved is the best path found through all episodes and the pheromone matrix represents the learning performed by the algorithm. However, ACS does not guarantee an optimal solution [5]. Since it is a valid documented stochastic approach to solving the TSP, ACS provides a solid algorithm to merge clustering into for result comparison [2].

2.3 Data Clustering

Data clustering is used to group data into common sets using an unsupervised method. Much like the HRL problems and TSP, clustering is a difficult problem combinatorially. Stochastic algorithms to estimate the final cluster solutions have been develop to provide near-optimal solutions with significantly less computational time. These algorithms provide the TSP task decomposition used to merge ACO domain with HRL methods. This paper looks at two common clustering algorithms which meet this goal, k-means and G-means.

2.3.1 k-Means

Data clustering by the k-means algorithm is a stochastic search and cannot guarantee the optimal clustering solution. The objective is to cluster an n-dimensional data set into k clusters based on the given attribute values [9]. This paper uses the k-means algorithm to cluster a 2-dimensional data set. The mathematical objective is to minimize the global error of all clusters [9]

$$\min E = \sum_{i=1}^{k} \sum_{x_j \in S_i} |x_j - \mu_i|^2, \quad (11)$$

where there are k clusters of data points S_i and μ_i is the mean of each cluster's data points.

The number of clusters, k, is determined before any clustering begins. The algorithm begins with a random selection of these k means. All data points are then assigned to the closest mean. After all data sets are determined, the means' locations are recalculated based on the attribute values of the data points assigned to them. The process is then repeated until a predetermined convergence rate is passed; when the total delta of the means is less than a threshold. This signals convergence of the algorithm and clustering is considered complete. With the presented algorithm, no care is taken with the initial selection and the solution may actually decrease the effectiveness of the partitioning efforts.

2.3.2 G-Means

The issues with k-means clustering stem from the assumptions made about the data set, i.e. the value of k. Ad-

ditionally, mean-based clustering algorithms, k-means included, assume the data is in a unimodal distribution, such as Gaussian [7]. Therefore, only one mean should represent the data in that distribution. Using too many means creates a complex and inefficient representation. Likewise, using too few abstracts the distribution differences and creates a too simple data representation [7]. With a constant k value, k-means cannot distinguish between the complex and abstracted representations. G-means alleviates the need to predetermine k and uses a statistical calculation to decide whether to split a simple mean into two or keep it the same [7].

The algorithm performs the clustering in the same manner as k-means with a possiblity for increasing k after each episode. A set of means is created, starting with only one mean. The G-means algorithm calls the k-means with the single mean. The data is then clustered and examined. G-means uses the Anderson-Darling statistic to determine if a mean should be split or not [7]. The statistical calculation to be examined, modified for mean estimation [7]

$$A^2_*(Z) = A^2(Z)(1 + \frac{4}{n} - \frac{25}{n^2}), \tag{12}$$

where

$$A^2(Z) = -\frac{1}{n}\sum_{i=1}^{n}(2i-1)[\log(z_i) + \log(1 - z_{n+1-i})] - n \tag{13}$$

These equations provide the test for the split of a cluster. The algorithm follows these steps to determine if a cluster is to be split. A mean is selected and a significance level α is chosen. Two centers are initialized based on the selected mean. Hamerly suggests two methods to determine the new means. We use the first method he suggests; select a small vector, m, which provides two new means at $\mu \pm m$ [7]. The algorithm then runs k-means on the data set and the two new starting means. The means calculated by k-means produce a vector between them, $v = c_1 - c_2$. The algorithm projects the data onto the new vector v by

$$\forall x \in X, x' = \langle x, v\rangle / \|v\|^2, \tag{14}$$

and transforms it to a mean of 0 and variance of 1. Finally, z_i is calculated and substituted into Equation 12. If $A^2_*(Z)$ is within the confidence level α, then reject the new means and replace with the original one. Otherwise, discard the original mean and accept the new ones. Hamerly showed this algorithm correctly identifies the number of centers needed regardless of the data distribution or density [7]. By using G-means, the clusters will not have a predetermined k value and more accurately and effectively partition the data.

3. METHODOLOGY

The first step is to develop the combined HRL and ACO algorithm using MAXQ and the ACO metaheuristic. The basic concept is to use ACO to replace the current learning methods in MAXQ and increase the searchable state space for a set of domains. Following this is the application of HRL concepts to ACS.

3.1 MAXQ with Ant-Q Learning

The identified modifications create a MAXQ-AntQ algorithm used to find a solution to a multiple task problem

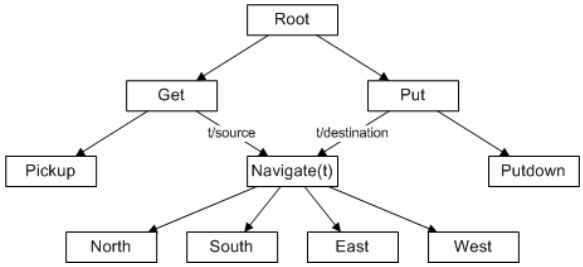

Figure 1: A sample task hierarchy graph for the Taxi World Problem.

with a predefined task hierarchy. The new algorithm demonstrates ant colony optimization techniques are useful in a hierarchical reinforcement learning domain. Through rigorous testing of several taxi world problems, the MAXQ-AntQ algorithm demonstrates its ability to converge to a solution in fewer episodes than Dietterich's MAXQ-Q algorithm. The taxi world problem uses the task hierarchy identified in Figure 1.

Using the MAXQ algorithm as the foundation to apply Ant-Q learning to, the resultant MAXQ-AntQ algorithm, has four modifications:

1. Adaptation of the Ant-Q value function for primitive nodes
2. Iteration through the set of ants (adds to non-primitive node implementation)
3. Probabilistic action selection (replaces greedy policy)
4. Adaptation of the Ant-Q value function

These changes implement a version of the ant colony optimization metaheuristic within the HRL algorithm, merging the two domains to solve larger problems faster.

The first modification replaces the value function with a modified Ant-Q value function that uses the sub-task pseudo-reward function $r_t(i)$ instead of ΔAQ, where i is the subtask index:

$$AQ_t(i,s) \leftarrow (1 - \alpha_t(i)) \cdot AQ_t(i,s) + \alpha_t(i) \cdot r_t(i). \tag{15}$$

The second modification implements a set of ants to solve the current task. By looping through M ants the algorithm creates the colony and creates the global pheromone matrix. This matrix holds the Ant-Q value function value for primitive actions as a state-action pair. In addition, a composite Ant-Q value function is created to store the higher level node values given a state-action pair. The next modification replaces MAXQ's greedy policy for action selection with Ant-Q's probabilistic selection, Eq. 6. This probability causes the ants to explore the solution space based upon the probability q_0. This change doesn't affect the overall algorithm as much as other modifications, as it is similar to the ε-greedy policy currently used by MAXQ. The final change modifies the composite value function. It replaces the current MAXQ composite function with the Ant-Q value function. Since the MAXQ value function takes an eligibility trace (reward degradation) into account, the Ant-Q value function is modified to fit this desire. The reward degradation is determined by the number of actions needed to make

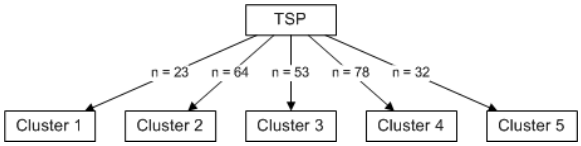

Figure 2: A sample hierarchy for a 250-city TSP with the number of cities assigned to each cluster.

the current task terminal [3].

$$AQC_{t+1}(i,s,a) \leftarrow (1-\alpha_t(i)) \cdot AQC_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [AQC_t(i,s',a^*) + AQ_t(a^*,s')] \quad (16)$$

There is a second MAXQ-Ant-Q value function, a composite value, $A\tilde{Q}C(i,s,a)$. This composite value, calculated at non-leaf nodes, is used to identify primitive tasks and provide a value for the higher level task with a state-action pair [3].

$$A\tilde{Q}C_{t+1}(i,s,a) \leftarrow (1-\alpha_t(i)) \cdot A\tilde{Q}C_t(i,s,a) + \alpha_t(i) \cdot \gamma^N \cdot [\tilde{R}_i(s') + A\tilde{Q}C_t(i,s',a^*) + AQ_t(a^*,s)] \quad (17)$$

3.2 HRL Ant Colony System

This section applies the concepts promoted in Dietterich's MAXQ to the ant colony optimization domain for solving the TSP. To exploit the benefits of HRL, the TSP has a task hierarchy imposed on it. This paper examines the use of data clustering to create this hierarchy. Two algorithms are tested, k-means and G-means detailed above. These algorithms partition the TSP into a set of subproblems, which when solved individually combine to generate the complete solution.

The combination of the ACS with a data-clustering algorithm is shown in Algorithm 1. The objective is to effectively create a hierarchy of subtasks by partitioning the overall problem into smaller TSPs. Unlike Dietterich's MAXQ, the actual hierarchy is not programmed by the developer, instead, the algorithm makes use of the data clustering techniques to determine the hierarchy dynamically. Figure 2 shows a sample hierarchy generated for a 250 city TSP. This hierarchy not only changes from problem to problem, but each episode of a problem could have a different hierarchy. The ability to dynamically create the hierarchy is one of the areas Dietterich recommends as future extension to MAXQ [3]. Fortunately, TSPs have a potential partitioning method built into the domain, other domains are not as easily dynamically partitioned. The new algorithm follows the structure of MAXQ-AntQ. It is a recursive function that traverses the task hierarchy with the subtasks being clusters from the TSP. In the same manner as MAXQ-AntQ, it clusters the TSP and each ant takes a path through the cluster and each cluster builds a path in that cluster. The ant then returns to select the next cluster to move to. Start and end cities are selected between clusters using a local greedy search that identifies the cluster means that are closest together and the closest cities in each cluster pair.

Algorithm 1 HRL ACS with Clustering Algorithm

```
{function HRL-ACS-C(TSP t)}
let T_best = ∅ be the best tour found for the TSP t
if t is a cluster TSP then
    {Generate solution for cluster TSP}
    T_best = ACS-TSP(t) {ACS TSP alogorithm }
else
    {Generate clusters and solutions for each subproblem}
    let C = the set of clusters in t {by k-means or G-means}
    let TOURS = ∅ be the set of tours for all clusters
    for each c ∈ C do
        TOURS(c) = HRL-ACS-C(TSP(c))
    end for
    {Create a TSP for the means of the clusters}
    let M = the set of means of C
    T_m = ACS-TSP(TSP(M)) {ACS-TSP algorithm [5]}
    {Combine the tours generated for all subproblems}
    randomly select first edge, (i, j) ∈ T_m
    find edge (q, r)|d(q, r) = min d(∀x ∈ C_i, ∀y ∈ C_j)
    let T_i = TOURS(i) be the tour for cluster i
    select start city s_0 ∈ C_i where edge (q, s_0) ∈ T_i
    add T_i to T_best
    add edge (q, r) to T_best
    let T_j = TOURS(j) be the tour for cluster j
    add T_j to T_best with starting city r
    for each remaining cluster C_l ∈ T_m do
        assign q ← T_best^last
        find edge (q, r)|d(q, r) = min d(1, ∀y ∈ C_l)
        add T_l to T_best with starting city r
    end for
    assign q ← T_best^last
    add edge (q, s_0) to T_best
end if
return T_best
{end HRL-ACS-C}
```

4. RESULTS

This section discusses the testing of the HRL ACO algorithms on two problem domains. There were two tests conducted, the first, on the taxi world problem, includes various problems of increasing state space size and compares mean training runtime to convergence between the MAXQ-Q and MAXQ-AntQ algorithms. The second, on TSP, includes problems increasing in city cardinality and compares tour statistics and number of iterations of ACS-TSP, HRL ACS with k-means, and HRL ACS with G-means. The implementations introduced in this paper were compared to the results of their respective foundation algorithms, MAXQ-Q for taxi world and ACS-TSP for TSP. The data is analyzed and statistical differences are highlighted between the baselines and created algorithms. An Anderson-Darling test verifies the data is a normal distribution with an α of 0.01; this value tests for more than one out of 100 values not under a normal distribution curve. A t-test using an α of 5% determines significant differences between the mean convergence for each algorithm.

4.1 MAXQ-AntQ

To assess the feasibility of merging ACO with MAXQ HRL techniques, 30 runs were performed on a set of taxi world problems. The Taxi World is a grid world problem, given a grid of size l x m with walls described as lines between grid cells that block movement between the separated cells. Four destinations are located randomly across the grid, labelled as red, blue, green, and yellow. The passenger starts

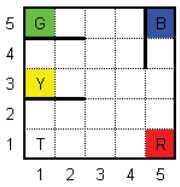

Figure 3: A sample 5x5 Taxi World Problem, T is the taxi, G, R, B, and Y are pickup and dropoff locations.

randomly at one of these four locations. The taxi can start at any grid location. The GLIE policy imposed is for the taxi to first navigate from it's start location to the passenger's location. Second, pickup the passenger. Third, navigate from the passenger's start location to the passenger's destination. Fourth, drop off the passenger. This series of tasks is considered an episode. An example 5x5 problem is shown in Figure 3. Problems sizes of 5x5, 7x7, 10x10, 20x20, 30x30, and 50x50 resulting in 500-50,000 states are tested. In a world size of 5x5, there are 500 possible states: 25 squares, 5 locations for the passenger (counting four destinations and the taxi), and 4 destinations [3]. Data collected includes the number of moves in the current solution found by the algorithm and the training run of convergence.

Since both algorithms, MAXQ-ANT and MAXQ-Q, use the same update equations, the focus is on the convergence of the algorithms to the known optimal solution. Although not guaranteed to find the optimal solution, the algorithms tend to converge to it for the smaller problems, with the convergence for the larger solution space problems further from optimal. The algorithms were allowed to run for 1000 training runs or until the update of the policy was less than 1.0%. Parameters for each learning method were kept consistent with documented parameters [3] [6], $\lambda = 0.9$, $\alpha = 0.1$, $\beta = 2$, $\delta = 1$, and $\rho = 0.1$. An episode is considered complete when a solution is returned which meets the four goals dictated by the taxi world problem.

The MAXQ-Q algorithm provides a documented baseline for the taxi world problem domain. Examining the data in Table 1, the modified algorithm MAXQ-AntQ converges in significantly fewer episodes than Dietterich's original algorithm. The improvement is an average of 12.1% fewer episodes to convergence by using the HRL-ACO-MAXQ algorithm over MAXQ-Q. The best improvement is noted in the 7x7 sized problem, a 17.8% less episodes mean convergence rate. This improvement stems from the ability of MAXQ-AntQ to use the set of ants at each level in the hierarchy rather than the single agent used in MAXQ-Q [3]. Although neither algorithm converges to the optimal solution every time, both converge to near-optimal solutions with no significant difference in solution quality. These results show Ant-Q can be adapted for use in a hierarchical reinforcement learning domain. By integrating Ant-Q with MAXQ-Q, not only does the learning rate increase, the combination of these two concepts lends itself to other domain combination possibilities.

4.2 HRL ACS

The second set of tests incorporate the hierarchical decomposition of the TSP with the ACS-TSP algorithm. The

Table 1: Convergence data for MAXQ and MAXQ-AntQ on selected Taxi World Problems. Numbers represent the episode number at convergence.

Problem	MAXQ		MAXQ-AntQ	
	Best	Mean	Best	Mean
5x5	94	112±15	83	**97±17**
7x7	109	129±21	91	**106±19**
10x10	215	232±29	184	**214±32**
20x20	487	496±14	378	**421±41**
30x30	744	781±48	687	**724±49**
50x50	944	984±21	811	**876±46**

tests include problems ranging in size from 48 cities to 50,000 cities. These problems were tested with three algorithms, ACS-TSP, HRL ACS with hierarchy generated by k-means clustering, and HRL ACS with hierarchy generated by G-means clustering. The algorithms were allowed to run a set number of ants for a constant number of time steps. Data collected includes the tour statistics and run time. The results identify a tradeoff between solution time and size versus solution quality. The TSPLIB [12] problems were selected to show the trends of each algorithm as the problem size grows. In addition, several randomly created problems are used to show the differences between the three algorithms and highlight the effects of clustering on larger and non-geographically modeled problems. Since these algorithms were created for the sole purpose of this experiment, no known optimal value is available to compare against, however each algorithm's solutions can be contrasted. As before, these experiments used the parameters documented by Dorigo in all three algorithms [5]. In addition, the larger problems show the ability for the modified algorithms, HRL ACS with k-means and G-means, to handle these problems, whereas TSP-ACS cannot in a feasible amount of time.

Table 2 shows the tour and timing statistics for the three algorithms. In the table there are several problems lwith results abeled as "—"; these problems have run times that exceed 7 days (86,400 seconds). Unfortunately, the clustering did not show a continuum of the quality of solutions from ACS-TSP to the other methods. However, the data showed k-means (k is set to the number of cities divided by 25) produces solutions an average of 5.0% further from optimal than ACS-TSP. Whereas the G-means algorithm produced solutions approximately 9.5% further from optimal.

Figure 4 shows the mean episode run time for each algorithm, measured in seconds. Comparing the data, once a problem could be clustered, there was a significant decrease in run time from the basic ACS-TSP to the other algorithms. HRL ACS with k-means produced a solution on average 20.88 times faster than ACS-TSP. HRL ACS with G-means had similar improvements, with an average speedup of 20.94 times faster. Taking all problems into account, G-means results in a decrease of 9.5% in run time compared with k-means. This benefit is important given the set of problems ACS-TSP was unable to solve in a reasonable time. With problems larger than 5,000 cities, a test run is unobtainable as the estimated time to complete an episode of 1000 time steps is over 7 days. However, with both k-means and G-means clustering, the algorithms were able to produce a solution to the problem in just over 1 hour per episode. The time decrease is the main benefit of the

Table 2: Comparison of Mean % from Known Best Tour Length and Run Time for ACS-TSP, HRL ACS with k-Means Clustering, and HRL ACS with G-Means Clustering. A "—" in results indicates run time was greater then 7 days and "N/A" identifies no known best solution.

Name	ACS-TSP		HRL ACS k-Means		HRL ACS G-Means	
	% Best	Run Time(s)	% Best	Run Time(s)	% Best	Run Time(s)
att48	6.99%‡	99.31	7.03%	98.11	19.35%	16.60‡
eil51	9.56%‡	114.95	8.03%	24.24†	14.04%	12.99‡
eil101	13.75%† ‡	993.23	15.47%	86.23†	22.75%	22.54‡
a280	22.59%† ‡	14377.42	27.68%	657.15†	27.11%	174.26‡
att532	23.23%† ‡	66638.92	27.16%	4476.29†	26.32%	1960.91‡
pr1002	35.92%	70841.01	24.93%†	3736.97†	22.77%‡	1005.07‡
u2152	28.34%‡	615482.70	27.83%	4752.79†	30.66%	2179.71‡
rl5915	—		37.58%†	16980.36†	42.20%‡	9463.71‡
rl11849	—		35.02%†	43874.86†	51.33%‡	19630.67‡
usa13509	—		37.01%†	7754.47†	44.27%‡	30344.16‡
d18512	—		31.87%†	15156.63†	38.14%‡	40437.22‡
rand75	N/A	321.05	N/A	73.13†	N/A	21.22‡
rand200	N/A	5514.16	N/A	232.93†	N/A	48.28‡
rand500	N/A	30704.61	N/A	676.09†	N/A	120.15‡
rand1000	N/A	129595.10	N/A	2386.40†	N/A	345.85‡
rand2000	N/A	432708.33	N/A	3535.09†	N/A	536.14‡
rand5000	N/A	—	N/A	13476.09†	N/A	2039.26‡
rand10000	N/A	—	N/A	28929.07†	N/A	102654.70‡
rand25000	N/A	—	N/A	128269.40†	N/A	937460.60‡
rand50000	N/A	—	N/A	275468.46†	N/A	1130765.40‡

† identifies significantly better results between ACS-TSP and HRL ACS with k-means.
‡ identifies significantly better results between ACS-TSP and HRL ACS with G-means.

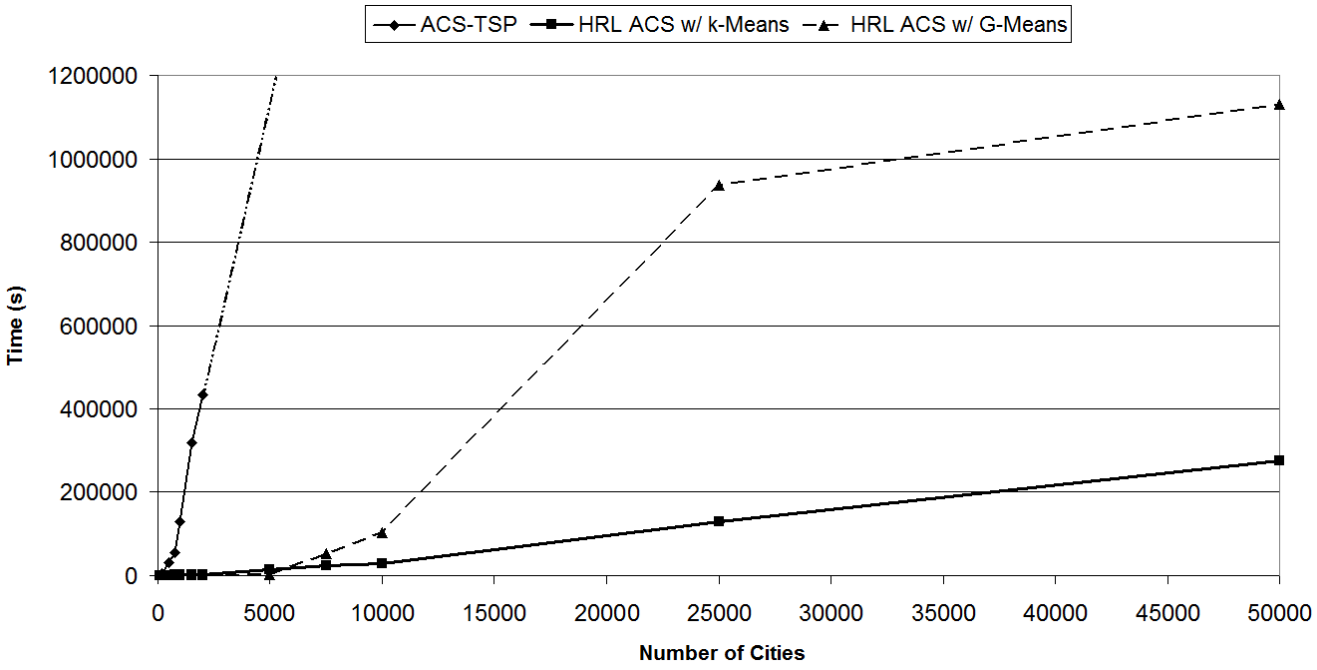

Figure 4: A graph of the three TSP algorithms' episode run time.

modified ACS-TSP algorithms. By providing a solution to large complex TSPs in a faster manner, more problems can be efficiently solved.

5. CONCLUSIONS

The results of testing the algorithms, MAXQ-AntQ and HRL ACS with clustering, against the baseline algorithms, MAXQ-Q and ACS-TSP, demonstrate that the combination of the two concepts, HRL and ACO, is not only feasible but beneficial. The first algorithm, MAXQ-AntQ, followes a similar convergence pattern as the MAXQ-Q algorithm with an added benefit of increasing the learning rate by decreasing the number of training runs to convergence by an average of 12.1%. There is no significant difference in quality of solution, demonstrating Ant-Q learning inserted into MAXQ provides a benefit to the algorithm.

The second algorithm is a modified ACS clustering algorithms. Although both k-means and G-means on average provided solutions further from optimal than ACS-TSP, the difference was only 5.0% for k-means and 9.5% for G-means. This decrease in solution quality was acceptable as the run time speed up for both algorithms was greater than 20 times. This is especially significant with large scale problems as the modified algorithms obtain solutions while ACS could not.

One area not inspected by this research is the similarities between multilevel techniques and HRL. Multilevel techniques attempt to make problem instances simpler by coarsening the problem and refining the solutions of each coarsened problem in reverse order [17]. The algorithm constructs a series of smaller and coarser versions of the original problem, with the hope each coarser problem retains the important features of its parent problem [17]. The hypothesis of this technique is that the coarsening gradually smooths the objective function, a local search function should work well as an optimization metaheuristic [17]. Much like HRL, multilevel techniques require a hierarchy to be created within the original problem domain. Unlike HRL where the problem domain is split into low-level tasks, multilevel attempts to simplify the entire problem [17]. These techniques, HRL and multilevel, are similar and could be combined in an effort to provide another method to generate solutions for combinatorial optimization problems.

There are also several other areas to continue research into combining HRL and ACO. The first is the selection of the HRL algorithm. A selection of another HRL algorithm could provide similar, if not better, results and demonstrate the same properties. In addition, there are three areas to examine in automatically determining the hierarchy using clustering: the clustering technique, the cluster combination, and the base TSP algorithm. These areas can all be explored for more options and may provide a better algorithm design through testing. The clustering technique research could look at other cluster methodologies, X-means [11], or even additional modifications to the two selected. The final area is the ACS-TSP algorithm. There are several documented improvements to this baseline algorithm including 2/3-opt [5] and Max-Min Ant System [14]. Those adaptations could only help with the solutions found using this paper's modified algorithms. By increasing the efficiency of the foundation algorithms used, it will benefit all parts of the TSP hierarchy.

6. REFERENCES

[1] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003.

[2] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Inc., 198 Madison Avenue, New York, New York 10016, 1999.

[3] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence*, (13):227–303, Nov. 2000.

[4] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *Computational Intellligence Magazine, IEEE*, 1(4):28–39, 2006.

[5] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.

[6] L. M. Gambardella and M. Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem. In *International Conference on Machine Learning*, pages 252–260, 1995.

[7] G. Hamerly and C. Elkan. Learning the k in k-means. In *Proceedings of the Seventeenth Annual Conference on Neural Information Processing Systems (NIPS)*, pages 281–288, 2003.

[8] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. Technical report, Cambridge, MA, USA, 1993.

[9] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Symposium on Mathematical Statistics and Probabilities*, pages 281–297, 1967.

[10] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines, Jan. 17 1997.

[11] D. Pelleg and A. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. *ICML 2000*, 2000.

[12] G. Reinelt. Tsplib 95, 2007.

[13] W. D. Smart. Explicit manifold representations for value-functions in reinforcement learning. In *Proceedings of the Eighth International Symposium on Artificial Intelligence and Mathematics*, January 2004. Paper number AI&M 25-2004.

[14] T. Stutzle and H. Hoos. The max-min ant system and local search for combinatorial optimization problems, 1999.

[15] R. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, 1998.

[16] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999.

[17] C. Walshaw. A Multilevel Approach to the Travelling Salesman Problem. Tech. Rep. 00/IM/63, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, August 2000.

[18] C. Watkins. *Learning From Delayed Rewards*. PhD thesis, King's College, Oxford, May 1989.