9-2007

Scaling Ant Colony Optimization with Hierarchical Reinforcement Learning Partitioning

Erik J. Dries

Scaling Ant Colony Optimization with Hierarchical Reinforcement Learning Partitioning

THESIS

Erik Dries, Captain, USAF

AFIT/GCS/ENG/07-16

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis 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 United States Government.

AFIT/GCS/ENG/07-16

Scaling Ant Colony Optimization with Hierarchical Reinforcement Learning Partitioning

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Erik Dries, B.S.C.S.

Captain, USAF

September 2007

AFIT/GCS/ENG/07-16

Scaling Ant Colony Optimization with Hierarchical Reinforcement Learning Partitioning

Erik Dries, B.S.C.S.
Captain, USAF

Approved:

/signed/	31 Aug 2007
Gilbert L. Peterson, PhD (Chairman)	date
/signed/	31 Aug 2007
Gary B. Lamont, PhD (Member)	date
/signed/	31 Aug 2007
Maj Christopher B. Mayer, PhD (Member)	date

AFIT/GCS/ENG/07-16

Abstract

This research merges the hierarchical reinforcement learning (HRL) domain and the ant colony optimization (ACO) domain. The merger produces a HRL ACO algorithm capable of generating solutions for both domains. This research also provides two specific implementations of the new algorithm: the first a modification to Dietterich's MAXQ-Q HRL algorithm, the second a hierarchical ACO 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 and SARSA, 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 research 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 research tests two clustering algorithms, k-means and G-means. The results demonstrate the algorithm with data clustering produces solutions 20 times faster but with 5-10% decrease in solution quality.

Acknowledgements

I would like to thank my thesis advisor, Dr. Gilbert Peterson, for his invaluable knowledge, guidance, and pressure in this endeavor. I would also like to thank my wife for her unending support even when I left her on the other side of the country.

Erik J. Dries

Table of Contents

Page

List of Figures

List of Tables

Scaling Ant Colony Optimization with Hierarchical Reinforcement Learning Partitioning

I. Introduction

Problems throughout the real world require optimized solutions. These problems include navigation, mapping, and planning. Developed 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 research. ACO generates solutions efficiently and effectively but does not scale well with large problems. This research merges the methods developed for reinforcement learning and hierarchical reinforcement learning 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 problem and traveling salesman problem.

This chapter provides an overview of the research conducted in this investigation. It covers the problem to be solved, an overview of hierarchical reinforcement learning and the goals and objectives of this research investigation. Chapter I also highlights the assumptions and risks of this research and provides an overview of the thesis document.

1.1 Problem Statement

Reinforcement learning has been shown to be useful for autonomous agents in their control and planning [20]. Unfortunately the documented domains tend to be simple, single-goal problems and are not directly reflective of the real world. Hierarchical reinforcement learning methods attempt to allow for more of the real world to be modeled by two techniques. The first being reuse of common procedures or solutions throughout the problem. Second, the ability to decompose the problem into

smaller, less computationally taxing tasks. These methods allow HRL domains to model more realistic problems and provide quality solutions efficiently for them.

These advantages should be able to be transferred to other learning domains, including ant colony optimization. By merging the ACO domain with HRL methods, the research demonstrated there is the benefit of a speed increase for solving the problems with little to no loss in solution quality. This research completes the discussed combination and provides data to show it is both feasible and beneficial.

The research direction is two-fold. The first goal is to show 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 [4] and the documented taxi world problem. By incorporating the Ant-Q ACO learning method, the overall run time is reduced significantly for larger state space problems. The second goal is to transfer HRL methodologies to the ant colony optimization domain and the traveling salesman problem to demonstrate a decomposition of the TSP increases the speed of the solving algorithm with little to no loss in solution quality. To demonstrate this, this thesis looks at the logical decomposition of spatial location and clusters the cities into separate problems to provide the hierarchy decomposition.

1.2 Research Goal

The overall and guiding goals of this research are: 1) to demonstrate ACO learning algorithms placed into HRL algorithms reduces the overall learning time for those domains; and 2) to transfer HRL methodologies to the ACO domain and the TSP to demonstrate a decomposition of the TSP increases the speed of the algorithm with little to no significant loss in solution quality.

1.3 Assumptions

The techniques and algorithms presented in this thesis avoid a requirement that developers use a specific language for the implementation. However, they are presented in a manner which draws upon current object-oriented programming fundamentals. This assertion allows for the use of both statically-checked and dynamically-checked languages.

The two problem domains presented, taxi world and the traveling salesman problem are assumed to be constructed and designed in accordance with the problem requirements outlined in Chapters III and IV. This is to alleviate any issues with more complex problems and the ability for these algorithms to handle them effectively.

1.4 Overview

The structure of this thesis is as follows: Chapter I introduces the problem and research goals. Chapter II provides a thorough overview of reinforcement learning, three hierarchical reinforcement learning techniques, focusing on one as the selected domain to test application against. Chapter II continues with the introduction and discussion of ant colony optimization and data clustering, again focusing on those specific algorithms selected to research. Chapter III presents the detailed design of a hierarchical reinforcement learning method and the adaptation of an ant colony optimization algorithm to the taxi world problem. Chapter IV then extends the concepts from the previous chapter to an ant colony optimization domain and presents a detailed design of an algorithm using the HRL concepts with ACO as its base. An implementation for each domain is used to highlight the benefits or lack thereof of these modified algorithms with results discussed in Chapter V. The final chapter, Chapter VI, presents concluding remarks and recommendations for future research into the use of HRL techniques in the ACO domain and the TSP.

II. Backgound

This chapter introduces the foundations used in this thesis. Reinforcement learning was developed as a method to have an agent learn near-optimal plans through interaction with the external environment [20]. This ability to interact with the environment allows the agent to learn plans without needing a model of state space interactions. This state abstraction creates an online learning agent and reduces the computational requirements for planning. Other benefits of reinforcement learning are the ability to use function approximations reducing the time needed to solve problems and increasing the size of solvable problems [18]. However, they are effected by dimensionality; the exponential growth in memory and computational requirements as the problem size grows [20]. For this reason, several methods of hierarchical reinforcement learning (HRL) have been designed attempting to alleviate the issue.

HRL functions by decomposing the problem into smaller subproblems and are learned concurrently with an overarching method to combine these subproblem solutions to give a near-optimal solution for the complete initial problem [2]. HRL approaches are reviewed and one is selected to apply the decomposition in the HRL learning method to ant colony optimization (ACO) learning. The majority of HRL methods rely on the theory of semi-Markov decision processes to allow the decomposition and develop an algorithm which converges on a near-optimal solution [2]. This chapter presents the basis for reinforcement learning, Markov decision processes (MDP) [20], current hierarchical reinforcement learning methods, and techniques used by MAXQ HRL. It further identifies the MAXQ framework developed by Dietterich [4] to adapt for ant colony optimization. Finally, it explores the ACO domain and presents the algorithms used in merging the two areas to create a version of ACO which scales to larger problem spaces.

2.1 Markov and Semi-Markov Decision Processes

The majority of reinforcement learning methods are based on Markov decision processes. MDPs provide a simple, discrete-time state and action framework to study

algorithms and the derived properties [2]. A MDP models problems as a sequence of stages represented by the state of the system s, in which an event of action a occurs, resulting in a state *s'*. The agent receives a real-valued reward r, evaluated from a function based on the state-action pair, *(s,a)*. A policy determines the action the agent performs for a given state. The policy is what reinforcement learning algorithms search for, hopefully converging towards an optimal policy π^*.

For any given policy π and state s, an expected infinite-horizon discounted return can be calculated as shown in Equation 2.1. This constitutes the value function for policy π at state s.

$$V^{\pi}(s) = E\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s, \pi\} \tag{2.1}$$

The objective again is to find an optimal policy π^* that maximizes the value function for every state in the problem. Another method to determine an optimal policy is to utilize a state-action function. This separates the value function in Equation 2.1 into a state-action pair function, see Equation 2.2. It denotes the expected infinite-horizon discounted return for executing action a in state s, selected by policy π.

$$Q^{\pi}(s,a) = E\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s, a_t = a, \pi\} \tag{2.2}$$

The optimal policy selects the maximum state-action value as the selected action in a given state. The value functions referring to an optimal policy are denoted by V^* and Q^*. Using the Bellman equation, reinforcement learning methods use these equations to produce the optimal value functions, V^* see Equation 2.3, Q^* see Equation 2.4.

$$V^*(s) = \max_{a \in A_s} [R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')] \tag{2.3}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a \in A'_s} Q^*(s',a') \tag{2.4}$$

These lend to the ability for dynamic programming and reinforcement learning methods to abstract the current and resultant state space from the overall problem and remove the need to know about the rest of the state space. All recent reinforcement learning methods generalize the MDPs into semi-Markov decision process SMDP [2]. A SMDP creates a discrete time step to make a decision, removing the continous nature of MDPs and allowing an agent to appear to wait for a given amount of time and instantaneously transition to the next state [11]. By including this positive time step into the above optimal value function equations, the Bellman equations for V^* and Q^* become [2]

$$V^*(s) = \max_{a \in A_s} [R(s,a) + \gamma \sum_{s',\tau} P(s', \tau | s, a) V^*(s')] \tag{2.5}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s',\tau} P(s', \tau | s, a) \max_{a \in A'_s} Q^*(s', a'). \tag{2.6}$$

2.2 Reinforcement Learning

Using the functions presented above, a reinforcement algorithm attempts to learn an optimal value function for an unknown 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 three learning methods, two used by Dietterich in his MAXQ HRL method, Q learning [22] and SARSA(0) [17] and the ant colony optimization algorithm, Ant-Q [7], for adaptation into the MAXQ HRL decomposition. Adapting the state-action pair functions from above, the value function equations are similar for all three learning methods, the value is updated after every iteration of performing an action and observing the resultant state and reward.

2.2.1 Q-Learning. Q-Learning was introduced by Watkins [22] as a reinforcement learning method designed to handle a non-deterministic MDP. Originally

defined as a dynamic programming method to provide a framework for learning algorithms, Watkins Q-learning became a standard reinforcement learning technique [22].

Q-learning relies on a value function using the current state and possible action to control the agent. Shown in Algorithm 1, in Q-learning the agent searches for the best possible value function and uses an ε-greedy action selection policy to decide what action to take at each state. An ε-greedy selection algorithm takes the highest Q-value function result the majority of the time, but makes a random action with a probability of ε [20]. This stochastic nature ensures if enough trials are completed, all possible actions will be selected and the optimal policy learned [20]. Updating

Algorithm 1 Q-Learning Algorithm [20]

1: Initialize $Q(s, a)$ arbitrarily
2: **repeat**
3: Choose action, a from state, s using policy derived from $Q(s, a)$, (ε-greedy)
4: Take action, a, observe new state, s', and reward, r
5: Update $Q(s, a)$ value
6: Accept new state, $s \leftarrow s'$
7: **until** s is terminal

$Q(s, a)$ value function follows Equation 2.7.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)], \tag{2.7}$$

where α_t is the learning rate [22]. Q-learning has been proven to converge to the optimal action-value function Q^* with probability of 1 [10] if the following holds true

$$\lim_{T\to\infty} \sum_{t=1}^{T} \alpha_t = \infty \text{ and } \lim_{T\to\infty} \sum_{t=1}^{T} \alpha_t^2 < \infty. \tag{2.8}$$

2.2.2 State action reward state action (SARSA) Learning. SARSA, was introduced by Rummery [17] as a reinforcement learning method able to handle any MDP. It is similar to Q-learning with additional arguments to perform the $Q(s, a)$ value functions updates. The algorithm also includes an additional action selection step, see Algorithm 2. The name comes from the quintuple used to perform the

update, s current state, a action, r reward observed, s' next state, a' next action selected. The algorithm change involves a second action selection using the next state and its possible actions. The SARSA value function update is:

Algorithm 2 SARSA Learning Algorithm [17]

1: Initialize $Q(s, a)$ arbitrarily
2: **repeat**
3: Choose action, a from state, s using policy derived from $Q(s, a)$, (ε-greedy)
4: Take action, a, observe new state, s', and reward, r
5: Choose action, a' from state, s' using policy derived from $Q(s', a')$, (ε-greedy)
6: Update $Q(s, a)$ value
7: Accept new state, $s \leftarrow s'$ and action selected, $a \leftarrow a'$
8: **until** s is terminal

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)], \tag{2.9}$$

where α_t is the learning rate [17]. SARSA's learning function will also converge to the optimal action-value function if it follows equation 2.8 and uses a GLIE policy to choose actions. A GLIE (Greedy in the Limit with Infinite Exploration) policy is any policy satisfying: 1) each action is executed infinitely often in every state that is visited infinitely often; and 2) in the limit, the policy is greedy with respect to the Q-value function with probability 1.

2.3 Hierarchical Reinforcement Learning

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 [14], splitting the problem into a set of temporal tasks, called *options* [21], and creating a hierarchy of tasks to solve the problem [4]. All methods decompose the large problem into smaller, more manageable parts that then have individual solutions that combine to create a final solution. These methods were created to battle the issue of large state space by reducing the processed state space each part requires.

2.3.1 Hierarchical Abstract Machines. A HAM is a program which places a set of constraints on the policy for a learning agent. The constraints limit the actions possible given a state and the HAM [14]. These machines can be created in a hierarchy, a tree structure, to provide a set of constrained policies and constrained actions possible [14]. The hierarchy allows the highest level to learn only when to use a lower level action, versus attempting to learn the entire state. This abstraction is useful for complex behaviors in robots and complex problems, including multiple goal systems.

The machines in a HAM are defined by a set of states, a transition function, and an initialize function [14]. There are four possible state types within a machine [14]:

- Action - execute an action in the environment
- Call - call another HAM as a subroutine
- Choice - a nondeterministic state transition
- Stop - halts execution and returns to previous HAM.

The transition function stochastically moves the machine state from one to another after an action, choice, or call is completed. The transition function takes the current state and environment into account while transitioning to another state. This allows the learning to affect the environment and the environment to affect the learning [14]. Parr does explain the environment can be partially observable and the machines can still function, but assumes the environment is completely observable in the tests [14].

A HAM is started with a single machine which controls the execution of all other machines and monitors the completion of all machine actions. Parr uses a grid world to explain the setup of the navigation HAM. The grid world is made up of a series of hallways and intersections with randomly placed obstacles in the hallways. Parr's navigation HAM has three layers to it. The top layer is a machine used to choose the direction of movement down a hallway in one of four cardinal compass points [14]. This machine is used as the initial start machine in the HAM. The second level is

a set of four machines which actually move the agent in the specified direction [14]. These machines introduce the constraints of individual machines within the HAM. Each machine only selects an action in line with the direction they are specified to move the agent in [14]. Within these directional machines, a choice is made to handle obstacles in one of four ways, two back away from the obstacle and the others follow walls in an attempt to circumvent the obstacle [14]. The final layer in the HAM activates the primitive controls on the agent to perform the selected actions. The transition function within Parr's navigation HAM uses partial state descriptions (from sensors) to move the state from machine to machine and produce an output action for the agent to undertake.

Since HAMs operate and learn within a reduced state space, the learning rate is faster and exploration occurs more with HAM than a simple reinforcement learning algorithm [14]. Parr developed a HAMQ, which takes the concepts from Q-learning and applies the HAM concepts. The addition of machine state takes the Q value function from Equation 2.7 to

$$Q([s_c, m_c], a) \leftarrow Q([s_c, m_c], a) + \alpha[r_c + \beta_c V([t, n]) - Q([s_c, m_c], a), \tag{2.10}$$

which extends the state action pair to the state, machine state action triplet [14]. Since it is based off of Q-learning, HAMQ also converges to the optimal policy for any state with probability of 1 [14] [22].

The hierarchical learning done HAMs is based upon a set of machines with constraints on possible actions selected from each machine. The ability for one machine to call another and vice versa creates a linkage between all machines and can be implemented to learn the optimal policy given the set of machines can cover the set of all possible actions [14]. Parr demonstrated HAMs can be used in reinforcement learning domains with a decrease in convergence time [14].

2.3.2 Option Hierarchy. The second method of HRL is Sutton's *option* method. Sutton attempts to abstract complex problems using temporal abstractions [21]. This abstraction follows how people think of an action or goal. For example, if someone wants to call a taxi, they must lookup the number, pick up the phone, dial the number, and talk to the taxi dispatcher. This series of actions are abstracted by the person into one goal, call a taxi. However, the abstraction does not stop there; each of those goals can be broken into the individual actions require to satisfy those goals [21]. Sutton attempts to use temporal abstraction as the basis to create a hierarchy of actions and have an agent learn and perform complex actions and multiple goals [21].

Since reinforcement learning techniques use the Markov assumptions to limit state knowledge to previous state and current state, many cannot take advantage of temporal abstraction [21]. Therefore, Sutton introduces SMDPs and uses the extended reinforcement learning techniques to allow for these discrete-event continuous-time models which can take a variable amount of time into consideration [21]. Sutton merges the concepts between MDP and SMDPs by splitting the unknown variable time steps for SMDPs into set length time steps. He can accurately perform reinforcement learning using temporal abstractions with his *options* methods [21].

Options consist of three parts: a policy to select actions, a termination condition, and an initiation set. The initiation set is the set of requirements before the policy is allowed to be invoked [21]. If the option is to take an action with an MDP, the algorithm handles it just like Q-learning. With the assumption that the option will complete the task or at least know when it can be terminated based on the termination function [21]. If the option requires using a SMDP, there is an additional timeout function used to ensure the option does not continue indefinitely [21]. The actions selected are not merely based upon s_t, the options take the state history into account; allowing the invocation of actions based upon a series of time steps (artificial or not) [21].

There can be two types of options: primitive options which are single time step actions; and multi-step options which take a variable amount of time to perform multiple actions. The multi-step options rely on the primitive options to perform the actual actions as they are similar to the state action pairs used by reinforcement learning [21]. Since the options are defined similarly to actions and can be terminated in a set way, a program can create a sequence of options and provide a higher option to call the sequence of lower options, creating a temporal abstracted hierarchy. Using a simple grid world with a set of rooms and adjoining hallways, Sutton demonstrates how an option policy can be used to perform complex actions and achieve multiple goals. For instance if an agent needs to get from one room to another and place something down in a corner, a set of options can be created. The high level option is the sequence of options, move to second room, move to corner, and place object down. These three options in turn create their own sequence of options, multi-step or primitive. The hierarchy is built until all options can be terminated by a set of conditions. An additional bonus is the ability for options to share termination conditions between sets [21].

The hierarchy allows the options to learn a policy based on abstraction. Similar to Parr's HAMs, the options method creates levels of abstraction to alleviate issues with complex state spaces. Options can additionally handle SMDPs with temporal abstraction since the policies take a small history of states into account and artificially produce a MDP based on constant time steps [21]. Sutton showed the options method can effectively and efficiently provide solutions for complex problems and multiple goal problems. The learning and planning rate decreased over basic reinforcement learning techniques by the addition of the option hierarchy [21].

2.3.3 MAXQ. All basic algorithms for learning or planning are considered "flat" - they treat the search space as one flat space; they do not take into account steps or hierarchies in the problem. MAXQ was developed by Dietterich as a method to allow for hierarchical MDPs to be learned by a system [4]. HRL methods take the

search space and decompose it into a hierarchy of subtasks in one of three documented methods. The first method, employed by Sutton takes an *option* method, where the policy is fixed and implemented by the programmer [21]. The second method defines the subtasks in terms of a non-deterministic finite-state controller, used in Parr and Russell's hierarchy of abstract machines [14]. The final method, used in MAXQ, defines the subtasks in terms of a termination predicate and a local reward function [4]. Where the first two methods rely on reducing the problem into a single SMDP, MAXQ reduces the problem to a set of SMDPs, who's policies can be learned simultaneously. All three methods do share a common drawback, the learned policy may be suboptimal. Nonetheless, the learned policy will be the best possible policy based on the constraints introduced by the programmer [4].

The task of the taxi problem, described in detail in Section 3.1, illustrates the need to support temporal abstraction, state abstraction, and subtask sharing. Temporal abstraction is needed as the different subtasks take differing amounts of time and this is inconsequential for the policy learning [4]. State abstraction is needed to prune irrelevant state components from the overall problem solution. Finally, subtask sharing allows the learning algorithm to learn the subtask policy once and reuse the subtask as needed again in the hierarchy [4].

The first step to solve a hierarchical problem is to identify the subtasks, primitive actions, and related hierarchy, taking each need from above into account; Figure 2.1 shows an example of a task graph for the taxi problem. The decomposition takes the core MDP and separates into a set of subtasks, these subtasks could 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 [2]. An important aspect of the task graph is the arbitrary order of children, the order is determined by the policy at the parent's level. The graph only limits the action choices at each subtask [4]. 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, which identifies when

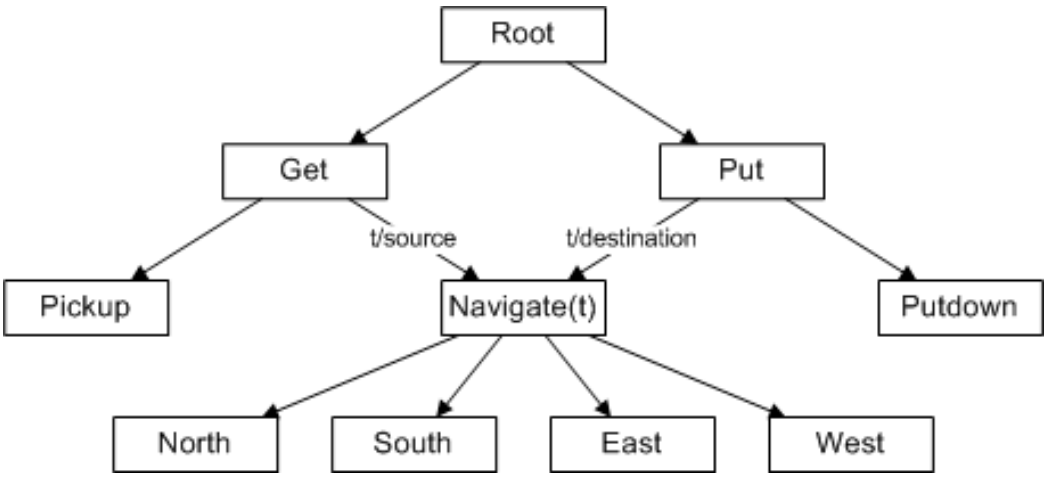


Figure 2.1: A task hierarchy graph for the Taxi World Problem.

the subtask policy is complete and can stop execution. Third, each subtask a pseudo-reward function that assigns reward values to all states encountered in the subtask [4]. MAXQ adds one additional component to allow for a series of SMDPs to be solved, a stack containing the subtasks hierarchy called is passed into each subtask called. At any given time, the top of the stack stores the current subtask. This allows the individual subtasks to not be concerned with the state outside of its SMDP.

Given the hierarchical decomposition into n subtasks, as described by the task graph, a hierarchical policy is defined as $\pi = \pi_0,...,\pi_n$, where π_i is the policy of subtask i. The hierarchical value function used by MAXQ is defined as

$$V^{\pi}(i,s) = V^{\pi}(\pi_i(s),s) + \sum_{s',\tau} P_i^{\pi}(s',\tau|s,\pi_i(s))\gamma^{\tau}V^{\pi}(i,s'), \tag{2.11}$$

where $V^{\pi}(i,s)$ is the expected return for completing subtask i. Similarly, the state-action value function, can be defined as

$$Q^{\pi}(i,s,a) = V^{\pi}(a,s) + \sum_{s',\tau} P_i^{\pi}(s',\tau|s,\pi_i(s))\gamma^{\tau}Q^{\pi}(i,s',\pi(s')). \tag{2.12}$$

Dietterich identifies the summation in the state-action value function as the completion function, C^π

$$C^\pi(i,s,a) = \sum_{s',\tau} P_i^\pi(s',\tau|s,\pi_i(s))\gamma^\tau Q^\pi(i,s',\pi(s')), \tag{2.13}$$

which gives the expected return for completing subtask i after subtask a is performed. Rewriting Equation 2.12 using the completion function yields

$$Q^\pi(i,s,a) = V^\pi(a,s) + C^\pi(i,s,a). \tag{2.14}$$

These equations allow the hierarchical value function to be represented, assuming n subtasks, in set order a_i, i_n^0, starting at the root.

$$\begin{aligned} V^\pi(0,s) &= V^\pi(a_n,s) + C^\pi(a_{n-1},s,a_n) && (2.15)\\ &\quad + \cdots + C^\pi(a_1,s,a_2) + C^\pi(0,s,a_1), && (2.16) \end{aligned}$$

where

$$V^\pi(a_n,s) = \sum_{s'} P(s'|s,a_n)R(s'|s,a_n). \tag{2.17}$$

The decomposition is the foundation for the MAXQ learning algorithm, which is able to learn hierarchical policies. The algorithm is recursively applied to a form of a SMDP Q-learning to estimate the completion function for subtasks. If the agent follows the GLIE policy, Section 2.2.2, and is further constrained to break ties in the same order, the algorithm converges with probability 1 to the unique recursively optimal policy fir the root task in the task graph [4]. A recursively optimal policy is a hierarchical policy such that for all subtasks, the subtask policy is optimal for that SMDP [4]. This differs from hierarchical optimal policy, which is optimal across all the policies learned within the hierarchical constraints [2]. 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 [2].

Dietterich provides two versions of the MAXQ algorithm. First, MAXQ-0 which assumes only at the completion of the subtask will the agent receive a reward. Second, MAXQ-Q which allows the addition of a pseudo-reward. For instance, as described in the taxi problem, each movement has a reward of -1, a bad pickup/putdown is -20. These rewards are taken into consideration for the learning of the recursively optimal policy, adding additional computations to the value function.

Algorithm 3 MAXQ-Q Algorithm [4]

{function MAXQ-Q(MaxNode i, State s)}
let $seq = ()$, be the sequence of states visited while executing i
if i is a primitive MaxNode **then**
  execute i, receive r, and observe result state s'
  $V_{t+1}(i,s) \leftarrow (1-\alpha_t(i)) \cdot V_t(i,s) + \alpha_t(i) \cdot r_t$
  push s onto the beginning of seq
else
  let $count = 0$
  while $T_i(s)$ is false **do**
    choose an action a according to the current exploration policy $\pi_x(i,s)$
    let $childSeq$ = MAXQ-Q(a,s), where $childSeq$ is the sequence of states visited while executing action a. (in reverse order)
    observe result state s'
    let $a^* = \arg\max_{a'}[\tilde{C}_t(i,s',a') + V_t(a',s')]$
    let $N = 1$
    for each s in $childSeq$ **do**
      $\tilde{C}_{t+1}(i,s,a) \leftarrow (1-\alpha_t(i)) \cdot \tilde{C}_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [\tilde{R}_i(s') + \tilde{C}_t(i,s',a^*) + V_t(a^*,s)]$

      $C_{t+1}(i,s,a) \leftarrow (1-\alpha_t(i)) \cdot C_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [C_t(i,s',a^*) + V_t(a^*,s')]$
      $N \leftarrow N+1$
    append $childSeq$ onto front of seq
    $s \leftarrow s'$
return seq
{end MAXQ-Q}

2.4 Ant Colony Optimization

Mimicking the foraging behaviors of a colony of ants, an algorithm can be used to model the ants activities and the natural strategy used by the colony. ACO attempts to follow these ideas based on the laying of pheromone across the ant's path to mark the path taken [5]. Using the concept of a metaheuristic, ACO can be used to provide solutions for many NP-complete problems [5]. Since the emerging behavior for following the basic ACO metaheuristic algorithm, Algorithm 4, 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 [3].

Algorithm 4 The Ant Colony Optimization Metaheuristic [5]

```
1: Set parameters, initialize pheromone trails
2: while termination condition not met do
3:    for m = 1 to M do
4:       {Construct ant solutions}
5:       Initialize partial solution, S_m^p = ∅
6:       repeat
7:          Select a feasible solution component using stochastic selection, N(S_m^p)
8:          {Selection based on Equations 2.18 and 2.19}
9:          Add solution component to partial solution, S_m^p ∪ N(S_m^p)
10:      until S_m^p is a full solution
11:      {Apply local search (optional)}
12:      Perform local search to solutions found (problem specific)
13:   {Update pheromone}
14:   Decrease all pheromone through evaporation
15:   Increase pheromone associated with the set of good solutions
16:   {Pheromone updates based on Equations 2.22 and 2.23}
```

Following the ACO metaheuristic several versions have been created to handle different NP-complete problems by creating a unique model for the assigned problem. It has covered domains from the TSP to the quadratic assignment problem. This range of problems has made ACO a viable algorithm for those complex domains [5]. These ACO variations not only generate solutions for specific problems, but are also enhancements to the metaheuristic including modifications to the algorithm [3]. These variations are discussed in the following sections.

2.4.1 Ant Colony System. The ant colony system (ACS) uses the ACO metaheuristic to mimic the real world behavior of an ant colony and its method of finding and remembering a food source location [3]. This memory is created by the introduction of a pheromone matrix used by ACS to save the results of an ant learning a path. ACS uses a positive feedback method to reinforce the better solution found by a set of ants [3]. The stochastic nature of the ant system allows a solution to escape from local minima and promote better exploration of the solution space. Since ACS is considered a local search, an optimal solution is not guaranteed [6]. However, the computation requirements are significantly less for ACS than a deterministic global search [3].

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. Each ant finds a solution based on a series of next city selections. ACS 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

$$j = \arg\max_{u \in J_i^k} \{[\tau_{iu}(t)] \cdot [\eta_{iu}]^\beta\}, \tag{2.18}$$

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

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)] \cdot [\eta_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}(t)] \cdot [\eta_{il}]^\beta}. \tag{2.19}$$

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 selection another path away from the strongest

pheromone, and the closest city selection is a local-greedy search [3]. After an ant has selected the next city, Equation 2.20 shows the local update to the pheromone on the selected edge.

$$\tau_{ij}(t) \leftarrow (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \tau_0, \tag{2.20}$$

where ρ is a constant and τ_0 is

$$\tau_0 = (n \cdot L_{nn})^{-1}, \tag{2.21}$$

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, the shortest path and length are updated. A global pheromone update is then made based on the best solution currently found

$$\forall (i,j) \in T^+, \tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta\tau_{ij}(t), \tag{2.22}$$

where

$$\Delta\tau_{ij}(t) = 1/L^+, \tag{2.23}$$

and T^+ and L^+ are the shortest tour and length respectively. This process of ants finding solutions and updating pheromone is repeated for a constant number of time steps. The last shortest path saved is the best path found through all episodes and the pheromone matrix represents the learning performed by the algorithm. Since it is a valid documented stochastic approach to solving the TSP, ACS provides a solid algorithm to build the clustering onto and compare results against [3].

2.4.2 Other Variations. There are other versions of ACO used to generate solutions for problems. The first documented ACO algorithm, ant system's (AS) main characteristic is that pheromone values are updated by all ants who generate a solution each iteration. Similar to ACS's global pheromone update, AS updates the

pheromone on a selected edge by

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \sum_{k=1}^{M} \Delta\tau_{ij}^{k}, \tag{2.24}$$

where ρ is the evaporation rate, M is the number of ants, and $\Delta\tau_{ij}^{k}$ is

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k, & \text{if ant } k \text{ used edge } (i,j) \text{ in its tour,} \\ 0, & \text{otherwise} \end{cases}, \tag{2.25}$$

where Q is a constant, and L_k is the length of the tour generated by ant k [5]. The edge selection in AS is determined by the probability calculation,

$$p_{ij}^{k} = \begin{cases} \frac{\tau_{ij}^{\alpha}\cdot\eta_{ij}^{\beta}}{\sum_{c_{il}\in N(S_k^p)}\tau_{il}^{\alpha}\cdot\eta_{il}^{\beta}}, & \text{if } c_{ij} \in NS_k^p, \\ 0, & \text{otherwise} \end{cases}, \tag{2.26}$$

where $N(S_k^p)$ is the set of possible edges and η_{ij} is the visibility function,

$$\eta_{ij} = \frac{1}{d_{ij}}, \tag{2.27}$$

where d_{ij} is the distance between cities i and j [3].

The next variation of ACO, MAX-MIN ant system (MMAS)changes two aspects of AS. It allows only the best ant to update the pheromone and bounds the pheromone value [5] [19]. Modifying Equation 2.24 for the single best ant pheromone update results in

$$\tau_{ij} \leftarrow [(1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{best}]_{\tau_{min}}^{\tau^{max}}, \tag{2.28}$$

where τ_{max} and τ_{min} are the pheromone bounds [5]. If the resultant pheromone value is outside the declared bounds, it is changed to the nearest limit [19]. Using the best

ant also modifies Equation 2.25 by removing the constant Q resulting in

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/L_{best}, & \text{if } (i,j) \text{ is in the best tour,} \\ 0, & \text{otherwise} \end{cases}, \tag{2.29}$$

where L_{best} is the length of the best tour. This value may be either the best tour found that iteration, iteration-best, the best solution found through all iterations, best-so-far, or a combination of both [19].

2.5 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 and deterministic algorithms are not available. Therefore, stochastic algorithms to estimate the final cluster solutions have been develop to provide near-optimal solutions with significantly less computational times. These algorithms provide the TSP task decomposition used to merge ACO domain with HRL methods. This research looks at two common clustering algorithms which meet this goal, k-means and G-means.

2.5.1 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 a two dimensional data set into clusters based on the given attribute values [13]. The mathematical objective is to minimize the global error of all clusters [13]

$$\min E = \sum_{i=1}^{k} \sum_{x_j \in S_i} |x_j - \mu_i|^2, \tag{2.30}$$

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.

This stochastic nature of the initial means determination causes issues with the optimality of the final solution. Documented performance of the algorithm has shown the final solution quality depends heavily on the initial mean selection [9]. With the presented algorithm, no care is taken with the initial selection and the solution may actually decrease the effectiveness of the partitioning efforts. The other issue with the k-means algorithm is the set number of clusters, predetermined before the algorithm begins. With an incorrect number of clusters, results are varied and of low quality [9].

2.5.2 G-Means. The issues with k-means clustering stem from the assumptions made about the data set. The center-based clustering algorithms, k-means included, assume the data is in a unimodal distribution, such as Gaussian [9]. Therefore, only one mean should represent the data in that distribution. Using too many means create a complex and inefficient representation. Likewise, using too few abstracts the distribution differences and creates a too simple data representation [9]. 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 [9].

The algorithm performs the clustering in the same manner as k-means with a possibly 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. Hamerly discusses the use of the Anderson-Darling

statistic in determining if a mean should be split or not [9]. The statistical calculation to be examined, modified for mean estimation [9]

$$A^2_*(Z) = A^2(Z)(1 + \frac{4}{n} - \frac{25}{n^2}), \tag{2.31}$$

where

$$A^2(Z) = -\frac{1}{n}\sum_{i=1}^{n}(2i-1)[\log(z_i) + \log(1 - z_{n+1-i})] - n \tag{2.32}$$

These equations provide the test for the split of a cluster. The algorithm follows these steps to determine is 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. The first is to select a small vector, m, which provides two new means at $\mu \pm m$ [9]. The second method finds the main principal component s of the data (with an eigenvalue of λ) and produces

$$m = s\sqrt{\frac{2\lambda}{\pi}}, \tag{2.33}$$

which provides the two new centers [9]. For this application, we have selected the first version to avoid complex calculations within the partitioning method. 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

$$\forall x \in X, x' = \langle x, v\rangle / \|v\|^2, \tag{2.34}$$

and transforms it to a mean of 0 and variance of 1. Finally, z_i is calculated and substituted into Equation 2.31. If $A^2_*(Z)$ is within the confidence level α, then reject new means and replace 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 [9]. By using G-means,

the clusters will not have a predetermined k value and should more accurately and effectively partition the data.

2.6 Summary

The methodologies presented in this chapter represent the basis for this research. The problem decomposition by a hierarchical reinforcement learning method, allow a large and complex problem to be solved individually and combined in some hierarchical manner to produce a feasible solution. Whether the decomposition be done to use a set of machines, temporal state *options*, or a hierarchy of tasks, the smaller problems can be solved faster and with less computational requirements. In addition, the introduction of ant colony optimization as a possible reinforcement learning method will provide insight into the possibility of transferring the HRL techniques to its domain.

Algorithm 5 ACS-TSP Algorithm [3]

/* Initialization */
for every edge (i, j) **do**
  $\tau_{ij} = \tau_0$
Generate a candidate list for each city
for $k = 1$ to total number of cities **do**
  Place ant k on a randomly chosen city
Let T^+ be the shortest tour found and L^+ be its length
/* Main Loop */
for $t = 1$ to t_{max} **do**
  for $k = 1$ to M **do**
    /*Solution Construction*/
    Get a local copy of all candidate lists
    Build tour t^k by doing the following $n - 1$ times:
    if the candidate list is not empty **then**
      Choose the next available city, $j \in J_i^k$ in the candidate list as follows:

$$j = \begin{cases} argmax_{u \in J_i^k} \{[\tau_{iu}]^\alpha \times [\eta_{iu}]^\beta\}, & \text{when } q \leq q_0 \\ J, & \text{otherwise} \end{cases}$$

      where $J \in J_i^k$ is chosen according to the probability:

$$p = \frac{[\tau_{iu}]^\alpha \times [\eta_{iu}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}]^\alpha \times [\eta_{il}]^\beta}$$

      and where i is the current location
    else
      choose the closest $j \in J_i^k$
    Remove j from all local candidate lists
    /* Local Pheromone Update */
    After each transition ant k applies the local update rule:
    $\tau_{ij} = (1 - \rho) \times \tau_{ij} + \rho \times \Delta\tau_0$
  for $k = 1$ to total number of cities **do**
    if an improved tour is found by ant k **then**
      update T^+ and L^+
  /* Pheromone Reinforcement */
  for every edge (i,j) $\in T^+$ **do**
    Update pheromone trails by applying the rule:
    $\tau_{ij} = (1 - \rho) \times \tau_{ij} + \rho \times \Delta\tau_{ij}, \Delta\tau_{ij} = 1/L^+$
/* Done */
print the shortest tour and its length

III. MAXQ with Ant-Q Learning

This chapter introduces ant colony optimization to the concepts promoted in Dietterich's MAXQ. It replaces the basic reinforcement learning methods used, Q-learning and SARSA, with a basic ant colony optimization algorithm, Ant-Q. Exploiting the similarities between Q-learning and SARSA, the modification of MAXQ to use Ant-Q is completed with only a few modifications. The major difference stems from Ant-Q's use of multiple agents or ants during every episode and at each level. These additional ants provides the value function with more updates and more state information. The domain used to showcase the modified algorithm is the taxi world problem.

3.1 Taxi World Problem

Described by Dietterich [4] and used throughout his research as an example domain, 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 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 [4]. The tasks are identified in Figure 3.1, where the leaf nodes are primitive actions and the parents subtasks.

There are six defined primitive actions in this domain (a) four navigation actions that move the taxi one square `North`, `South`, `East`, `West`, (b) a `Pickup` action, and (c) a `Putdown` action. The rewards are -1 for each action, +20 for successfully delivering a passenger, -10 if the taxi attempts to execute the `Pickup` or `Putdown` illegally. 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 [4]. Figure 3.2 shows an example 5x5 Taxi World problem. The labeled squares represent the

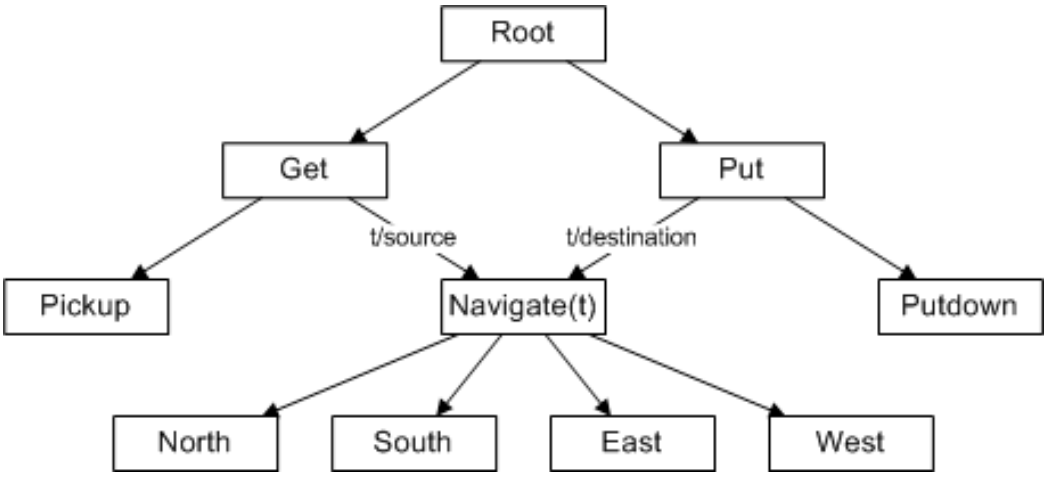


Figure 3.1: A task hierarchy graph for the Taxi World Problem.

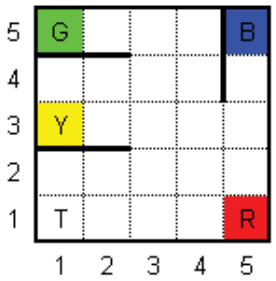


Figure 3.2: A sample problem grid (5x5) for the Taxi World

Taxi (T), the four possible destinations (R, G, B, and Y), the passenger is picked up from one destination and dropped off at another. The heavy lines indicate walls or obstacles the taxi cannot cross (adding constraints to the problem). All problems used in this research are formed similar to this example, with varying sizes, number of obstacles, and destination locations. This domain lends itself to the natural task decomposition shown in Figure 3.1.

3.2 Ant-Q

Based on the basic ACO algorithm, Ant-Q was developed to merge Q-learning with the ACO concept [7]. 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 [7].

However, Ant-Q uses a different learning method than Q-learning or SARSA. Ant-Q uses a pseudo Q-value function as the value function with a similar but varied learning function:

$$AQ(s,a) \leftarrow (1-\alpha) \cdot AQ(s,a) + \alpha \cdot [\Delta AQ(s,a) + \gamma \max_{a'} AQ(s',a')], \tag{3.1}$$

where AQ(s,a) is equivalent to Q(s,a) 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 [7]. The global-best update is calculated by

$$\Delta AQ(s,a) = \begin{cases} W/C_k, & \text{if (s,a) is performed by global best ant k} \\ 0, & \text{otherwise} \end{cases}. \tag{3.2}$$

The iteration-best update is the same calculation, but based upon the iteration's best ant, not the global best,

$$\Delta AQ(s,a) = \begin{cases} W/C_k, & \text{if (s,a) is performed by iteration best ant k} \\ 0, & \text{otherwise} \end{cases}, \tag{3.3}$$

where W is a constant set to 10, based on the ant system value [7]. Gambardella and Dorigo selected the iteration-best as the main source of delayed reinforcement update based on results of both methods [7].

Similar to the ε-greedy selection of Q and SARSA, 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 [7]. This random action selection allows the algorithm to explore the state space more and is one of the foundations for Ant-Q [3]. The stochastic calculation is based on the probability

$$P_a(s) = \frac{[AQ(s,a)]^\delta \cdot [HE(s,a)]^\beta}{\sum_{u \in A_s}[AQ(s,u)]^\delta \cdot [HE(s,u)]^\beta}, if a \in A_s \tag{3.4}$$

where δ and β are constants, u is all action possible actions, and $HE(s,a)$ is a heuristic evaluation for the state action pair [7] (Note: δ is used in this text in place the common α parameter to avoid confusion with the HRL α parameter). The similarities between Ant-Q, Q-learning, and SARSA allow Ant-Q to be easily incorporated into any reinforcement learning algorithm but exploit the ACO metaheuristic.

3.3 MAXQ with Ant-Q Learning

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

1. Adaptation of the Ant-Q value function for primitive nodes (replaces $V_t(i,s)$)
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 (replaces $\tilde{C}_t(i,s,a)$ and $C_t(i,s,a)$)

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 new algorithm is shown in Algorithm 6.

The first modification replaces the value function with the Ant-Q value function. The Ant-Q value update changes from Equation 3.1 to

$$AQ_t(i,s) \leftarrow (1-\alpha_t(i)) \cdot AQ_t(i,s) + \alpha_t(i) \cdot r_t(i). \tag{3.5}$$

It replaces the ΔAQ with the reward for performing the primitive action. This sets the AQ value function in line 5 to the basic Ant-Q learning and promotes the local update of pheromone based on the primitive action selected.

The second modification implements the strength of ant colony optimization, allowing a set of ants to solve the current task. By looping through M ants, see line 9, the algorithm creates the colony and creates the global pheromone matrix. This matrix holds the Ant-Q value function value for primitive actions with 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. The probability allows the ants to explore the solution space more as it will generate a random action with probability, q_0, see line 11. 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 as described in Equation 3.1 can be modified to fit this desire. The reward

degradation is determined by the number of actions needed to make the current task terminal [4]. Looking at the modified algorithm, see line 17, there is a second Ant-Q value function created, a composite value. This composite value is used to identify not primitive tasks and provide a value for the higher level task with a state-action pair [4].

Algorithm 6 MAXQ-Q with Ant-Q Learning Algorithm

1: {function MAXQ-AntQ(MaxNode i, State s)}
2: let seq=() be the sequence of states visited while executing i
3: **if** i is a primitive MaxNode **then**
4: execute i, receive r, and observe result state s'
5: $AQ_{t+1}(i,s) \leftarrow (1-\alpha_t(i)) \cdot AQ_t(i,s) + \alpha_t(i) \cdot r_t$
6: push s onto the beginning of seq
7: **else**
8: let $count = 0$
9: **for** $k = 1$ to M **do**
10: **while** $T_i(s)$ is false **do**
11: choose an action a according to probability
$$P_a(s) = \begin{cases} \frac{[AQ(s,a)]^{\delta} \cdot [HE(s,a)]^{\beta}}{\sum_{u \in A_s} [AQ(s,u)]^{\delta} * [HE(s,u)]^{\beta}} & \text{if } q_0 \leq q \\ \text{random action } a \in A_s & \text{otherwise} \end{cases}$$
12: let $childSeq$ = MAXQ-AntQ(a,s), where $childSeq$ is the sequence of states visited while executing action a. (in reverse order)
13: observe result state s'
14: let $a^* = \arg\max_{a'}[A\tilde{Q}C_t(i,s',a') + AQ_t(a',s')]$
15: let $N = 1$
16: **for** each s in $childSeq$ **do**
17: $A\tilde{Q}C_{t+1}(i,s,a) \leftarrow (1-\alpha_t(i)) \cdot A\tilde{Q}C_t(i,s,a) + \alpha_t(i) \cdot \gamma^N[\tilde{R}_i(s') + A\tilde{Q}C_t(i,s',a^*) + AQ_t(a^*,s)]$
18: $AQC_{t+1}(i,s,a) \leftarrow (1-\alpha_t(i)) \cdot AQC_t(i,s,a) + \alpha_t(i) \cdot \gamma^N[AQC_t(i,s',a^*) + AQ_t(a^*,s')]$
19: $N \leftarrow N+1$
20: append $childSeq$ onto front of seq
21: $s \leftarrow s'$
22: return seq
23: {end MAXQ-AntQ}

3.4 Summary

These identified modifications are used to create a MAXQ-AntQ algorithm used to find a solution to multiple goal 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 produce solutions faster than the MAXQ-Q algorithm. This result provides the springboard to transfer the HRL techniques to the ant colony optimization domain and the traveling salesman problem, discussed in the following Chapter.

IV. Ant Colony System with Partioning

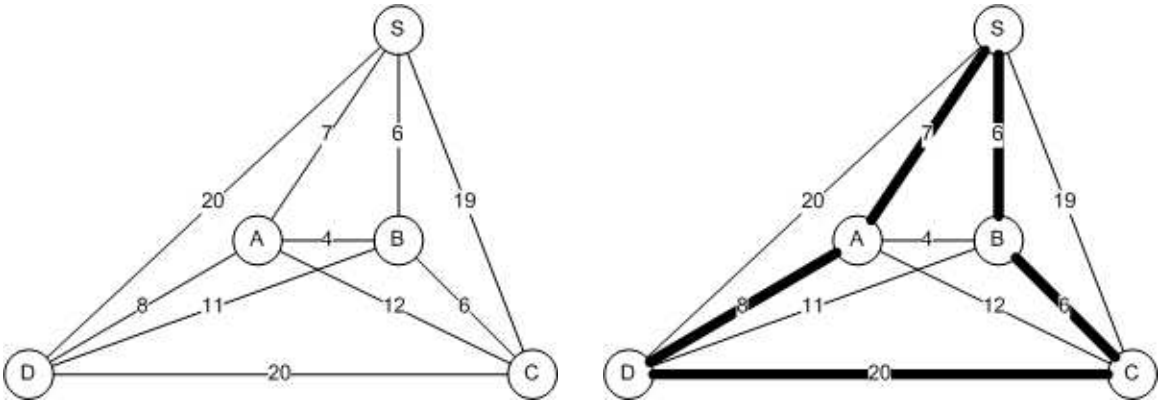
This chapter introduces the concepts promoted in Dietterich's MAXQ to the ant colony optimization domain. It exploits the task hierarchy used with the taxi world problem and implements a hierarchy for the traveling salesman problem (TSP). Data clustering algorithms are used to create these potential tasks within the TSP hierarchy. These clusters are considered primitive tasks and are solved individually to aid in the generation of a complete solution. The information presented in this chapter identifies the problem domain, discusses the clustering algorithms, and then presents the modifications done for this research.

4.1 Traveling Salesman Problem

The TSP is a problem used to demonstrate the complexity of the NP-complete domain. It is simply described as given a set of cities, find the least cost route to visit all cities and return to the starting city. In the standard version, used by this algorithm, the problem is assumed to be symmetric, meaning the cost of traveling from city a to city b is the same as traveling from city b to city a. Since the problem is NP-complete, there is no effective algorithm to solve the general TSP.

The complexity of the TSP is based on the combinatorics of constructing a path through all cities. If there are n cities, given a starting city, there are $n-1$ cities to choose as the next step in the path. The next city is chosen from a set of $n-2$ cities and so forth. Combining these values, results in a $(n-1)!$ search space. However, since the problem is symmetric, the space can be divided by 2, resulting in a final search space of $(n-1)!/2$ [1]. This fast growing search space limits the ability to use a deterministic algorithm to look for the optimal tour and stochastic algorithms are created to more efficiently locate a feasible tour [1].

A sample TSP is shown in Figure 4.1. It has a city cardinality of five and there are twelve possible tours to inspect. A tour consists of the ordered list of cities visited and the cost of the path. The cost of a tour is calculated by the sum of connecting edges in the tour, including the final edge from the last city back to the starting city.



(a) (b)

Figure 4.1: A sample TSP, (a) 5-cities and related edge costs (b) the optimal solution for this TSP.

4.2 Data Clustering

To exploit the benefits of HRL, the TSP needs a task hierarchy imposed on it. This research examines the use of data clustering to create this hierarchy. Two algorithms are reviewed, k-means and G-means, discussed in the next two sections. These algorithm partition the TSP into a set of subproblems, which when solved individually combine to generate the complete solution.

4.2.1 k-Means. Each clustering method has a different manner to determine the exact number of means used and the resultant size of the subproblems. The k-means algorithm meets this objective fairly well if the number of means, k, is chosen well. Following the guideline of reducing the overall problem to a TSP no larger than 50 cities, the algorithm performs a simple calculation to determine a constant k for the problem. By dividing the number of cities by 50, the partitioning algorithm determines the k desired. Although this manner does not remove the issues present with an inaccurate k value [13], it does allow the partitioning to aid in the overall efficiency of the ACS algorithm.

In order to achieve the best partitioning, the algorithm assumes a uniform distribution with the k-means algorithm and moving the means in order to have each

subproblem contain a desired number of cities, $\bar{C}$. This is an inaccurate assumption, as seen with the examination of the TSPs. Therefore, the majority of the subproblems are not balanced with the possibility of many subproblems only containing a few cities while large subproblems remain, producing inefficiency. This inefficiency leads to the addition of a better clustering algorithm to reduce the possibility of overlapping means and inefficient partitioning. The G-means algorithm was selected to produce the partitions, but also avoid the amount of issues possibly created by the constant k value [9].

Algorithm 7 k-Means Clustering Algorithm [13]

1: $k = n/\bar{C}$ {$\bar{C}$ is desired cluster size}
2: **for** $i = 1$ to k **do**
3: m_k = random coordinate within problem range
4: **repeat**
5: {Repeat calculations until no changes}
6: reset flag to no changes made
7: {Assign cities to cluster}
8: **for** $i = 1$ to n **do**
9: $assigned_n^t = \min_m(d(n, m))$
10: **if** $assigned_n^t \neq assigned_n^{t-1}$ **then**
11: flag changes made
12: {Recalculate cluster means}
13: **for** $i = 1$ to k **do**
14: $m_i^x = \frac{\sum_{j=1}^{n} j_x, where j \in C_i}{|C_i|}$
15: $m_i^y = \frac{\sum_{j=1}^{n} j_y, where j \in C_i}{|C_i|}$
16: **until** no changes made

4.2.2 G-Means. Using the documented benefit of G-means finding a better number of means to represent the clusters, this version of the clustering hierarchy algorithm uses the same principles presented in the k-means clustering but replaces the clustering technique with G-means. G-means expands off of k-means but uses a statistical test to determine if the current number of means is appropriate or if additional ones are needed. The changes are all front-loaded against the k-means algorithm, by placing another loop around the k-means clustering section of Algorithm 7.

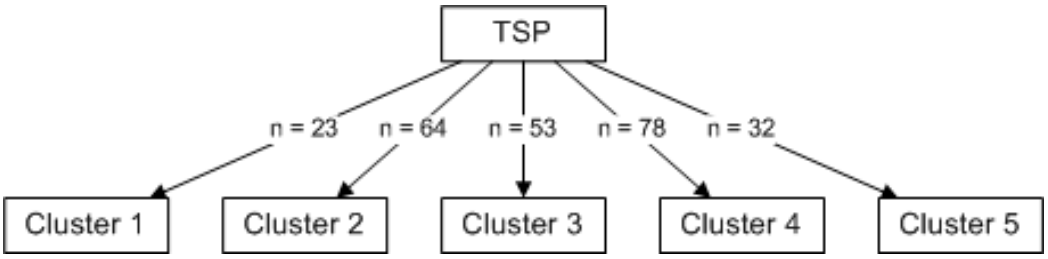


Figure 4.2: A sample hierarchy for a 250-city TSP with the number of cities assigned to each cluster.

The G-means clustering begins with 1 mean, performs k-means clustering, assigning all cities to the single cluster. The second step splits the current mean into two children. The children are selected by adding and subtracting a vector from the current mean, c to produce two new means located at $m \pm c$. The data set is then run through k-means again to assigned the cities to the two new child means and recalculate their location. Once the cities are assigned, the algorithm calculates the Anderson-Darling statistical test on the cities assigned to determine if the new distribution is accurately represented by the new means. If it is, the algorithm accepts the children and replaces the single old mean with the two new means. If not accurately represented, the children are rejected and the old mean remains. This process is repeated until there are no more accepted splits or all clusters are below minimum desired size. This minimum check avoids the possibility of clusters being too small and negating the benefits of using a ant colony optimization technique on the problem.

4.3 HRL ACS with Clustering

The combination of the ACS with a data-clustering algorithm is straight-forward. The objective is to effectively create a hierarchy of subtasks to partition 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 actual hierarchy. Figure 4.2 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

Algorithm 8 G-Means Clustering Algorithm [9]

1: $k = 1$ {start with 1 cluster}
2: m_1 = random coordinate within problem range
3: use k-means to assign cities and recalculate means {see k-means algorithm, lines 6-16)}
4: **repeat**
5: {Repeat actions until no new splits made}
6: **for** $i = 1$ to k **do**
7: **if** $|C_k| > \bar{C}$ **then**
8: let $X \leftarrow$ cities assigned to C_i
9: create children means, $m_{i_1} = m_i + c$ and $m_{i_2} = m_i - c$
10: perform k-means on X and new child means
11: calculate vector, $v = m_{i_1} - m_{i_2}$
12: let $X' \leftarrow$ projection of X onto v
13: transform X' to data set with $\mu = 0$ and $\sigma = 1$
14: let $Z \leftarrow$ the cumulative distribution function $\forall x_i \in X, z_i = CDF(x_i)'$
15: calculate $A^2(Z)$ for Z (Equation 2.32
16: calculate $A^2_*(Z)$ to adjust for means estimation (see Equation 2.31)
17: **if** $A^2_*(Z) > \alpha$ **then**
18: {α is desired critical value}
19: $m_i \leftarrow m_{i_1}$
20: $m_{k+1} \leftarrow m_{i_2}$
21: $k \leftarrow k + 1$
22: perform k-means on current means and all cities
23: **until** iteration with no splits completed

dynamically create the hierarchy is one of the areas Dietterich recommends as future extension to MAXQ [4]. Fortunately, TSPs have a potential partitioning method built into the domain, other domains are not as easily dynamically partitioned. Algorithm 9, shows the addition of clustering to the ACS-TSP algorithm, Algorithm 5. The algorithm follows the structure of MAXQ-AntQ, Algorithm 6. It is a recursive function that traverses the task hierarchy with the base cases of the cluster TSPs, the leaf nodes. If the parameter TSP is a base case, the function returns the best solution found by ACS-TSP. Otherwise, it clusters the TSP using either k-means or G-means and calls HRL-ACS-C on each child cluster. The solutions for all children are then combined using a local greedy search.

Algorithm 9 HRL ACS with Clustering Algorithm

{function HRL-ACS-C(TSP t)}
let $T_{best} = \emptyset$ 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) {Algorithm 5}
else
  {Generate clusters and solutions for each subproblem}
  let C = the set of clusters in t {by k-means or G-means, Algorithms 7-8}
  let $TOURS = \emptyset$ be the set of tours for all clusters
  for each $c \in C$ **do**
    $TOURS(c)$ = HRL-ACS-C($TSP(c)$)
  {Create a TSP for the means of the clusters}
  let M = the set of means of C
  T_m = ACS-TSP($TSP(M)$) {Algorithm 5}
  {Combine the tours generated for all subproblems}
  randomly select first edge, $(i, j) \in T_m$
  find edge $(q, r) | d(q, r) = \min d(\forall x \in C_i, \forall y \in C_j)$
  let $T_i = TOURS(i)$ be the tour for cluster i
  select start city $s_0 \in C_i$ where edge $(q, s_0) \in T_i$
  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 $C_l \in T_m$ **do**
    assign $q \leftarrow T_{best}^{last}$
    find edge $(q, r) | d(q, r) = \min d(1, \forall y \in C_l)$
    add T_l to T_{best} with starting city r
  assign $q \leftarrow T_{best}^{last}$
  add edge (q, s_0) to T_{best}
return T_{best}
{end HRL-ACS-C}

4.4 Summary

This new algorithm provides the framework to dynamically create a task hierarchy similar to the MAXQ decomposition discussed in Chapter II. This application of HRL techniques to the ACO domain should show the ability for complex large problems to be decomposed in a logical manner and solved efficiently and effectively. The ability to dynamically create the hierarchy is a by product of using the TSP as the

test problem. TSP has a logical task hierarchy in geographical location, but this will not be true for all problems. The key to adapting the HRL methods to any complex problem is the task decomposition, which the ACS with clustering can provide.

V. Results

This chapter discusses the testing the HRL ACO algorithm on two problem domains as detailed in Chapters III and IV. There were two set of tests conducted, the first, on the taxi world problem, includes various problems of increasing state space size and compares mean training run of convergence between the MAXQ-Q and MAXQ-AntQ algorithms. The second, on TSP, includes problems increasing in city cardinality and compares tour statistics and time of ACS-TSP, ACS with k-means, and ACS with G-means. The implementations introduced in this research 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.

5.1 MAXQ with Ant-Q Learning

To assess the feasibility of merging ACO with MAXQ HRL techniques, episodic runs were performed on a set of taxi world problems with both algorithms. Problems with sizes of 5x5, 7x7, 10x10, 20x20, 30x30, and 50x50 resulting in 500-50,000 states were tested. The problem list is shown in Table 5.1 with details in Appendix A. Data collected includes the number of moves in the current solution found by the algorithm, based on the training run number and the training run of convergence. Since the learning algorithm/equations are the only difference between MAXQ-ANT and MAXQ-Q, the focus is on the convergence of the algorithms to the known optimal solution. Although not guaranteed to find the optimal solution, the algorithms tended 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 a maximum of 1000 training runs or until the convergence of the solution was within a set percentage of 1.0%.

To see if the Ant-Q modifications are valid as a learning method within MAXQ, the data collected matches the documented abilities of the MAXQ-Q algorithm based the convergence of the learning algorithms. MAXQ-Q uses a cumulative mean reward

Table 5.1: Selected Taxi World Problems (refer to Appendix A to see graphical representation)

Size	# States	Optimal Policy
5x5	500	14 primitive actions
7x7	980	24 primitive actions
10x10	2000	21 primitive actions
20x20	8000	59 primitive actions
30x30	18000	56 primitive actions
50x50	50000	215 primitive actions

to monitor convergence. These tests use the count of the sequence of moves performed each training run.

5.1.1 Experiment Setup. The problems selected test the ability of the algorithm to solve multi-goal problems with increasing state space size. Exact problems are shown in Appendix A. Table 5.1 shows the pertinent information for each problem. With the final problem of 50,000 possible states, the testing shows the modified algorithm can handle the larger more complex problems.

Parameters for each learning method were kept consistent with the authors' documented parameters. Dietterich uses a relatively large value for γ, between 0.7 and 0.9, and a small value for α, between 0.1 and 0.25 [4]. These values are taken from the development of Q-learning and the tests done within Watkin's Q-learning [22]. The tests in this work use a γ value of 0.9 and α value of 0.1. These values allow the eligibility trace of the reward to be propagated across all action-state values and the value update to be taken in relatively small increments. For the Ant-Q parameters the Gambardella and Dorigo tested values are used, β is 2 and δ is 1 [7].

Each algorithm was allowed a maximum of 1000 training runs, each consisting of a single episode. An episode is considered complete when a solution is returned which meets the four goals dictated by the taxi world problem. The solution must contain a sequence of events navigating the taxi to the pickup location, picking up the passenger, navigating to the destination location, and dropping the passenger off. The convergence of the function value space is compared after each training episode. If the

Table 5.2: Selected Parameters for MAXQ-Q and MAXQ-AntQ Testing

Parameter	**Value**
γ	0.9
α	0.1
β	2.0
δ	1.0
T_{max}	1000
Convergence	1.0%

difference from Q_{t-1} to Q_t is less than 1.0%, the algorithm is considered converged and execution is halted. The data collected includes the sequence of primitive actions in each training run's solution and the training run convergence is calculated if any. All parameters are listed in Table 5.2.

5.1.2 Solution Convergence. The MAXQ-Q algorithm provides a documented baseline for the taxi world problem domain. Examining the data in Table 5.3, the modified algorithm MAXQ-AntQ discussed in Chapter III converges in significantly fewer training runs than Dietterich's original algorithm. 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.

The improvement is an average 12.1% faster convergence rate by using the HRL-ACO-MAXQ algorithm over MAXQ-Q. The best improvement is noted in the 7x7 sized problem, a 17.8% faster mean convergence rate. This improvement stems from the ability of MAXQ-AntQ to use a set of agents at each level in the hierarchy rather than the single agent used in MAXQ-Q [4]. 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

Table 5.3: Convergence data for MAXQ and MAXQ-AntQ on all selected Taxi World Problems. Numbers represent the training run number of convergence (*training runs capped at 1000, bolded represents statistically significantly faster convergence).

	MAXQ			**MAXQ-AntQ**		
Problem	**Best**	**Worst**	**Mean**	**Best**	**Worst**	**Mean**
5x5	94	124	112±15	83	114	**97±17**
7x7	109	155	129±21	91	137	**106±19**
10x10	215	257	232±29	184	249	**214±32**
20x20	487	504	496±14	378	461	**421±41**
30x30	744	812	781±48	687	757	**724±49**
50x50	944	1000*	984±21	811	997	**876±46**

MAXQ-Q, not only does the learning rate increase, the combination of these two concepts lends itself to other domain combination possibilities.

5.2 ACS with clustering

The second set of tests incorporates the hierarchical decomposition of the TSP with the ACS-TSP algorithm. The tests included problems ranging in size from 48 cities to 50,000 cities. These problems were tested by the three algorithms outlined in Chapter IV, 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, run time, and clustering for the two modified algorithms. The results identify a tradeoff between solution time and size versus solution quality.

5.2.1 Experiment Setup. The problems selected are listed in Tables 5.4 and 5.5. They vary in size from 48 cities to 50,000 cities. These problem were selected to show the trends of each algorithm as the problem size grows. The problems taken from TSPLIB [16] are listed in Table 5.4. In addition, several randomly created problems are used to show the differences between the three algorithms and highlight the effects of clustering on non-geographically modeled problems, listed in Table 5.5. Since these algorithms were created for the sole purpose of this experiment, no known

Table 5.4: Selected TSPLIB Traveling Salesman Problems (Optimal tour length taken from TSPLIB [16]).

Name	Number of Cities	Distance Measurement	Optimal Path
att48	48	ATT	10628
eil51	51	Euclidean	426
berlin52	52	Euclidean	7542
eil76	76	Euclidean	538
eil101	101	Euclidean	629
lin105	105	Euclidean	14379
bier127	127	Euclidean	118282
ch150	150	Euclidean	6528
pr226	226	Euclidean	80369
a280	280	Euclidean	2579
fl417	417	Euclidean	11861
att532	532	ATT	27686
rat783	783	Euclidean	8806
pr1002	1002	Euclidean	259045
d1291	1291	Euclidean	50801
vm1748	1748	Euclidean	336556
u2152	2152	Euclidean	64253
rl5915	5915	Euclidean	[565040,565530]
rl11849	11849	Euclidean	[920847,923368]
usa13509	13509	Euclidean	[19947008,19982889]
brd14051	14051	Euclidean	[468942,469445]
d18512	18512	Euclidean	[644650,645488]

optimal value is available to compare against, however each algorithm's solutions can be contrasted. In addition, the larger problems show the ability for the modified algorithms, HRL ACS with k-means and G-means, to handle larger problems than TSP-ACS can in a feasible amount of time.

As before, these experiments used the parameters documented by Dorigo in all three algorithms [6], notably β is 2, δ is 1, ρ is 0.1 and T_{max} is 1000. As a divergence from ACS-TSP, two parameters M, the number of ants and $|CL|$, the size of the candidate list were changed based on documented improvements [19]. The number of ants is set to the number of cities in the problem, with a maximum value of 100. The candidate list size with set equal to one third the number of cities, with a maximum

Table 5.5: Selected Random Traveling Salesman Problems.

Name	Number of Cities	Distance Measurement
rand75-1	75	Euclidean
rand75-2	75	Euclidean
rand75-3	75	Euclidean
rand100-1	100	Euclidean
rand100-2	100	Euclidean
rand100-3	100	Euclidean
rand200-1	200	Euclidean
rand200-2	200	Euclidean
rand200-3	200	Euclidean
rand500	500	Euclidean
rand750	750	Euclidean
rand1000	1000	Euclidean
rand1500	1500	Euclidean
rand2000	2000	Euclidean
rand5000	5000	Euclidean
rand7500	7500	Euclidean
rand10000	10000	Euclidean
rand25000	25000	Euclidean
rand50000	50000	Euclidean

value of 100. These limitations were placed to ensure the large problems could be solved on the test machine with the amount of memory available. The final parameter in ACS-TSP is the value of τ_0, this was calculated using the nearest neighbor heuristic tour solution's length. This sets the initial pheromone value to the inverse tour length. The final parameters are needed by the two clustering algorithms. In k-means, the number of clusters was set to be equal to the number of cities divided by 50 plus one. This ensured a mean cluster size of approximately 50. In G-means, the statistical test used by the algorithm required a confidence value. This was selected to be 0.0001, which determines the critical value for the Anderson-Darling test to pass or not [9]. The algorithm also includes a minimum cluster size of 25. This ensures a cluster with a smaller size cannot be split, regardless of the distribution. All parameters are listed in Table 5.6.

Table 5.6: Selected Parameters for ACS-TSP and HRL ACS with Clustering Testing

Parameter	**Value**
β	2.0
ρ	0.1
T_{max}	1000
M	Number of Cities (max 100)
$\|CL\|$	(Number of Cities)/3 (max 100)
k	[(Number of Cities)/50]+1
α (G-means)	0.0001
Minimum cluster size (G-means)	25

Each algorithm was run on the 41 problems identified for 30 episodes each problem. An episode is considered a set of 1000 time steps. The algorithm was reset after each episode and the data collected before reset. Data collected includes the tour statistics and run time for the three algorithms, also clustering statistics for the two modified algorithms were collected. 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. With the test showing a normal distribution, the differences cited between algorithms were statistically calculated by a t-test, with an α of 5%.

5.2.2 Tour Statistics. Tables 5.7-5.12 show the tour statistics for the three algorithms. In Tables 5.7 and 5.8 there are several problems labeled as "—"; these problems have run times that exceed 7 days (86,400 seconds) and are considered unobtainable based on the total estimated run time of the algorithm for the problem. Significance of solutions and run times are determined by t-test with an α of 0.05. 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 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, refer to Table 5.15.

The reason for the difference in solution quality comes from the combination of the subproblems to generate a complete solution. With the local greedy selection of the connecting edge, the HRL ACS algorithm makes several inefficient selections. Figures 5.3.c and 5.4.c show the edges selected in *rand200-3* to combine the subproblems to form a complete solution. These edges are not the optimal connections and many cause a decrease in solution quality.

The largest difference between ACS-TSP and k-means was on problem *fl417*, a large disparity of 22.9% further away from optimal. However, on problem *u2152*, k-means performed better than ACS-TSP, finding solutions 1.5% closer to optimal. Although these better solutions appeared only on one problem, it does show the k-means algorithm can be useful on large problems. Since k-means produced better results than G-means, there is no apparent benefit from selecting G-means over the tested k-means algorithm. G-means' worst problem was also *fl417* with solutions 33.6% longer than optimal than ACS-TSP. G-means only produced significantly better results than ACS-TSP on one problem *pr1002*, with 1.0% shorter solutions. However, G-means never out performs k-means significantly, with an average of 4.5% longer solutions. Since both HRL ACS algorithms perform worse, it indicates a few possibilities. The hierarchy of clusters was not created correctly (bad clustering), the final solution generation was inefficient, or clustering leads to a non-optimal partitioning of the problem space.

Examining results on the randomly generated problems, Tables 5.8, 5.10, and 5.12, demonstrates two benefits in using HRL ACS with clustering. Not only are the HRL ACS algorithms much faster, but they also have no significant difference in solution quality; or if they do, it is a better solution than ACS-TSP generates. These random TSPs were created using a normal distribution, resulting in natural clusters and a unique city layout. These problems also identify how the HRL ACS with clustering can provide solutions for very large problems in a feasible amount of time.

Table 5.7: Tour Length Statistics for ACS-TSP on TSPLIB TSPs. A "—" in results indicates run time was greater then 7 days.

Name	Best Tour	Worst Tour	Mean Tour	% from Optimal
att48	11233	11486	11370.40 ± 77.90	6.99%
eil51	444	480	466.73 ± 7.83	9.56%
berlin52	7569	7876	7765.10 ± 100.10	2.96%
eil76	566	584	574.90 ± 5.51	6.86%
eil101	699	728	715.47 ± 7.31	13.75%
lin105	15445	15984	15707.30 ± 153.72	9.24%
bier127	123891	130582	127370.20 ± 1475.56	7.68%
ch150	7077	7177	7129.10 ± 36.15	9.21%
pr226	88741	90092	89151.25 ± 640.86	10.93%
a280	3101	3213	3161.50 ± 38.63	22.59%
fl417	13937	15496	14064.62 ± 185.62	18.58%
att532	33778	34458	34118.00 ± 480.83	23.23%
rat783	10447	11824	10975.00 ± 106.84	24.63%
pr1002	351449	352760	352104.50 ± 927.02	35.92%
d1291	62454	64885	62448.00 ± 1645.58	22.93%
vm1748	410051	488644	422641.00 ± 4211.16	25.58%
u2152	76878	90313	82465.00 ± 1076.06	28.34%
rl5915	—			
rl11849	—			
usa13509	—			
brd14051	—			
d18512	—			

Overall, tour statistics show ACS-TSP is the best of the three algorithms to provide closer to optimal solutions. The two clustering algorithms provide feasible solutions but have no significant benefit to the quality of solutions; in fact they are significantly worse solutions as outlined above. These solution differences may be acceptable with certain problems, but the benefit of using clustering is evident in the run time to produce these solutions.

5.2.3 Timing. Tables 5.16 and 5.17 show the mean episode run time for each algorithm on all problems, measured in seconds. Comparing the data, once a problem could be clustered, there was a significant decrease in run time from the

Table 5.8: Tour Length Statistics for ACS-TSP on Random TSPs. A "—" in results indicates run time was greater then 7 days.

Name	**Best Tour**	**Worst Tour**	**Mean Tour**
rand75-1	374	385	381.60 ± 3.17
rand75-2	346	366	356.00 ± 5.19
rand75-3	365	380	374.10 ± 4.56
rand100-1	418	427	423.10 ± 2.81
rand100-2	383	393	388.60 ± 3.63
rand100-3	423	436	428.70 ± 4.42
rand200-1	598	605	601.50 ± 4.95
rand200-2	621	628	624.50 ± 4.95
rand200-3	618	624	620.45 ± 3.71
rand500	1011	1120	1020.44 ± 16.76
rand750	1116	1265	1185.05 ± 20.19
rand1000	1585	1684	1621.55 ± 23.59
rand1500	1754	1944	1798.46 ± 46.45
rand2000	2107	3106	2345.00 ± 58.13
rand5000	—		
rand7500	—		
rand10000	—		
rand25000	—		
rand50000	—		

Table 5.9: Tour Length Statistics for HRL ACS with k-Means Clustering on TSPLIB TSPs.

Name	**Best Tour**	**Worst Tour**	**Mean Tour**	**% from Optimal**
att48	11217	11530	11375.30 ± 96.13	7.03%
eil51	448	500	460.20 ± 16.03	8.03%
berlin52	7769	8359	8124.20 ± 214.50	7.72%
eil76	576	600	586.20 ± 7.08	8.96%
eil101	700	767	726.30 ± 22.18	15.47%
lin105	16802	18180	17462.00 ± 541.16	21.44%
bier127	131945	154366	138128.40 ± 6177.48	16.78%
ch150	7095	8129	7550.30 ± 329.26	15.66%
pr226	93786	103547	99489.40 ± 3124.75	23.79%
a280	3221	3410	3292.90 ± 53.97	27.68%
fl417	16059	17663	16775.50 ± 571.45	41.43%
att532	34082	36353	35205.90 ± 697.50	27.16%
rat783	10648	10912	10781.00 ± 81.99	22.43%
pr1002	320359	326770	323613.20 ± 2528.03	24.93%
d1291	68916	71739	70242.00 ± 1419.25	38.27%
vm1748	425496	438887	433516.00 ± 5678.58	28.81%
u2152	80440	83078	82136.00 ± 995.27	27.83%
rl5915	772771	781951	777361.00 ± 6491.24	37.58%
rl11849	1242933	1243809	1243371.00 ± 619.43	35.02%
usa13509	27080659	27550067	27329690.00 ± 160457.00	37.01%
brd14051	613678	622333	618191.60 ± 2638.25	31.83%
d18512	843005	854439	850101.10 ± 8161.80	31.87%

Table 5.10: Tour Length Statistics for HRL ACS with k-Means Clustering on Random TSPs.

Name	**Best Tour**	**Worst Tour**	**Mean Tour**
rand75-1	369	399	384.30 ± 9.25
rand75-2	339	372	358.20 ± 11.39
rand75-3	379	399	388.70 ± 7.56
rand100-1	448	476	456.70 ± 8.64
rand100-2	402	472	435.30 ± 18.92
rand100-3	446	481	461.50 ± 10.91
rand200-1	577	643	620.00 ± 20.42
rand200-2	578	651	625.00 ± 22.46
rand200-3	605	647	630.60 ± 13.35
rand500	949	1039	983.80 ± 28.06
rand750	1110	1169	1134.50 ± 26.49
rand1000	1355	1390	1375.90 ± 14.07
rand1500	1629	1696	1665.00 ± 23.21
rand2000	1897	1970	1925.90 ± 21.17
rand5000	2665	2725	2695.30 ± 20.09
rand7500	2907	2982	2947.60 ± 25.23
rand10000	3032	3126	3068.20 ± 27.33
rand25000	23998381	24190544	24103771.10 ± 67368.91
rand50000	234985706	237348635	235990494.70 ± 777856.22

Table 5.11: Tour Length Statistics for HRL ACS with G-Means Clustering on TSPLIB TSPs.

Name	**Best Tour**	**Worst Tour**	**Mean Tour**	**% from Optimal**
att48	12519	12883	12684.40 ± 129.44	19.35%
eil51	448	511	485.80 ± 21.34	14.04%
berlin52	8194	8369	8299.40 ± 61.93	10.04%
eil76	574	604	583.10 ± 9.45	8.38%
eil101	740	785	772.10 ± 14.43	22.75%
lin105	19458	20958	20189.70 ± 473.46	40.41%
bier127	142935	152570	145971.60 ± 2757.17	23.41%
ch150	7618	7864	7698.10 ± 74.77	17.92%
pr226	100659	110497	107432.10 ± 3188.78	33.67%
a280	3187	3424	3278.20 ± 60.34	27.11%
fl417	17237	18494	18047.40 ± 362.37	52.16%
att532	33978	36124	34971.90 ± 565.17	26.32%
rat783	10694	10986	10881.70 ± 85.65	23.57%
pr1002	318020	324915	318020.00 ± 2227.14	22.77%
d1291	67624	68852	68338.00 ± 447.15	34.52%
vm1748	445692	452338	449367.00 ± 2114.75	33.52%
u2152	83488	84743	83949.90 ± 388.47	30.66%
rl5915	797563	810057	803460.40 ± 5560.79	42.20%
rl11849	1382245	1405399	1393561.00 ± 8976.97	51.33%
usa13509	28711849	28839532	28778014.00 ± 56539.92	44.27%
brd14051	642637	648140	646013.50 ± 1951.86	37.76%
d18512	887854	894026	890488.10 ± 2280.85	38.14%

Table 5.12: Tour Length Statistics for HRL ACS with G-Means Clustering on Random TSPs.

Name	**Best Tour**	**Worst Tour**	**Mean Tour**
rand75-1	416	447	437.70 ± 8.83
rand75-2	368	394	378.90 ± 8.58
rand75-3	383	459	433.40 ± 22.12
rand100-1	463	491	475.30 ± 8.10
rand100-2	426	454	441.40 ± 8.44
rand100-3	499	523	519.50 ± 6.54
rand200-1	613	673	647.20 ± 17.78
rand200-2	638	673	664.00 ± 10.35
rand200-3	647	664	653.70 ± 5.52
rand500	989	1027	1006.60 ± 11.69
rand750	1210	1274	1237.60 ± 23.46
rand1000	1293	1325	1306.71 ± 11.61
rand1500	1630	1675	1656.10± 11.64
rand2000	1868	1925	1902.30 ± 17.66
rand5000	2493	2526	2505.25 ± 14.71
rand7500	2710	2763	2731.40 ± 20.96
rand10000	2912	2923	2917.50 ± 7.78
rand25000	25798896	25969934	25888543.30 ± 60230.36
rand50000	240767950	242495321	241518813.10 ± 594617.44

Table 5.13: Side by Side Mean Tour Length for ACS-TSP, HRL ACS with k-Means Clustering, and HRL ACS with G-Means Clustering on TSPLIB TSPs. A "—" in results indicates run time was greater then 7 days.

Name	**ACS-TSP**	**HRL ACS k-Means**	**HRL ACS G-Means**
att48	11370.40 ± 77.90‡	11375.30 ± 96.13	12684.40 ± 129.44
eil51	466.73 ± 7.83‡	460.20 ± 16.03	485.80 ± 21.34
berlin52	7765.10 ± 100.10†‡	8124.20 ± 214.51	8299.40 ± 61.93
eil76	574.90 ± 5.51†‡	586.20 ± 7.08	583.10 ± 9.45
eil101	715.47 ± 7.31†‡	726.30 ± 22.18	772.10 ± 14.43
lin105	15707.30 ± 153.72†‡	17462.00 ± 541.17	20189.70 ± 473.47
bier127	127370.20 ± 1475.56†‡	138128.40 ± 6177.48	145971.60 ± 2757.17
ch150	7129.10 ± 36.15†‡	7550.30 ± 329.26	7698.10 ± 74.77
pr226	89151.25 ± 640.86†‡	99489.40 ± 3124.75	107432.10 ± 3188.78
a280	3161.50 ± 38.63†‡	3292.90 ± 53.97	3278.20 ± 60.35
fl417	14064.62 ± 185.62†‡	16775.50 ± 571.45	18047.40 ± 362.37
att532	34118.00 ± 480.83†‡	35205.90 ± 697.50	34971.90 ± 565.17
rat783	10975.00 ± 106.84	10781.00 ± 81.99†	10881.70 ± 85.65‡
pr1002	352104.50 ± 927.02	323613.20 ± 2528.03†	318020.00 ± 2227.14‡
d1291	62448.00 ± 1645.58†‡	70242.00 ± 1419.25	68338.00 ± 447.15
vm1748	422641.00 ± 4211.16†‡	433516.00 ± 5678.58	449367.00 ± 2114.75
u2152	82465.00 ± 1076.06‡	82136.00 ± 995.27	83949.90 ± 388.47
rl5915	—	777361.00 ± 6491.24†	803460.40 ± 5560.79‡
rl11849	—	1243371.00 ± 619.43†	1393561.00 ± 8976.97‡
usa13509	—	27329690.00 ± 160457.00†	28778014.00 ± 56539.92‡
brd14051	—	618191.60 ± 2638.25†	646013.50 ± 1951.86‡
d18512	—	850101.10 ± 8161.80†	890488.10 ± 2280.85‡

† 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.

Table 5.14: Side by Side Mean Tour Length for ACS-TSP, HRL ACS with k-Means Clustering, and HRL ACS with G-Means Clustering on Random TSPs. A "—" in results indicates run time was greater then 7 days.

Name	**ACS-TSP**	**HRL ACS k-Means**	**HRL ACS G-Means**
rand75-1	381.60 ± 3.17‡	384.30 ± 9.25	437.70 ± 8.83
rand75-2	356.00 ± 5.19‡	358.20 ± 11.39	378.90 ± 8.58
rand75-3	374.10 ± 4.56†‡	388.70 ± 7.56	433.40 ± 22.12
rand100-1	423.10 ± 2.81†‡	456.70 ± 8.64	475.30 ± 8.10
rand100-2	388.60 ± 3.63†‡	435.30 ± 18.92	441.40 ± 8.44
rand100-3	428.70 ± 4.42†‡	461.50 ± 10.91	519.50 ± 6.54
rand200-1	601.50 ± 4.95†‡	620.00 ± 20.42	647.20 ± 17.78
rand200-2	624.50 ± 4.95‡	625.00 ± 22.47	664.00 ± 10.35
rand200-3	620.45 ± 3.71†‡	630.60 ± 13.35	653.70 ± 5.52
rand500	1020.44 ± 16.76	983.80 ± 28.06†	1006.60 ± 11.69‡
rand750	1185.05 ± 20.19‡	1134.50 ± 26.49†	1237.60 ± 23.46
rand1000	1621.55 ± 23.59	1375.90 ± 14.07†	1306.71 ± 11.62‡
rand1500	1798.46 ± 46.45	1665.00 ± 23.21†	1656.10 ± 11.64‡
rand2000	2345.00 ± 58.13	1925.90 ± 21.17†	1902.30 ± 17.66‡
rand5000	—	2695.30 ± 20.09†	2505.25 ± 14.71‡
rand7500	—	2947.60 ± 25.23†	2731.40 ± 20.96‡
rand10000	—	3068.20 ± 27.33†	2917.50 ± 7.78‡
rand25000	—	24103771.10 ± 67368.91†	25888543.30 ± 60230.36‡
rand50000	—	235990494.70 ± 777856.22†	241518813.10 ± 594617.44‡

† 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.

Table 5.15: Side by Side % from Optimal Tour Length for ACS-TSP, HRL ACS with k-Means Clustering, and HRL ACS with G-Means Clustering for best and mean tour lengths. A "—" in results indicates run time was greater then 7 days.

	ACS-TSP		**HRL ACS k-Means**		**HRL ACS G-Means**	
Name	**Best**	**Mean**	**Best**	**Mean**	**Best**	**Mean**
att48	5.69%	6.99%‡	5.54%	7.03%	17.79%	19.35%
eil51	4.23%	9.56%‡	5.16%	8.03%	5.16%	14.04%
berlin52	0.36%	2.96%†‡	3.01%	7.72%	8.64%	10.04%
eil76	5.20%	6.86%†‡	7.06%	8.96%	6.69%	8.38%
eil101	11.13%	13.75%†‡	11.29%	15.47%	17.65%	22.75%
lin105	7.41%	9.24%†‡	16.85%	21.44%	35.32%	40.41%
bier127	4.74%	7.68%†‡	11.55%	16.78%	20.84%	23.41%
ch150	8.41%	9.21%†‡	8.69%	15.66%	16.70%	17.92%
pr226	10.42%	10.93%†‡	16.69%	23.79%	25.25%	33.67%
a280	20.24%	22.59%†‡	24.89%	27.68%	23.58%	27.11%
fl417	17.50%	18.58%†‡	35.39%	41.43%	45.33%	52.16%
att532	22.00%	23.23%†‡	23.10%	27.16%	22.73%	26.32%
rat783	18.64%	24.63%	20.92%	22.43%†	21.44%	23.57%‡
pr1002	35.67%	35.92%	23.67%	24.93%†	22.77%	22.77%‡
d1291	22.94%	22.93%†‡	35.66%	38.27%	33.12%	34.52%
vm1748	21.84%	25.58%†‡	26.43%	28.81%	32.43%	33.52%
u2152	19.65%	28.34%‡	25.19%	27.83%	29.94%	30.66%
rl5915	—		36.76%	37.58%†	41.15%	42.20%‡
rl11849	—		34.98%	35.02%†	50.11%	51.33%‡
usa13509	—		35.76%	37.01%†	43.94%	44.27%‡
brd14051	—		30.86%	31.83%†	37.04%	37.76%‡
d18512	—		30.77%	31.87%†	100.00%	100.00%‡

† 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.

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 improved the time k-means took by a decreased run time of 9.5%.

This benefit was significantly important in the set of unobtainable problems with ACS-TSP. With problems larger than 5,000 cities, a test run was unobtainable as the estimated time to complete an episode of 1000 time steps for the *u2152* was approximately 50 days using the tested implementation. 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. This speedup increased as the problem size increased. For the 18,512 city problem, *d18512*, the time for both clustering algorithms to provide a solution was less than ACS-TSP's run time for the 532 city problem, *att532*.

There is however a balance point where k-means begins to produce solutions faster than G-means. At approximately 6,000 cities, the k-means run time is less than G-means. This is due to the number of clusters and the resulting TSP created from the means. As discussed in the next section, G-means generates more clusters, resulting in a larger TSP used by the recombination algorithm to generate the complete solution. This large TSP slowed the overall G-means algorithm down in relation to k-means. Figure 5.1 shows the three algorithms in relation to size of the TSP and run time. The graph identifies the significant benefit of using the HRL ACS with clustering.

The time decrease is the main benefit of the modified ACS-TSP algorithms. By providing a solution to large complex TSPs in a faster manner, more problems can be efficiently solved. This doesn't change the quality of solutions difference and the benefits may not outweigh the decrease in effectiveness.

5.2.4 Clustering. The final set of data collected is the clustering completed by the two different algorithms. K-means uses a set number of clusters, determined at the start of the algorithm, based on the number of cities divided by a constant. G-

Table 5.16: Average single episode runtime for TSP algorithms on TSPLIB TSPs. A "—" in results indicate that the run time was greater then 7 days.

Name	**ACS-TSP**	**HRL ACS k-Means**	**HRL ACS G-Means**
att48	99.31 ± 0.72	98.11 ± 0.94	16.60 ± 0.47‡
eil51	114.95 ± 6.27	24.24 ± 1.74†	12.99 ± 0.65‡
berlin52	75.41 ± 0.41	51.55 ± 3.19†	35.83 ± 0.96‡
eil76	332.14 ± 1.85	76.92 ± 1.88†	72.01 ± 0.62‡
eil101	993.23 ± 6.08	86.23 ± 10.01†	22.54 ± 1.22‡
lin105	1267.44 ± 449.92	121.08 ± 41.87†	32.86 ± 3.90‡
bier127	2030.45 ± 91.30	339.44 ± 107.07†	34.05 ± 1.58‡
ch150	2871.27 ± 11.16	286.22 ± 12.16†	150.10 ± 0.70‡
pr226	7795.24 ± 14.98	637.52 ± 16.77†	58.42 ± 5.43‡
a280	14377.42 ± 1148.32	657.15 ± 32.59†	174.26 ± 38.19‡
fl417	25998.78 ± 65.46	2180.84 ± 563.57†	374.16 ± 3.47‡
att532	66638.92 ± 197.84	4476.29 ± 122.22†	1960.91 ± 124.69‡
rat783	68457.44 ± 191.86	1676.22 ± 67.26†	693.49 ± 13.47‡
pr1002	70841.01 ± 3024.64	3736.97 ± 798.39†	1005.07 ± 155.21‡
d1291	224610.70 ± 358.53	7023.33 ± 1210.64†	1215.79 ± 7.30‡
vm1748	390912.11 ± 1070.56	5783.06 ± 1044.50†	1668.15 ± 8.23‡
u2152	615482.70 ± 209.53	4752.79 ± 225.81†	2179.71 ± 18.62‡
rl5915	—	16980.36 ± 1248.41†	9463.71 ± 30.35‡
rl11849	—	43874.86 ± 941.59†	19630.67 ± 841.68‡
usa13509	—	7754.47 ± 2991.83†	30344.16 ± 326.98‡
brd14051	—	9643.83 ± 1440.06†	25921.97 ± 1635.61‡
d18512	—	15156.63 ± 1184.45	40437.22 ± 1175.34‡

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

Table 5.17: Average single episode runtime for TSP algorithms on Random TSPs. A "—" in results indicate that the run time was greater then 7 days.

Name	**ACS-TSP**	**HRL ACS k-Means**	**HRL ACS G-Means**
rand75-1	321.05 ± 2.84	73.13 ± 6.09†	21.22 ± 1.21‡
rand75-2	323.02 ± 8.68	74.65 ± 3.79†	21.47 ± 0.61‡
rand75-3	337.01 ± 9.74	82.01 ± 7.61†	18.21 ± 0.69‡
rand100-1	811.94 ± 10.62	80.18 ± 2.69†	27.48 ± 0.71‡
rand100-2	800.05 ± 8.45	85.83 ± 7.49†	24.19 ± 0.75‡
rand100-3	787.94 ± 2.41	78.47 ± 4.54†	25.40 ± 0.84‡
rand200-1	5514.16 ± 5.55	232.93 ± 8.38†	48.28 ± 1.50‡
rand200-2	5519.07 ± 7.04	237.03 ± 12.20†	47.55 ± 1.89‡
rand200-3	5534.84 ± 23.59	234.74 ± 6.90†	46.68 ± 0.60‡
rand500	30704.61 ± 9.34	676.09 ± 11.17†	120.15 ± 0.74‡
rand750	54338.13 ± 5.77	1473.64 ± 18.65†	176.20 ± 1.21‡
rand1000	129595.10 ± 27.53	2386.40 ± 18.96†	345.85 ± 5.78‡
rand1500	317790.31 ± 11.07	3063.61 ± 8.52†	430.42 ± 18.53‡
rand2000	432708.33 ± 35.29	3535.09 ± 8.92†	536.14 ± 14.05‡
rand5000	—	13476.09 ± 16.73†	2039.26 ± 9.19‡
rand7500	—	23435.80 ± 14.26†	53407.52 ± 73.31‡
rand10000	—	28929.07 ± 74.97†	102654.70 ± 46.78‡
rand25000	—	128269.40 ± 316.7693†	937460.60 ± 1868.31‡
rand50000	—	275468.46 ± 739.85†	1130765.40 ± 2972.92‡

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

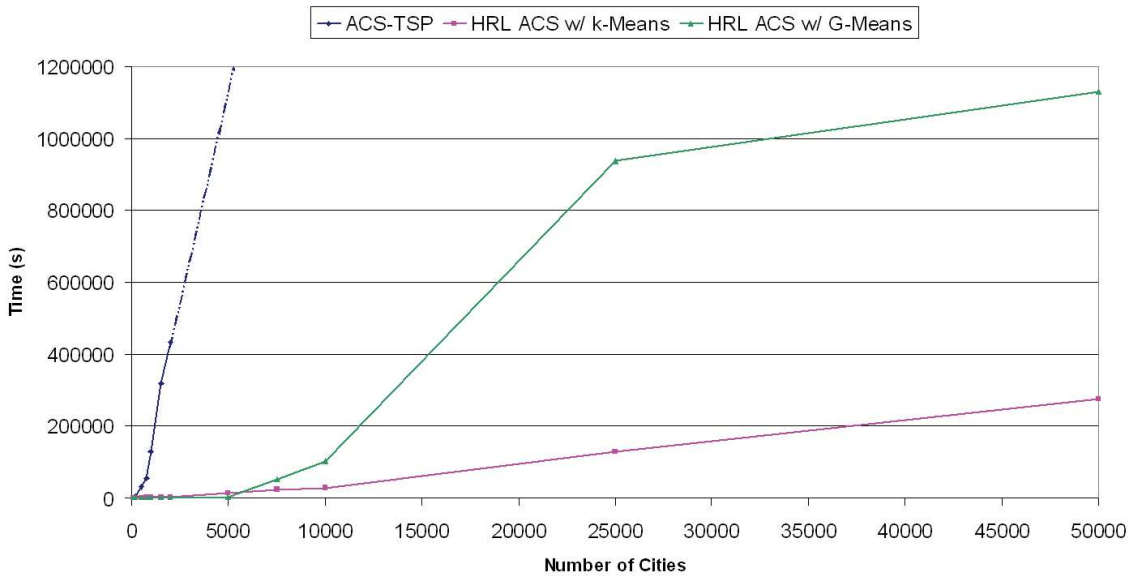


Figure 5.1: A graph of the three TSP algorithms' episode speed in relation to each other for the random TSPs.

means however, uses a statistical test to determine the correct number of clusters for the problem. Tables 5.18-5.21 show the clustering statistics of each algorithm. With the exception of the smallest problem, *att48*, G-means had an average of 111.3% more clusters but 47.8% smaller sized clusters. This difference highlights the issues with partitioning the TSP into small problems and recombining.

As noted above, ACS with k-means produces solutions further from optimal than basic ACS, and G-means produces even further from optimal solutions. The correlation happens to be between the number of clusters and the quality of solutions. As the ratio of cities to clusters approaches infinity, the better the solution found. Since k-means has a higher city to cluster average of 42.58, the solutions are better than G-means, with an average ratio of 22.76.

Figure 5.3 shows an example clustering of the *rand200-3* problem. The figure identifies the individual clusters and their solutions before, during, and after recombination. Notice that the recombination algorithm does not select the best edges between clusters and therefore degrades the solution quality. This identifies the ratio discussed above and Figure 5.4 shows the G-means algorithm result. Comparing

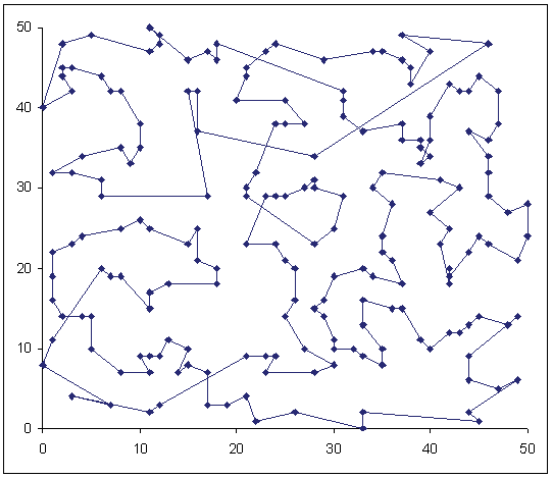


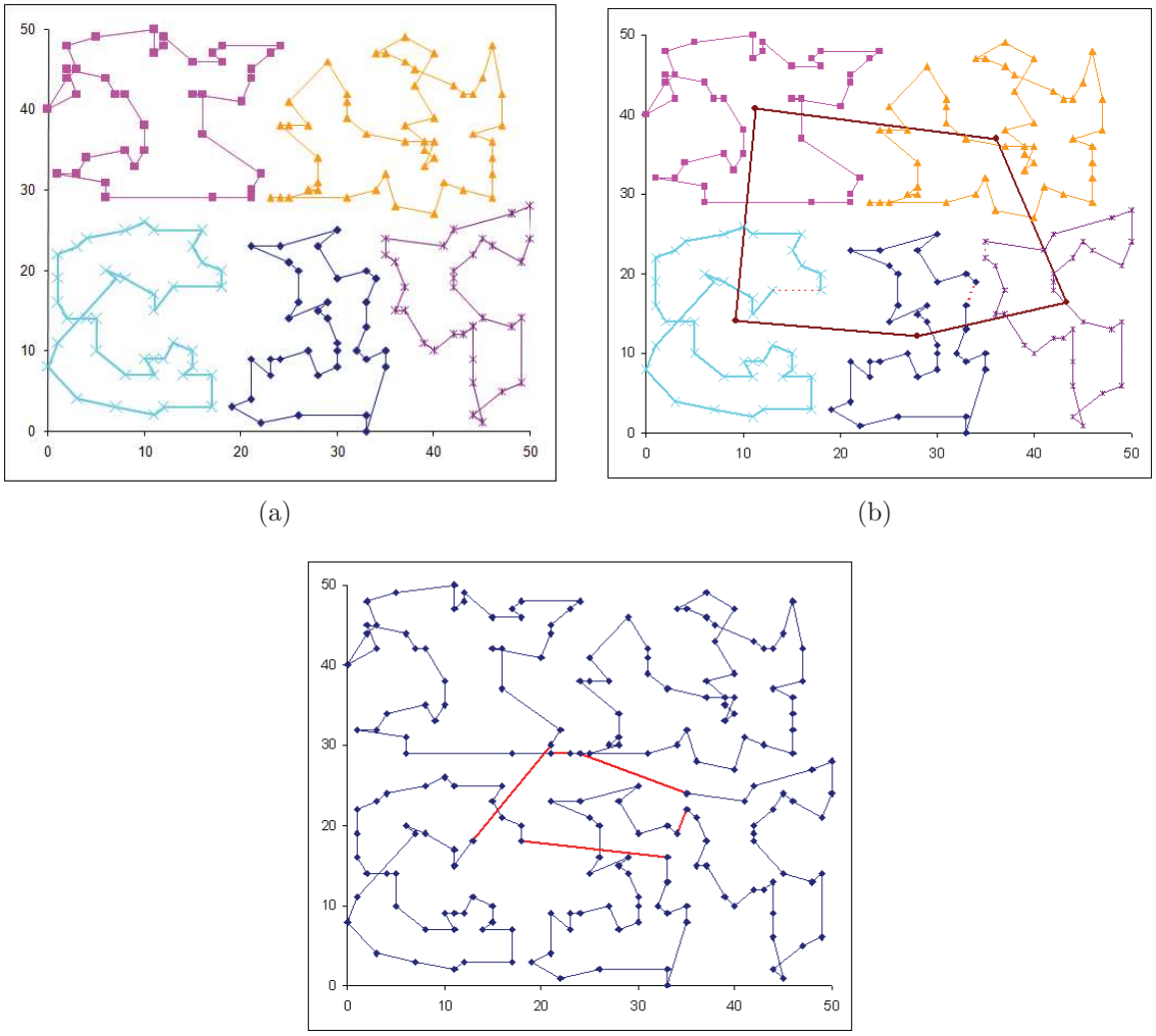
Figure 5.2: Plot of rand200-3 TSP Cities and the best tour found by ACS-TSP.

these two figures to ACS-TSP's solution to the same problem, Figure 5.2, the similarities can be identified as the individual solutions tend to converge to similar paths as the basic ACS-TSP. However, the recombination selects inefficient edges to link the clusters.

5.3 Summary

The implementation of MAXQ-AntQ established the use of ACO learning techniques within a HRL algorithm and problem domain. In fact, the Ant-Q learning adaptation increased the learning rate for the problem by an average of 12%, decreasing the number of training runs needed to converge to a near-optimal solution. This benefit coupled with no significant difference in solution quality, number of primitive actions, provides the framework to show ACO can be used in HRL and the motivation to see if the opposite is true, HRL techniques within the ACO problem domain, TSPs.

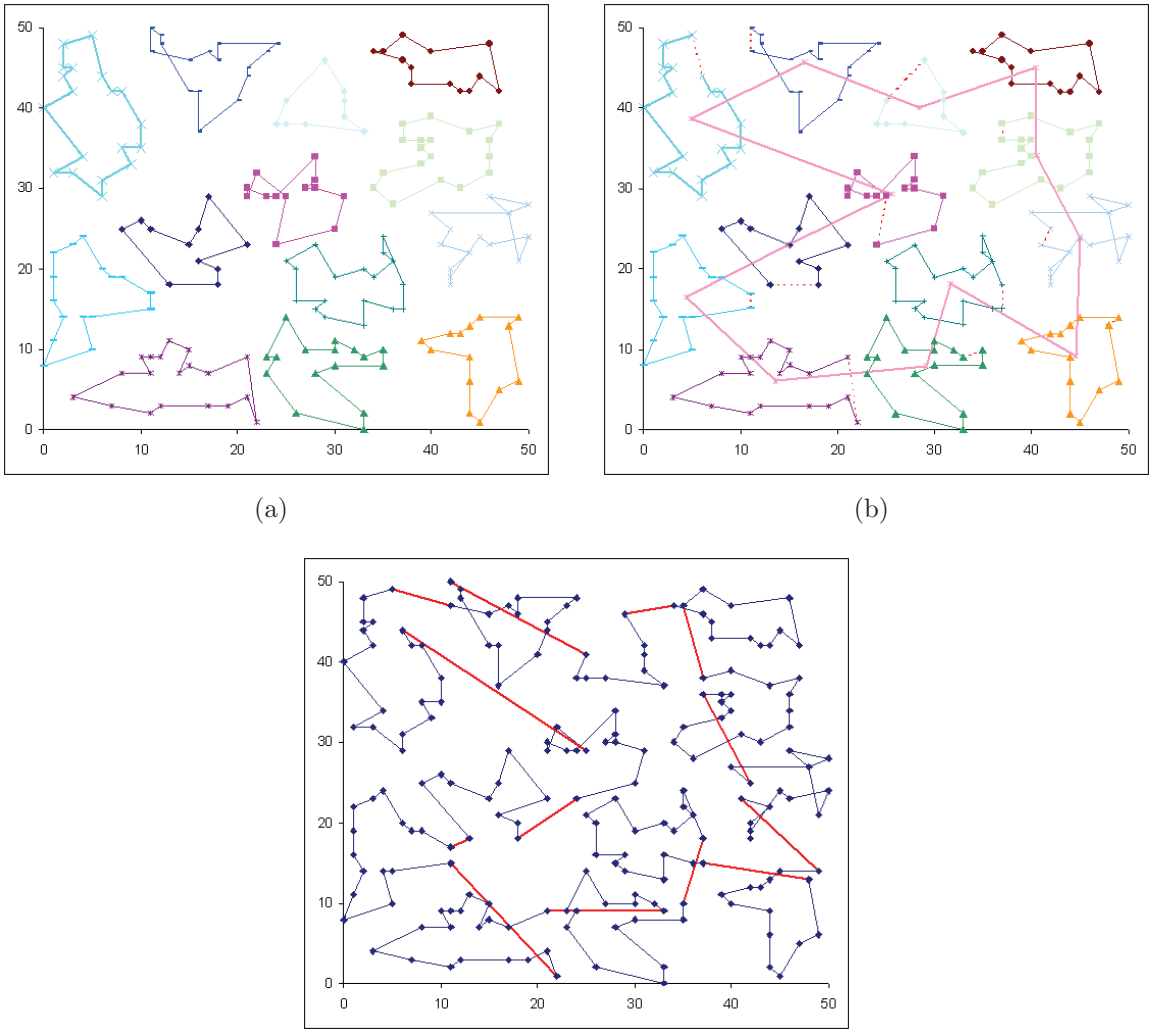
The results of using data clustering algorithms to create the TSP hierarchy, are a mix of benefits and disadvantages from the basic ACS-TSP algorithm. The solution quality is the disadvantage, with k-means producing 5.0% worse solutions and G-means producing 9.5% worse solutions. However, the benefits are the modified clustering algorithm produces the solutions significantly faster and generates a solution



(a) (b)

(c)

Figure 5.3: Plot of rand200-3 TSP Cities and k-Means Clustering Decomposition. (a) Each group of colored points represent a cluster and are solved as individual TSPs. (b) Using the cluster means as a hierarchical layer, the individual clusters are combined dictated by the solution to the means TSP. (c) The resulting TSP solution using the hierarchical components. The red lines indicate the additional edges added to combine the decomposed cluster TSPs.



(a) (b)

(c)

Figure 5.4: Plot of rand200-3 TSP Cities and G-Means Clustering Decomposition. (a) Each group of colored points represent a cluster and are solved as individual TSPs. (b) Using the cluster means as a hierarchical layer, the individual clusters are combined dictated by the solution to the means TSP. (c) The resulting TSP solution using the hierarchical components. The red lines indicate the additional edges added to combine the decomposed cluster TSPs.

Table 5.18: Statistics for k-Means Clustering on TSPLIB TSPs.

Name	**Number of Clusters**	**Largest Cluster Size**	**Smallest Cluster Size**	**Mean Cluster Size**
att48	1	48	48	48.00 ± 0.00
eil51	2	33	18	26.00 ± 3.05
berlin52	2	42	10	26.00 ± 15.42
eil76	2	41	35	38.00 ± 2.00
eil101	3	49	22	33.67 ± 7.50
lin105	3	60	11	35.00 ± 11.37
bier127	3	96	3	42.33 ± 25.72
ch150	3	64	42	50.00 ± 6.92
pr226	5	70	41	56.50 ± 13.75
a280	6	76	37	56.00 ± 8.48
fl417	9	156	3	52.13 ± 35.21
att532	11	141	0	53.85 ± 30.45
rat783	16	78	31	52.20 ± 8.60
pr1002	21	107	0	51.17 ± 25.11
d1291	26	120	0	52.16 ± 36.78
vm1748	35	117	9	51.50 ± 21.24
u2152	44	93	20	49.91 ± 12.73
rl5915	118	93	0	50.10 ± 20.29
rl11849	236	110	0	50.22 ± 20.30
usa13509	271	462	0	49.85 ± 61.24
brd14051	282	236	0	49.83 ± 46.14
d18512	371	228	0	49.90 ± 35.37

Table 5.19: Statistics for k-Means Clustering on Random TSPs.

Name	**Number of Clusters**	**Largest Cluster Size**	**Smallest Cluster Size**	**Mean Cluster Size**
rand75-1	2	48	27	37.50 ± 4.19
rand75-2	2	45	30	37.50 ± 4.45
rand75-3	2	48	27	37.50 ± 8.76
rand100-1	3	42	26	33.33 ± 5.85
rand100-2	3	47	19	33.33 ± 7.91
rand100-3	3	45	25	33.33 ± 5.47
rand200-1	5	53	24	40.00 ± 7.26
rand200-2	5	52	23	40.00 ± 8.08
rand200-3	5	54	26	40.00 ± 7.67
rand500	11	84	14	45.46 ± 12.12
rand750	16	74	25	46.95 ± 9.89
rand1000	21	78	13	47.62 ± 12.63
rand1500	31	82	20	48.39 ± 11.90
rand2000	41	96	30	48.78 ± 11.53
rand5000	101	100	13	49.51 ± 14.00
rand7500	151	119	12	49.67 ± 14.55
rand10000	201	128	9	49.75 ± 15.49
rand25000	501	133	0	49.90 ± 24.23
rand50000	1000	150	0	50.00 ± 34.38

Table 5.20: Statistics for G-Means Clustering on TSPLIB TSPs.

Name	**Mean Number of Clusters**	**Largest Cluster Size**	**Smallest Cluster Size**	**Mean Cluster Size**
att48	3.00 ± 0.00	22	13	16.00 ± 4.32
eil51	3.00 ± 0.00	21	15	17.00 ± 1.60
berlin52	2.00 ± 0.00	40	12	26.00 ± 14.36
eil76	2.00 ± 0.00	39	37	38.00 ± 1.03
eil101	6.90 ± 0.32	19	10	14.64 ± 2.78
lin105	8.30 ± 0.95	32	4	12.65 ± 6.66
bier127	11.90 ± 0.32	23	4	10.67 ± 5.44
ch150	4.00 ± 0.00	43	31	37.50 ± 4.26
pr226	14.90 ± 0.32	22	7	15.17 ± 5.47
a280	9.70 ± 0.68	50	19	28.87 ± 5.71
fl417	14.90 ± 0.32	50	4	27.99 ± 13.45
att532	17.00 ± 0.00	50	11	31.29 ± 11.11
rat783	22.00 ± 0.00	47	28	35.59 ± 4.23
pr1002	29.90 ± 0.32	49	6	33.51 ± 9.92
d1291	37.00 ± 0.00	48	25	34.89 ± 5.75
vm1748	52.00 ± 0.00	48	15	33.62 ± 7.33
u2152	62.00 ± 0.00	48	19	34.71 ± 6.99
rl5915	169.00 ± 0.00	50	18	35.02 ± 6.34
rl11849	729.00 ± 0.32	24	8	16.26 ± 3.24
usa13509	874.80 ± 2.68	32	4	15.44 ± 4.18
brd14051	823.40 ± 1.58	32	7	17.07 ± 3.26
d18512	1077.70 ± 0.95	26	3	17.18 ± 3.39

Table 5.21: Statistics for G-Means Clustering on Random TSPs.

Name	**Mean Number of Clusters**	**Largest Cluster Size**	**Smallest Cluster Size**	**Mean Cluster Size**
rand75-1	4.00 ± 0.00	21	17	18.75 ± 0.93
rand75-2	4.00 ± 0.00	23	15	18.75 ± 2.35
rand75-3	5.90 ± 0.32	19	8	12.71 ± 3.10
rand100-1	6.00 ± 0.00	24	9	16.67 ± 5.75
rand100-2	6.00 ± 0.00	22	11	16.67 ± 3.55
rand100-3	6.00 ± 0.00	24	9	16.67 ± 4.43
rand200-1	13.00 ± 0.00	24	8	15.39 ± 3.61
rand200-2	12.90 ± 0.32	23	8	15.50 ± 3.30
rand200-3	13.00 ± 0.00	20	9	15.39 ± 3.06
rand500	29.00 ± 0.00	24	12	17.24 ± 2.79
rand750	45.90 ± 0.32	24	8	16.34 ± 3.31
rand1000	59.86 ± 0.38	24	9	16.71 ± 3.61
rand1500	88.00 ± 0.00	24	10	17.05 ± 3.14
rand2000	124.90 ± 0.32	23	5	16.01 ± 3.17
rand5000	300.00 ± 0.00	24	7	16.69 ± 3.32
rand7500	466.20 ± 1.30	24	7	16.08 ± 3.45
rand10000	587.50 ± 0.71	44	16	29.63 ± 3.85
rand25000	1539.00 ± 0.00	24	4	16.46 ± 3.35
rand50000	1517.00 ± 0.00	50	4	32.96 ± 6.09

to larger problems in a feasible amount of time. HRL ACS with k-means clustering produces solutions an average of 20.88 times faster and HRL ACS with G-means clustering 20.94 times. The HRL ACS with clustering algorithm could also generate solutions to all problems including those considered infeasible by the ACS-TSP algorithm.

The differences between k-means and G-means is in the number of clusters created in the hierarchy. G-means averaged 111.3% more clusters than k-means. This produced mixed results; the solution quality was 4.5% worse than k-means, identifying the lower cities per cluster ratio plays a role in the final result. However, G-means would produce the results in approximately 10% faster run time. Unfortunately, this run time benefit of G-means has diminishing returns. As the problem size increased to the much larger problems, more than 15,000 cities, G-means began taking longer than k-means to produce worse solutions. This came from the large number of clusters creating a large TSP to solve for the recombination method.

The tradeoff between solution effectiveness and efficiency is needed to be made for the application of the modified ACS with clustering algorithms to be used. Both clustering algorithms produce worse results, but in a much faster and efficient manner. This benefit was evident in very large problems as the basic ACS-TSP solution was unobtainable with a limited amount of time. The results show k-means can be effective and efficient if the domain allows semi-optimal solutions, within approximately 25% of the known optimal.

VI. Conclusions

This research demonstrates the feasibility of combining two artificial intelligence domains and the effectiveness of a set of the resultant algorithm. This chapter reiterates the foundations used to validate the research goals and how the modified algorithms were designed. The test results from Chapter V are summarized in Section 6.2, and are followed by potential areas for future work.

6.1 Summary

There is a need to produce effective and efficient algorithms able to solve large, complex problems. These large problems can be considered models of the real world, where the size of the state space is larger than a simple deterministic or stochastic algorithm can handle [18].

The goal of reinforcement learning is to produce a policy an agent can follow to a near-optimal solution given a problem [20]. Most algorithms complete this task by combining function approximations with temporal difference methods [8]. In addition, the Markov assumption allows the algorithm to reduce the stored state history to only the previous state. This assumption allows the algorithm to lower the model requirements. However, reinforcement learning methods are not practical for real world problems as the problem may be multi-goal or just too complex for the algorithm to solve.

Hierarchical reinforcement learning methods attempt to model these complex problems by using one of two assumptions. First, the ability for an algorithm to reuse solutions or procedures without changing the model or state space within the module. This method is used by Parr's hierarchy of abstract machines [14] and Sutton's *option* algorithm [21]. Dietterich's MAXQ uses the second method, the ability to create a hierarchy of state abstraction and learn the current subtask's policy as a foundation for the higher levels to use as their policy [4]. MAXQ-Q uses two types of reinforcement learning to update the policy. The first, Q-learning, relies on a value function approximation in line with the MDP assumption [22]. This allows

the value function to be based upon the state-action pair and updated using only the current value and the observed reward [22]. This value function and resultant policy have been proven to converge to the optimal policy with probability of on if Equation 2.8. The second learning, SARSA, extends Q-learning and adds a second update function. This update function is dependent on the action to be be selected in the next state. This creates a quintuplet used to update the current state-action value. SARSA has also been proven to converge to the optimal policy if a GLIE policy is used for action selection. MAXQ takes the reinforcement learning in both these algorithms and creates a hierarchical reinforcement learning algorithm. The hierarchy is imposed on by the designer onto the problem. This means the programmer must know the problem decomposition beforehand, which could result in a bad hierarchy decomposition [4]. This is the foundation this research was built from.

The first stated goal of this research was to demonstrate that ACO learning methods could scale up using HRL techiniques. To do this, a modified MAXQ algorithm was created to include Ant-Q learning. Ant-Q is an ACO adaptation of Q-learning and was easily inserted into the MAXQ algorithm. By modifying four areas, Ant-Q became the learning mechanism within the MAXQ HRL. First, the value function at primitive action nodes was changed to update the pheromone values of the state-action pair in accordance with Ant-Q. Second, each hierarchical level incorporates the colony concept of ACO and runs through a constant number of ants performing the action selection. The third change alters the action selection from an ε-greedy selection to the ACO probabilistic calculation. Finally, the composite value function at non-primitive nodes is replaced with the Ant-Q value function containing the pheromone for each hierarchical level.

The second goal of the research was to demonstrate HRL techniques could be used in the ACO domain. This was done with by using the task hierarchy from HRL on the TSP. By adapting the concept to the TSP using a graphical data clustering method it allows for the creation of the task hierarchy. This not only provided the desired breakdown but also did so dynamically, extending Dietterich's future direction.

Using the base ACS as a foundation, the clustering provide a means to break the larger problems into smaller ones and reduced the problem complexity and requirements. The last part is to determine how to recombine the small problems to provide an overall solution to the initial TSP. This was done using a simple greedy edge selection based on the TSP solution constructed from the cluster means.

6.2 Results

The results of testing the created algorithms, MAXQ-AntQ and HRL ACS with clustering, against the baseline algorithms, MAXQ-Q and ACS-TSP, demonstrated the combination of the two domains, HRL and ACO, was not only feasible but beneficial. The first research goal was accomplished by the adapted MAXQ-AntQ algorithm. It followed 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 was no significant difference in quality of solution, demonstrating Ant-Q learning inserted into MAXQ provided a benefit to the algorithm.

The second goal was accomplished by testing the 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 faster. This proved to be especially significant with large scale problems as the modified algorithms could obtain a solution while the ACS could not. The final measurement compared the two clustering algorithms' ability to decompose the hierarchy. G-means tended to have 111.3% more clusters than k-means. This allowed G-means to increase speed over k-means by and additional 9.5%, due to the average cluster size being 47.8% smaller. However, tests revealed there was a breaking point for G-means where the smaller cluster size would not benefit the overall algorithm. Around 12,000 cities, G-means required more time to complete an episode due to the combination TSP size. In addition, the solution

quality for G-means did not outperform k-means for any problem and therefore, k-means demonstrated the best application of the research goal by providing an 20.88 times speed up with only a 5.0% average loss in solution quality over ACS-TSP.

6.3 Future Extensions

There are several areas to continue the research into combining HRL and ACO. The first is the selection of the HRL algorithm. MAXQ appeared to be well suited versus the other HRL methods discussed in Chapter II as it provided a simple framework to break the problem into subproblems. A selection of another HRL algorithm could provide a similar, if not better, result and demonstrate the same properties. In addition, there has been recent work into the development of an ACS-gridworld algorithm. This algorithm uses the concepts of area of expertise to produce policies in a grid world environment [12].

The next area to expand is the design of the clustering algorithms. There are three areas to look at: 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 [15], or even additional modifications to the two selected. For instance, add a minimum cluster size to avoid the issue with G-means not performing well on large problem sizes. The results showed where the algorithm is lacking is in the recombination edge selection. Using only a greedy search results in inefficient overall solutions being produced. An effort could focus on the edge selection between clusters, including finding the best connection taking into consideration the final city to the next cluster. Or use the area of expertise discussed above with ACS-gridworld [12], instead of using the actual individual solutions, use the pheromone matrix created by the subproblems as the baseline for the larger problem; this should provide a quickly converging algorithm and may require only running the overall ACS-TSP algorithm a fraction of the time steps than before. The finally area is the ACS-TSP algorithm. There are several documented improvements to this baseline

algorithm including 2/3-opt [6] and Max-Min Ant System [19]. Those adaptations could only help with the solutions found using this research's modified algorithms. By increasing the quality of the foundation algorithm used, it will benefit all parts of the TSP hierarchy.

Appendix A. Taxi World Problems

This appendix contains the graphical representation for all taxi world problems. In addition it lists the taxi starting location, pickup, destination goals with the primitive action count in the optimal policy for the tested problem.

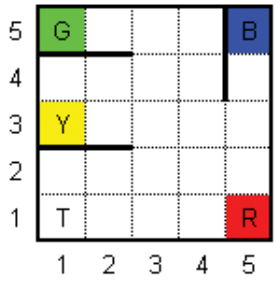


Figure A.1: The test 5x5 taxi world problem. Taxi starts in location (1,1), pickup is RED, and destination is GREEN. Optimal path is 14 primitive actions (4 move to RED, 1 PICKUP, 8 move to GREEN, 1 PUTDOWN)

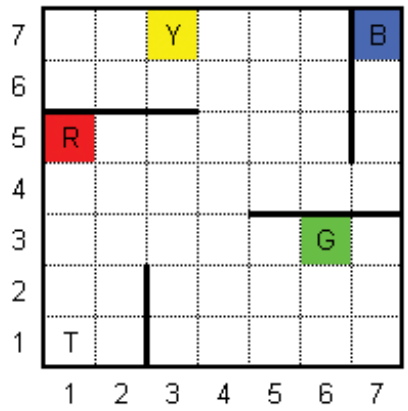


Figure A.2: The test 7x7 taxi world problem. Taxi starts in location (1,1), pickup is BLUE, and destination is YELLOW. Optimal path is 24 primitive actions (12 move to BLUE, 1 PICKUP, 10 move to YELLOW, 1 PUTDOWN)

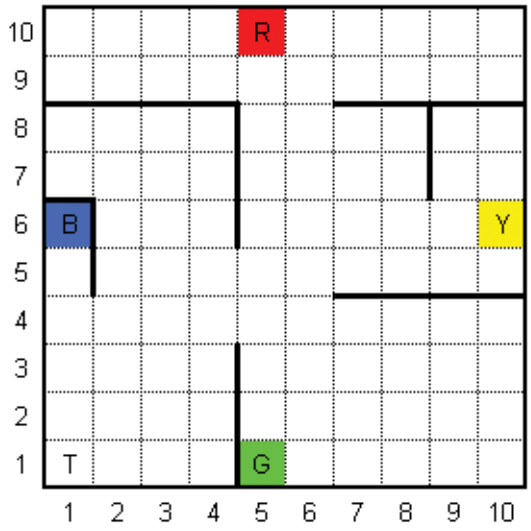


Figure A.3: The test 10x10 taxi world problem. Taxi starts in location (1,1), pickup is GREEN, and destination is BLUE. Optimal path is 21 primitive actions (10 move to GREEN, 1 PICKUP, 9 move to BLUE, 1 PUTDOWN)

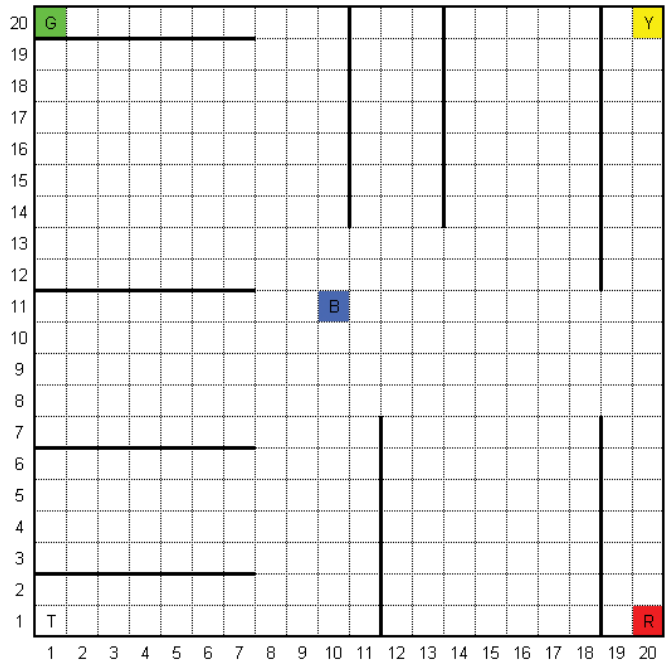


Figure A.4: The test 20x20 taxi world problem. Taxi starts in location (1,1), pickup is YELLOW, and destination is RED. Optimal path is 59 primitive actions (38 move to YELLOW, 1 PICKUP, 19 move to RED, 1 PUTDOWN)

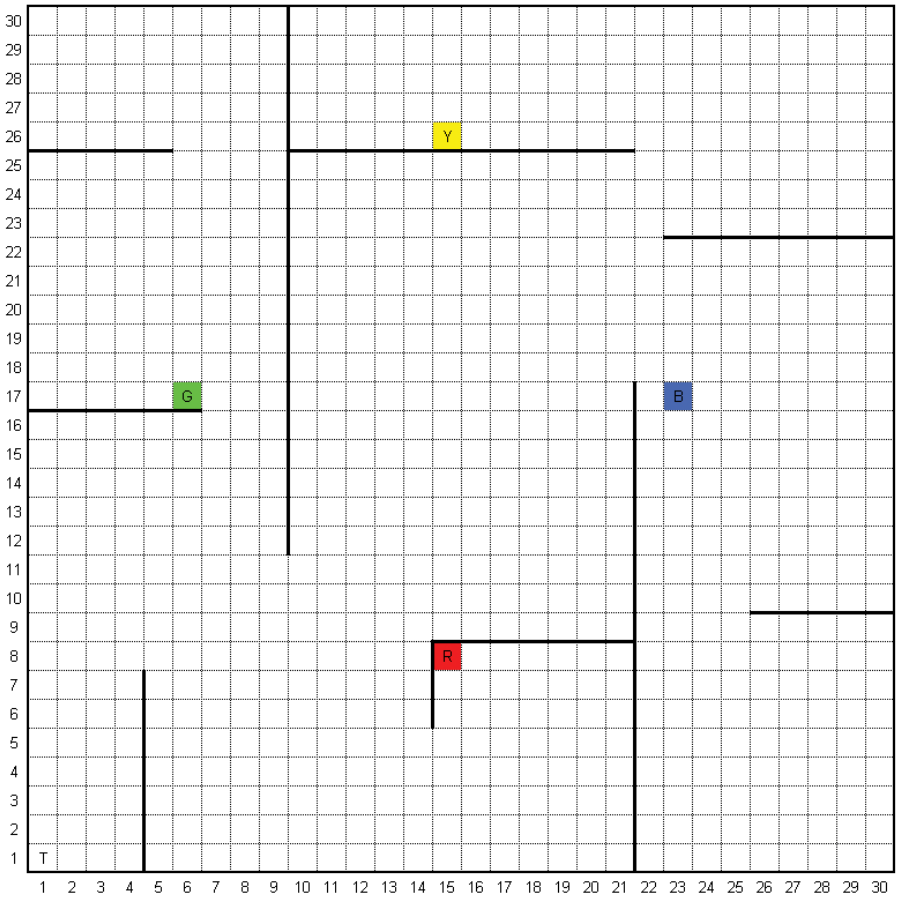


Figure A.5: The test 30x30 taxi world problem. Taxi starts in location (1,1), pickup is RED, and destination is BLUE. Optimal path is 56 primitive actions (27 move to RED, 1 PICKUP, 27 move to BLUE, 1 PUTDOWN)

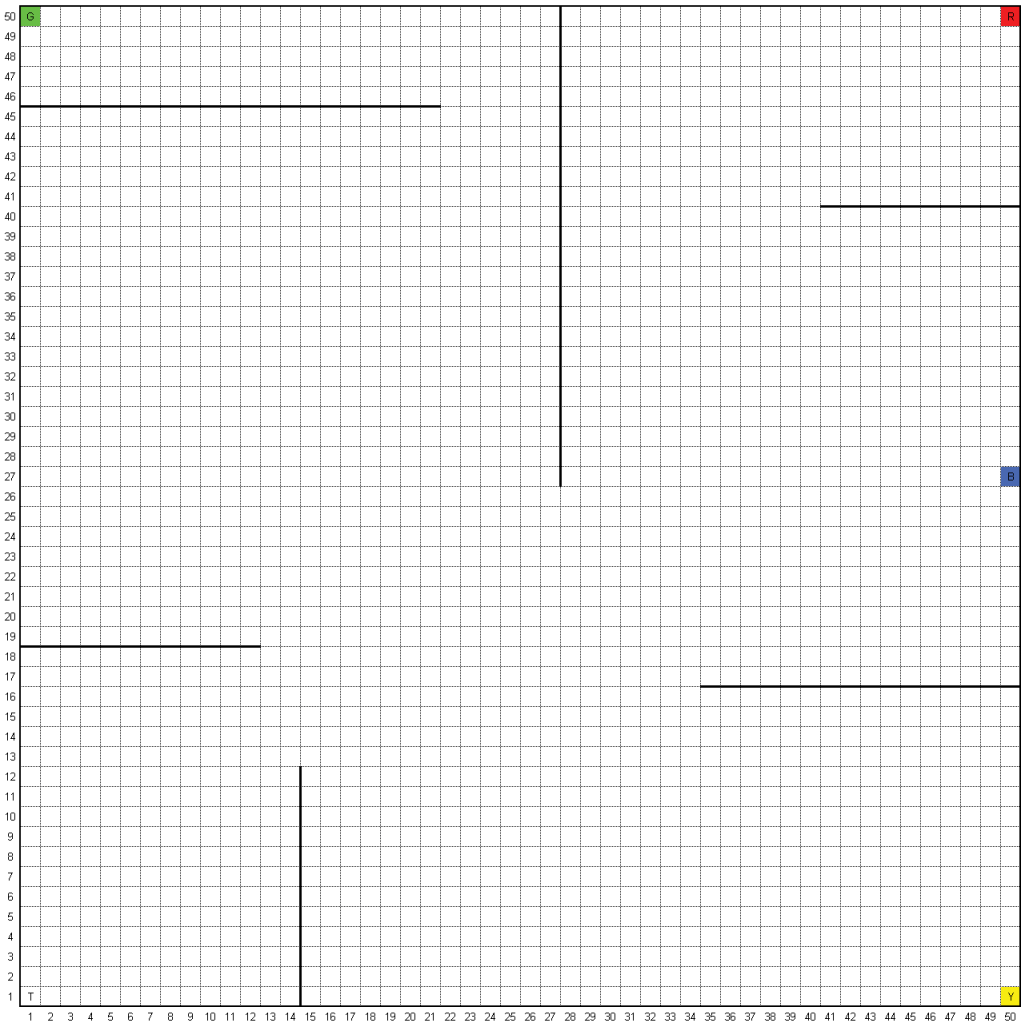


Figure A.6: The test 50x50 taxi world problem. Taxi starts in location (1,1), pickup is GREEN, and destination is YELLOW. Optimal path is 215 primitive actions (115 move to GREEN, 1 PICKUP, 98 move to YELLOW, 1 PUTDOWN)

Bibliography

1. Applegate, David L., Robert E. Bixby, Vasek Chvatal, and William Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 41 William Street, Princeton, New Jersey, USA, 08540-5237, 2007.

2. Barto, Andrew G. and Sridhar Mahadevan. "Recent Advances in Hierarchical Reinforcement Learning". *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003. ISSN 0924-6703.

3. Bonabeau, Eric, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Inc., 198 Madison Avenue, New York, New York 10016, 1999.

4. Dietterich, Thomas G. "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition". *Journal of Artificial Intelligence*, (13):227–303, November 2000.

5. Dorigo, Marco, Mauro Birattari, and Thomas Stutzle. "Ant Colony Optimization". *Computational Intellligence Magazine, IEEE*, 1(4):28–39, 2006. ISSN 1556-603X.

6. Dorigo, Marco and Luca Maria Gambardella. "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem". *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.

7. Gambardella, Luca Maria and Marco Dorigo. "Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem". *International Conference on Machine Learning*, 252–260. 1995.

8. Gordon, Geoffrey J. "Stable function approximation in dynamic programming". Armand Prieditis and Stuart Russell (editors), *Proceedings of the Twelfth International Conference on Machine Learning*, 261–268. Morgan Kaufmann, San Francisco, CA, 1995. URL `citeseer.ist.psu.edu/gordon95stable.html`.

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

10. Jaakkola, Tommi, Michael I. Jordan, and Satinder P. Singh. *On the Convergence of Stochastic Iterative Dynamic Programming Algorithms*. Technical report, Cambridge, MA, USA, 1993.

11. Jianyong, L. and Z. Xiaobo. "On Average Reward Semi-Markov Decision Processes with a General Multichain Structure". *Math. Oper. Res.*, 29(2):339–352, 2004. ISSN 0364-765X.

12. Kaelbling, Leslie Pack, Michael L. Littman, and Andrew P. Moore. "Reinforcement Learning: A Survey". *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

13. MacQueen, J. B. "Some Methods for classification and analysis of multivariate observations". *Proceedings of the Fifth Symposium on Mathematical Statistics and Probabilities*, 281–297. 1967.

14. Parr, Ronald and Stuart Russell. "Reinforcement Learning with Hierarchies of Machines", January 17 1997.

15. Pelleg, D. and A. Moore. "X-Means: Extending K-Means with Efficient Estimation of the Number of Clusters". *ICML 2000*, 2000.

16. Reinelt, Gerhard. "TSPLIB 95", 2007. URL `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.

17. Rummery, G. A. and M. Niranjan. *Online Q-learning using connectionist systems*. Cued/finfeng/tr 166, Cambridge University Engineering Department, Cambridge, England, 1994.

18. Smart, William D. "Explicit Manifold Representations for Value-Functions in Reinforcement Learning". *Proceedings of the Eighth International Symposium on Artificial Intelligence and Mathematics*. January 2004. Paper number AI&M 25-2004.

19. Stutzle, T. and H. Hoos. "The Max-Min Ant System and Local Search for Combinatorial Optimization Problems", 1999.

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

21. Sutton, Richard S., Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning". *Artif. Intell.*, 112(1-2):181–211, 1999. ISSN 0004-3702.

22. Watkins, Christopher. *Learning From Delayed Rewards*. Ph.D. thesis, King's College, Oxford, May 1989.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
13-09-2007	Masters Thesis	Sept 2006 – Sept 2007

TITLE AND SUBTITLE

SCALING ANT COLONY OPTIMIZATION WITH HIERARCHICAL REINFORCEMENT LEARNING PARTITIONING

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

AUTHOR(S)

Dries, Erik J, Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/07-16

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research merges the hierarchical reinforcement learning (HRL) domain and the ant colony optimization (ACO) domain. The merger produces a HRL ACO algorithm capable of generating solutions for both domains. This research also provides two specific implementations of the new algorithm: the first a modification to Dietterich's MAXQ-Q HRL algorithm, the second a hierarchical ACO 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 and SARSA, 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 research 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 research tests two clustering algorithms, k-means and G-means. The results demonstrate the algorithm with data clustering produces solutions 85-95% faster but with 5-10% decrease in solution quality.

15. SUBJECT TERMS

Ant Colony Optimization, Hierarchical Reinforcement Learning, Metaheuristic Optimization

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Gilbert L. Peterson (ENG)
U	U	U	UU	114	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565, ext 4281 (gilbert.peterson@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18