EVOLUTIONARY ALGORITHMS FOR SOLVING DEC-POMDP PROBLEMS

by

Barış Eker

B.S. Computer Engineering, Bilkent University, 1999

M.S. Computer Engineering, Bilkent University, 2002

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2012

ÖZET

MO-KGMKS PROBLEMLERİNİN ÇÖZÜMÜNDE EVRİMSEL ALGORİTMALAR

Merkezi Olmayan Kısmen Gözlemlenebilir Markov Karar Süreci (MO-KGMKS) modeli, kısmen gözlemlenebilir ortamlarda çoklu etmen planlama problemini adreslemektedir. Modelin NEXP-complete kompleksliğe sahip olmasından dolayı, sadece küçük boyutlu problemler optimal bir şekilde çözülebilir. Bu sebepten dolayı, birçok araştırmacı daha büyük boyutlu problemler için en iyiye yakın çözüm üretebilecek yaklaşık çözüm yordamları üzerine yoğunlaşmıştır. Bununla birlikte, şimdiye kadar geliştirilen yaklaşık çözüm yordamları bile büyük boyutlu problemleri ancak az sayıda adım için çözebilmektedir. Bunun bir nedeni etmen stratejilerini temsil ederken ve strateji uzayını tararkenki üssel hafıza gereksinimidir. Bu tezde sonlu adımlı MO-KGMKS problemlerini yaklaşık olarak çözmek için dört yeni yaklaşım sunuyoruz. İlk yaklaşım, MAP, MO-KGMKS problemlerini KGMKS problemi olarak modelleme ve daha sonrasında verimli bir KGMKS çözümcüsü ile çözme temellidir. Diğer yaklaşımların hepsi, yani ES-BV, ES-OH ve GA-FSC, evrimsel yordamların uygulanması temeline dayanmaktadır. ES-BV, MAP yönteminde olduğu gibi inanç vektörlerini kullanır ve strateji vektörlerini evrimsel stratejileri (ES) kullanarak bulmaya çalışır. ES-OH yaklaşımı gözlem tarihçesini kullanmayı, karar vermek için bunu bir sinir ağına girdi yapmayı ve sinir ağlarını eğitmek için ES kullanmayı önerir. GA-FSC yordamı ise stratejileri temsil etmek için sonlu durum kontrolcülerini kullanır ve genetik yordamları kullanarak en iyi stratejiyi arar. Bütün yordamlar en bilinen MO-KGMKS problemlerinde test edilmiştir. Sonuçlarımızı bu konudaki en ileri tekniklerle ve birbirleriyle karşılaştırdık. MAP haricinde bu çalışmadan geliştirilen yordamların varolan en iyi yordamlarla karşılaştırılabilir bir performansa sahip olduğunu ve GA-FSC kullanılması durumunda problemler için çözüm adım sayısının artırıldığını gösterdik.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

a	Action
A	The set of joint actions in a DEC-POMDP model
b	Belief
o	Observation
R	The immediate reward function in a DEC-POMDP model
S	Finite set of states in a DEC-POMDP model
T	The state transition function in a DEC-POMDP model
Ω	The set of joint observations in a DEC-POMDP model
Obs	The observation function in a DEC-POMDP model

LIST OF ACRONYMS/ABBREVIATIONS

DEC-POMDP	Decentralized Partially Observable Markov Decision Process
ES	Evolution Strategies
ES-BV	Evolution Strategies and Belief Vector
ES-OH	Evolution Strategies and Observation History
FSC	Finite State Controller
GA	Genetic Algorithms
GA-FSC	Genetic Algorithms and Finite State Controllers
MAP	Modeling as POMDP
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process

1. INTRODUCTION

Autonomous agents perceive their environment with their sensors and make decisions based on their perception to accomplish their tasks. The distinct situations in the environment are formally represented as *states*. The state is an abstraction; it should be defined properly to represent only the relevant information obtained, and it may also contain historical data. Given the state, the agent should be able to decide which action to perform. For this, it has to have a *policy* which is a mapping from the possible states to the best actions for these states. The *state space*, which is the set of all possible states, can be finite, countable or uncountably infinite depending on the environment. The cardinality of the state set has an impact on the difficulty of the problem.

Depending on the environment properties, several frameworks have been used to model such decision making problems. The Markov Decision Process (MDP), one of such frameworks, models the environments that can be fully observed by the agents, however, the outcomes of the agent actions are non-deterministic, meaning that an action taken in a state may not result in the same successor state for all cases. This model has been applied for single agent planning problems in many environments successfully [2]. However, in most real world problems, the environment is not fully observable and additionally, the sensor readings may include some noise. Such environments are called *partially observable* and the Partially Observable Markov Decision Process (POMDP) model have been proposed for such cases. This model have been used to solve many real-world problems like planning and navigation [3, 4].

There is an increasing interest in the multiagent planning in the last decade since there are cases that require a team of agents to act cooperatively in a partially observable environment. In such cases, each agent interacts with only a part of the environment and has only a partial view of the environment via local observations. Since there are other agents in the environment, each agent should also reason about how the other agents can behave and how that can affect the environment. These issues make planning in such

environments a challenging problem. The Decentralized POMDP (DEC-POMDP) is a model proposed for such environments when the agents do not communicate with each other [5, 6]. The Cooperative Box Pushing is an example problem for such environments. In this problem, there are two agents that aim to move some boxes to target places. There are two small boxes each can be moved by a single agent and one large box that requires the cooperation of the agents to be moved. Also, moving the large box has more reward than moving two small boxes, therefore it is better for the agents to cooperate whenever possible. Besides such experimental problems, many multiagent simulation or real world simulation environment have similar properties. As an example, Teambots is a simulation environment for testing multiagent planning algorithms in several domains, such as soccer [7]. The agents act autonomously in this environment in order for their team to win. The behaviour of the agents can be modelled with the DEC-POMDP and known DEC-POMDP approaches can be applied for this situation.

Optimally solving finite-horizon POMDPs has been shown to be PSPACE-complete; therefore, it is hard to solve POMDP problems exactly except for some small cases. The situation is worse for DEC-POMDP problems, because, solving them is NEXP-complete [6]. Nevertheless, there have been many studies that aim to solve DEC-POMDP problems exactly [8, 9, 10, 11, 12, 13, 14, 15, 16]. On the other hand, due to the complexity of the problem, many researchers concentrate on finding efficient approximate algorithms that can handle larger state spaces and larger horizons [17, 18, 19, 20, 21, 22, 23, 1, 5, 24, 25, 26, 27, 28]. There are also some studies that define some subsets of the DEC-POMDP model that are still expressive but easier to solve [29, 30, 31, 32, 33]. All of these studies try to find a policy offline and assume that the agents will use these pre-calculated policies during task execution. Recently, some studies that make the agents act while planning online without a pre-calculated policy have also appeared [34, 35].

The aim of a DEC-POMDP solver is to find a policy that will maximize the expected team reward, therefore this can be regarded as solving an optimization problem. Consequently, algorithms used for solving nonlinear unconstrained optimization problems such as hill climbing, simulated annealing and evolutionary algorithms are viable candidates

also for solving DEC-POMDP problems. Evolutionary algorithms have been used to solve learning and optimization problems in several disciplines [36, 37]. Genetic algorithms, which are a member of the evolutionary algorithms family, have been used as a component of some POMDP algorithms for finding the optimal values of parameters specific to the solution technique [38]. Genetic algorithms were used in one study [1] for solving DEC-POMDP problems. However, due to the exponential memory requirements caused by chromosome encoding, this approach is only able to handle small problems with short horizons.

In this study, we offer four novel approximate solution methods to solve finite horizon DEC-POMDP problems. The first method (MAP) is based on converting the DEC-POMDP problem into a POMDP problem and solving it using a POMDP solver. Each agent holds a belief vector and uses it together with the policy vectors obtained from the POMDP solver. The second method (ES-BV) also offers to use belief vectors, but it uses evolution strategies to search for policy vectors. These two methods require the agents to update their beliefs at each time step, however, making the belief update exactly is not possible in DEC-POMDP problems. For this reason, we offer an approximate belief update. Due to the problems in some cases with this approach, we propose another method (ES-OH) that uses the observation history instead of belief vectors and make the decision with the help of neural networks that take the observation history as input. In the final approach, (GA-FSC) we represent the agent policies with finite state controllers (FSC) that are used in some infinite horizon DEC-POMDP studies [19, 8] and search for the optimal FSCs using genetic algorithms. We show that all the algorithms developed in this study, except *MAP*, have comparable performance to that of the existing top algorithms and in the case of the *GA-FSC*, the solution horizon for the problems are extended at least an order of magnitude.

The organization of the rest of the thesis is as follows. In Chapter 2, we describe the formal DEC-POMDP model and how the agent strategies are represented. We also review the prior exact and approximate DEC-PODMP solution techniques in the literature. In Chapter 3, we describe the Modeling as POMDP method and evaluate its performance on

some well-known DEC-POMDP problems. In Chapter 4, we introduce two novel methods, *ES-BV* and *ES-OH*, both based on evolution strategies. We discuss why *ES-BV* performs better than *MAP* and why the *ES-OH* approach is needed after *ES-BV*. In Chapter 5, we present the *GA-FSC* approach and the results of our experiments with the exact and approximate fitness calculation. In Chapter 6, we compare our algorithms both in terms of expected reward performance and run-time performance. Finally, in Chapter 7, we conclude with a summary of our work and discuss our results.

2. DECENTRALIZED PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES

The Markov Decision Process (MDP) approach can be used for modeling the environments when the agents can sense the state exactly and the transitions between the states are non-deterministic. If the environment is partially observable, the agents are expected to acquire information about the state by getting observations and they maintain a belief about the current state. The Partially Observable MDP (POMDP) model is more suitable for such environments. In the POMDP model, the aim of the agents is to try to maximize their individual rewards regardless of whether there are other agents in the environment or not [39]. However, if there is a multiagent team working for the same goal, giving the reward to the team instead of the individual agents and trying to maximize this reward is more appropriate. There are several recent models that are based on this approach such as the Decentralized POMDP (DEC-POMDP) and the Markov Team Decision Problem, which are more general versions of the POMDP model [6, 40].

2.1. Formal Definition of The DEC-POMDP Model

In this study, we use the model proposed by Bernstein *et al.* [6], which is called *Decentralized Partially Observable Markov Decision Process (DEC-POMDP)*. In this model, there are multiple agents which interact with the environment via their actions and observations at discrete time steps. The agents submit an action at each time step and the combination of these actions result in a state transition in the environment. At the same time, each agent receives an observation from the environment that gives information about the environment state. According to the action submitted and the observation received, each agent updates its state information and waits for the next time step to submit an action and receive an observation. The probabilities of the state transitions according to the joint actions, the observation probabilities according to the current state and the joint action, and the reward values according to current state and joint action are defined in the DEC-POMDP model. The reward values give an insight to the agents about the desirable actions in each state,

whereas the state transition and observation probabilities show how the environment interacts with the agents. The agents are expected to make their decisions based on their action and observation histories without sharing any information with the other agents; however, there can be a central planner at the learning phase that collects information from these agents.

The formal definition of the DEC-POMDP model is given below. A DEC-POMDP model consists of 7-tuple $(n, S, A, T, \Omega, Obs, R)$ where n is the number of agents, S is a finite set of states, A is the set of joint actions, T is the state transition function, Ω is the set of joint observations, Obs is the observation function and R is the immediate reward function.

A, the set of joint actions, is defined as the Cartesian product of A_i $(i = 1, 2..., n)$ i.e. the set of actions available to $agent_i$. As an example, consider the case of two agents in the environment; A_1, A_2 and A can be as following: A_1={GoForward, TurnLeft, TurnRight}, A_2={GoForward, TurnRight} and A = $A_1 \times A_2$ = { (GoForward, GoForward), (GoForward, TurnRight), (TurnLeft, GoForward), (TurnLeft, TurnRight), (TurnRight, GoForward), (TurnRight, TurnRight)}.

At each time step, each $agent_i$ selects an action from its action set A_i and the collection of the selected actions constitutes a *joint action*. A joint action $a \in A$ consists of n actions and $a = (a_1, a_2, ..., a_n)$, where a_i means the action selected by $agent_i$ from the action set A_i at the current time step. For the above example, it may be the case that $a_1 = GoForward$ (selected action of $agent_1$) and $a_2 = TurnRight$ (selected action of $agent_2$) at a time step, therefore, the joint action $a = (GoForward, TurnRight)$. In another case, it may be the case that $a_1 = TurnLeft$ and $a_2 = TurnRight$, therefore $a = (TurnLeft, TurnRight)$.

The transition function, T, determines the probabilities of the possible next states given the current state S and the current joint action a.

The set of joint observations, Ω, is the Cartesian product of Ω_i $(i = 1, 2..., n)$ which

is the set of observations available to $agent_i$. A joint observation $o = (o_1, o_2, ..., o_n)$ is received by the agents from the environment at any time step. According to the DEC-POMDP model, $agent_i$ has access to only o_i and o_i is a member of the Ω_i observation set.

Obs, the observation function, specifies the probability of receiving the joint observation o given the current state S and the current joint action a.

R, is the immediate reward function and it specifies the reward taken by the multiagent team given the current state and the joint action.

Due to the nature of the DEC-POMDP model, each agent should make its decision based on its action and observation histories. The mapping from these histories to actions is called a *local policy*. The *team policy* or *joint policy* consists of *local policies*, one for each agent in the problem. In this study, we consider $finite\ horizon$ DEC-POMDP problems and we aim to find the *joint policy* that will maximize the expected sum of rewards for the given horizon.

2.2. DEC-POMDP Problems

Although, in the literature, there are many problems defined to evaluate POMDP algorithms, there are only a few defined for the DEC-POMDP model. In our study, we consider the problems used in the DEC-POMDP literature. Below we give a short description of each problem.

2.2.1. Decentralized Tiger (Dec-Tiger) Problem

The Tiger problem is a well-known problem used for illustrating the single agent POMDP model [41]. Nair *et al.* [5] introduced the two agent version of this problem, as the *Decentralized Tiger (Dec-Tiger) Problem*. The short description of the problem can be found in Table 2.1. This is a popular problem used in many DEC-POMDP studies,

however, most of these studies consider only very small horizons.

Table 2.1. Definition of the Dec-Tiger problem.

Definition	-	There are two doors and there is a hungry tiger behind one of them and untold riches behind the other one. There are two agents that listen the doors and hope to open the door with untold riches.
States	2	The tiger is either behind the left door or the right door.
Actions	3	Open Left, Open Right and Listen
Observations	2	Tiger is at left, Tiger is at right
Reward	-	The reward of (Listen, Listen) action is -2. If the agents open the correct door they get a positive reward. If they open the wrong door, there is a high punishment.

2.2.2. Multi-access Broadcast Channel

The Multi-access Broadcast Channel is one of the most frequently used problems in DEC-POMDP studies [12, 14, 42, 27, 25, 15, 22]. The brief description of the problem is given in Table 2.2.

2.2.3. Meeting In a Grid

The Meeting in a Grid a problem is another famous problem in the DEC-POMDP literature. In this problem, the agents run on a $m \times n$ grid and try to meet in the same grid cell. The initial setup of this problem can be seen in Figure 2.1 for a 3×3 grid as an example and the short problem description is given in Table 2.3.

Table 2.2. Definition of the Multi-access Broadcast Channel problem.

Definition	-	Two agents use the same channel to broadcast a message to each other. Each agent has a message queue which is either empty or full. If they both try to send a message at the same time, a collision occurs. If only one agent chooses to send and there is a message in its queue, its message is transmitted. They try to utilize the channel in the alloted time.
States	4	The status of the message queues of the agents determines the state. A message queue can be either empty or full, so it can take 2 different values for each agent. Therefore there are $2 \times 2 = 4$ different states.
Actions	2	Send Message, Wait
Observations	2	Collision, No Collision
Reward	-	The agents get reward 1 when a message is transmitted, otherwise they get reward 0.

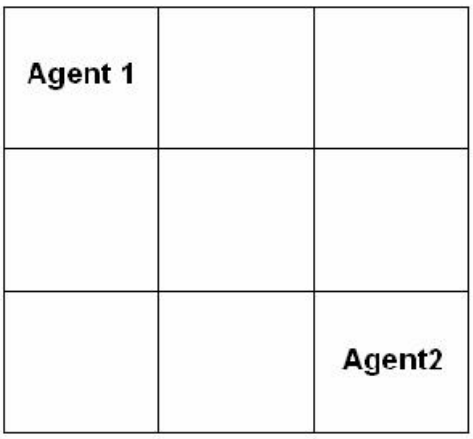

Figure 2.1. The initial setup of the Meeting in a Grid problem for a 3x3 grid.

Table 2.3. Definition of the Meeting In a Grid problem.

Definition	-	There are two agents moving on an $m \times n$ grid. The aim of the agents is to meet in the same grid cell and stay there as long as possible.
States	16	The positions of the agents determine the state. In a 2×2 grid, there are $2 \times 2 = 4$ different positions for each agent. Therefore, there are $4 \times 4 = 16$ different positions, meaning state, for the team
Actions	5	Go Up, Go Down, Go Right, Go Left, Stay
Observations	2	The agents can sense whether there is a wall to their right or left, so there are two observations. In the more general version, the agents can also see upper and lower cells. The agents can not sense each other.
Reward	-	When the agents are in the same grid cell, they get a reward of 1; otherwise the reward is 0.

2.2.4. Recycling Robots Problem

The Recycling Robots Problem is proposed by Amato *et al.* [43], however they tried to solve this DEC-POMDP problem for the infinite horizon case. We summarize the problem definition in Table 2.4.

2.2.5. Cooperative Box Pushing

Although the Cooperative Box Pushing is a well known robotics problem, it has been recently used in the DEC-POMDP domain and it is one of the largest problems considered so far [44, 28]. After Seuken *et al.* [28] formalized it as a DEC-POMDP problem, different versions of this problem that have slight modifications appeared in some studies. However, we present the original problem definition. The initial setup of the environment we used can be seen in Figure 2.2 and the short description of the problem is given in Table 2.5.

Table 2.4. Definition of the Recycling Robots problem.

Definition	-	In an office there are two agents which aim to collect cans. There are small and large cans; while the small cans can be picked up by one agent, the large cans need two agents to be picked up. Each agent has a battery having low and high states that can be observed by themselves but not by the other agent.
States	4	The state of the agents are determined by their battery level and each agent may be in low or high battery state. Therefore, there are $2 \times 2 = 4$ states in the environment.
Actions	3	Pick up a Small Can, Pick up a Large Can, Recharge.
Observations	2	Each agent can observe their battery state as low or high.
Reward	-	Picking up a small can brings reward 2 for each robot and picking up a large can brings reward 5.

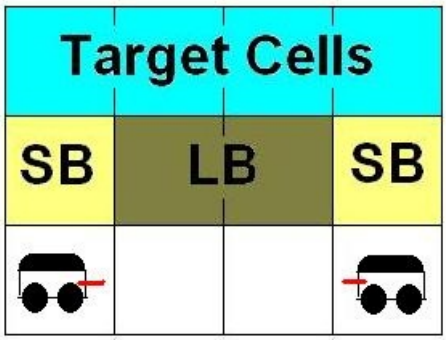

Figure 2.2. The initial setup of the Cooperative Box Pushing problem.

2.2.6. Fire Fighting Problem

This problem is recently introduced by Oliehoek *et al.* [26]. The short description is given in Table 2.6 as an example for the case of two fire fighter agents and three houses with three fire levels. Different versions of the problem can be generated by changing the

Table 2.5. Definition of the Cooperative Box Pushing problem.

Definition	-	There are two agents which aim to push two small boxes and a large box to the target cells. The small boxes can be pushed by one agent and the large boxes can only be pushed by two agents. The agents have a direction and can only move in the forward direction while they can only observe the cell in front of them. In order to move to the other directions they have to turn first. The agents move on a 4×3 grid.
States	100	The combination of different positions and the directions of the agents determines the number of states.
Actions	4	Move Forward, Turn Left, Turn Right, Stay.
Observations	5	Empty Field, Wall, Other Agent, Small Box, Large Box.
Reward	-	Moving the small boxes to the target area brings a reward of 10 and the large box brings a reward of 100. Hitting a wall or pushing the large box without the help of the teammate brings a reward of -5 and each time spent in the environment brings a reward of -0.1 for each agent

number of agents, houses or fire levels.

2.3. Histories and Policies

In a POMDP problem, an agent uses its past observations in order to make a decision. The observation history can be summarized by a probability distribution over environment states called *belief* which has been shown to be a sufficient statistics for representing the agent's past observations [45]. Since the belief of an agent can be represented as a vector, a policy for POMDP problems is basically a function mapping a belief vector to an action.

Table 2.6. Definition of the Fire Fighting problem with two agents and three houses with three fire levels.

Definition	-	There are two firefighter agents that aim to extinguish the fire in three houses. The fire level of each house is determined by an integer f. At each time step, each agent is free to go to any house. Being in a house helps to decrease the fire level there and if two agents are present in the same house, the fire is completely extinguished in that house. A house without a fire may catch fire from their neighbors and the fire level in a house without an agent may increase at each time step.
States	432	The combination of the positions of the agents and the fire levels of the houses determine the state. When the agent count or the the house count is increased, the state count is also increased. There are 432 states with 2 agents and 3 houses each having 3 fire levels.
Actions	3	Go to House 1, Go to House 2, Go to House 3.
Observations	2	Flame, No Flame
Reward	-	At each time step, the agents get $-f_i$ reward for each house burning, where f_i is the fire level of $house_i$.

In DEC-POMDP problems, the transition and observation functions are specified in terms of joint actions and observations, however each agent can access only its actions and observations. For this reason, an individual belief can not be computed [46] and the agents have to make their decisions based on their own past action and observation histories.

2.3.1. Histories

In this section, we define two histories that can be used for decision making in DEC-POMDP problems.

2.3.1.1. Action-observation history. If the action taken by $agent_i$ at time step t is a_t^i and the observation received is o_t^i, the action-observation history θ_t^i at time step t is:

$$\theta_t^i = (a_0^i, o_1^i, ..., a_{t-1}^i, o_t^i).$$

The joint action-observation history $\theta_t = (\theta_t^1, \theta_t^2, ..., \theta_t^n)$ defines the action observation history for all agents.

2.3.1.2. Observation history. If the observation received by $agent_i$ at time step t is o_t^i, the observation history ϕ_t^i at time step t is:

$$\phi_t^i = (o_1^i, ..., o_{t-1}^i, o_t^i).$$

The joint observation history $\phi_t = (\phi_t^1, \phi_t^2, ..., \phi_t^n)$ defines the observation history for all agents.

2.3.2. Policies

A DEC-POMDP policy is a mapping of histories to actions. In general, these are the action-observation histories. If the policy is a stochastic policy, given an action-observation history, the policy offers possible best actions together with a probability weight. An agent

chooses its current action randomly from these possible actions according to their probabilities, where the action with higher probability has a higher chance to be selected. On the other hand, if the policies are deterministic, meaning that there is only one best action for a given history, the history determines the best action exactly, therefore the observation history is sufficient to make a decision about the best action. In both cases, the policy can be represented as a tree. As expected, the policy tree for a deterministic policy is more compact compared to the stochastic policies. In the DEC-POMDP literature, deterministic policies are used to represent the finite horizon policies. In these trees, the nodes correspond to the best actions and the edges correspond to the possible observations for the time step under consideration. Each agent has its own policy tree. An example policy tree for an agent in the Dec-Tiger problem is given in Figure 2.3.

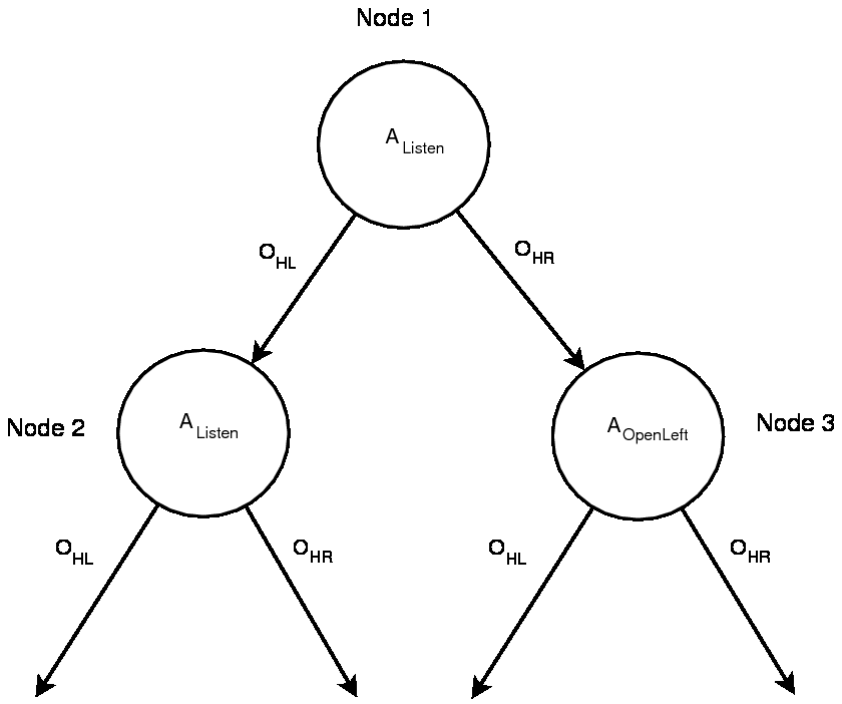

Figure 2.3. An example policy tree for an agent in the Dec-Tiger problem.

According to this policy, the agent starts in *Node 1* and it submits the A_{Listen} action. If it gets the observation O_{HL}, its current node becomes *Node 2* and it submits the A_{Listen} action again. However, if it gets the observation O_{HR}, its current node becomes *Node 3* and it submits action $A_{OpenLeft}$. In general, each agent in the environment may have a different policy.

Although policy trees can represent all possible DEC-POMDP policies, as the horizon gets larger, the size of the policy tree becomes very huge leading to memory problems. Therefore, it would be better to represent the policy in a more compact way which requires less memory and is as expressive as the policy trees. Finite state controllers are such structures, and they can be used to represent DEC-POMDP policies.

2.3.3. Finite State Controllers

A *Finite state controller (FSC)* is basically a finite state machine. The states in an FSC are not directly related with the environment states. In order to eliminate the ambiguity between the FSC states and environment states, we will refer to the FSC states as *nodes* from now on. The number of nodes in an FSC is determined by the particular DEC-POMDP solver. The transitions between the nodes depend on the current received observation. Each FSC node has a corresponding action and it is the best action for that node. An example FSC can be seen in Figure 2.4. There are three nodes in this FSC and it is designed for a problem having two observations. Neither the number of states in the environment, nor the number of actions or the horizon affect the structure or the size of the FSC. The size of a policy tree also is not affected by the number of environment states and the number of actions; however it grows exponentially with the growing horizon. Since each agent may have different policies, each of them will have different FSCs. Since infinite horizon policies can not be represented using policy trees, FSCs are especially used in infinite horizon DEC-POMDP studies.

In an FSC, there should be a starting node and node N_1 is the starting node in Figure 2.4. At the first time step, the agent submits action A_1, the best action for the node N_1. The agent then gets an observation from the environment and it updates its current FSC node according to the FSC definition. For example, if the agent gets observation O_1, a transition to node N_2 takes place and since the best action for N_2 is A_2, the agent chooses action A_2 in the next time step. If the agent gets observation O_2, there is a state transition to node N_3, and it chooses action A_1 in the next time step. The action submission and node update procedure continues up to the last horizon.

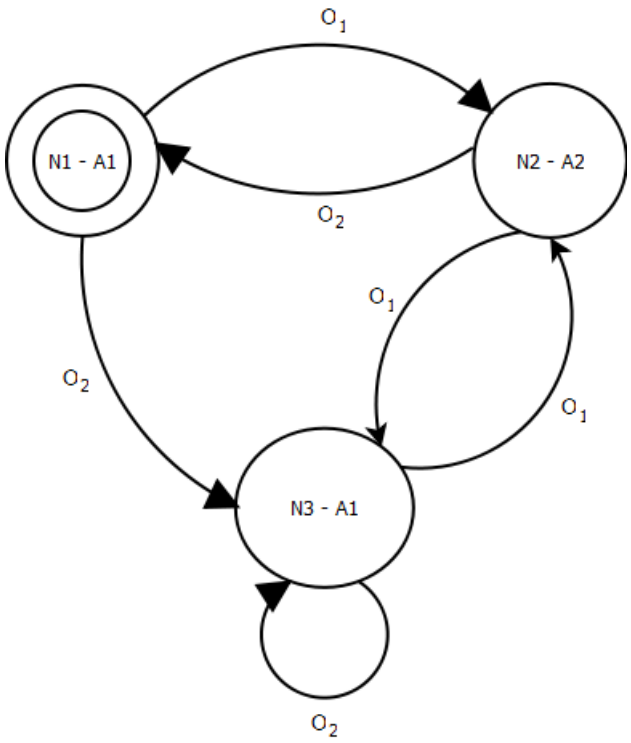

Figure 2.4. An example of a finite state controller.

2.4. DEC-POMDP Solution Techniques

Since it has recently been shown that solving DEC-POMDP problems is NEXP-complete, it is not possible to exactly solve DEC-POMDP problems having more than a few states or with more than a few horizon. On the other hand, it is important to study exact solutions to better understand the nature of the DEC-POMDP model. Due to this reason, there are studies on both exact and approximate solutions for DEC-POMDP problems.

2.4.1. Exact Solution Techniques

Although DEC-POMDP problems have NEXP complexity, there are many studies that try to solve these problems exactly. The early algorithms could only handle very small problems and short horizons, whereas the recent ones can handle much larger problems. Even though the exact solutions may be useful for some real-world problems, due to their high complexity, they are not suitable for using in problems with large state spaces.

2.4.1.1. Dynamic Programming for Partially Observable Stochastic Games. This is the first exact algorithm for solving DEC-POMDP problems. The Partially Observable Stochastic Games model is a more general version of the DEC-POMDP model in that the reward can either be given to the agents individually or to the team. This method represents the policy as a tree and tries to construct the tree in a bottom-up manner [12].

In order to calculate the value of a policy tree by using dynamic programming (DP), the authors formulate this calculation in a proper way. Each DP update has two stages. In the first stage, all possible $Q_t + 1$ policy trees are generated that make transitions to the root nodes of Q_t policy trees obtained previously and values of these policy trees are calculated using dynamic programming. In the second stage, the policy trees that are not promising to be a part of the optimal policy tree are determined and pruned.

The experiments with this algorithm is made on the Multi-access Broadcast Channel problem and it is observed that pruning significantly reduces the number of policy trees held in the memory. For example, in the horizon 4, only 1% of all possible policy trees are considered. In spite of this decrease, the required memory to generate possible trees for horizon 5 and prune them goes beyond the limits of the current computer memory capacities. Therefore, it is clear that this technique can not be used to solve complex problems and is of theoretical importance.

2.4.1.2. Multi-agent A*. MAA* is the first top-down optimal heuristic search algorithm for solving DEC-POMDP problems [14]. MAA* algorithm tries to find the joint policy for the team and it uses a top-down approach for this purpose.

The depth of the tree corresponds the horizon of the problem. At each MAA* iteration, the depth of the tree is increased by one. If we have a depth-n joint policy J_n, the algorithm firstly produces all possible $J_n + 1$ joint policies. The it prunes some of these policies by using some heuristics appropriate for the DEC-POMDP model. On the other hand, since the search space grows double exponentially, it starts to take too much time after a few horizons. This means that, only small problems for only a few horizons can be

solved exactly by using this algorithm.

2.4.1.3. Point-based Dynamic Programming for DEC-POMDPs. This is a dynamic programming approach that is offered by Szer [42] and it aims to avoid the linear programming phase in the previous dynamic programming approaches by computing relevant multi-agent belief states . They also offer an approximate solution approach. The approximate solution approach helps to obtain solutions for larger horizons. On the other hand, there is no significant improvement over the previous approach when the exact method is used.

2.4.1.4. Exact DP for DEC-POMDPs with lossless policy compression. Boularias *et al.* [11] mentions that one of the main reasons making DEC-POMDP problems intractable is the high dimensionality of the belief space. For this reason they offer a method to compress the policy belief space. By this way, they are able to obtain a speedup compared to dynamic programming, however, this still does not help to solve larger problems.

2.4.1.5. Optimal and Approximate Q-value Functions for Decentralized POMDPS. $Q - value$ functions that represent the value of taking an action at a given environment state are used in the MDP and POMDP literature. Oliehoek *et al.* [26] made the first study that formalizes the ways of applying $Q-Value$ functions to DEC-POMDP problems. It is claimed that calculating $Q - values$ exactly is infeasible except for very small problems. They offer some efficient ways of approximating these values. After the $Q - Value$ functions are calculated either exactly or approximately, these values are used by another algorithm called *Generalized MAA**, which extends *MAA** to allow it to use different heuristics, and a policy is produced.

2.4.1.6. Lossless Clustering Of Histories In DEC-POMDPs. This is a recent exact solution algorithm and its scalability with the horizon is better than the previous algorithms [15]. Since the number of possible observation sequences grows exponentially with the growing number of horizons; it becomes much harder to calculate the optimal policy. On the other hand, it is proven that if two histories satisfy a criterion, their optimal values are

the same, hence they can be treated as one policy. This technique extends *Generalized MAA** to allow it to cluster histories. In this way, a significant speedup is provided while solving DEC-POMDP problems optimally and allowing the solution of larger problems whenever lossless clustering is possible for the problem considered. However, this is not possible for all problems and clustering brings either very small or no speed up for some exceptional cases. It is mentioned that it is an open question which problems are suitable for such clustering. Spaan *et al.* [16] improved this method by incrementally expanding the child policies only when a next child has the highest heuristic value. They also offered a more compact policy representation and they were able to produce optimal solutions for larger horizons than the previous studies.

2.4.1.7. An Investigation Into Mathematical Program. for Finite Horizon DEC-POMDPs. Another recent study offers to represent policies as a sequence-form and they convert DEC-POMDP problems into Mixed Integer Linear Programming (MILP) problems using this representation [9]. The optimal policy is obtained by solving the generated MILPs. Although this technique brings a speedup when compared to the classical dynamic programming approach, it can still handle very low horizons. On the other hand it is an important study that brings the usage of mathematical programming into the DEC-POMDP literature and it contributes to the better understanding of DEC-POMDPs.

2.4.1.8. Optimizing Fixed-size Stochastic Controllers. All studies mentioned above offer solutions for finite horizons. Amato *et al.* recently offered a technique that uses finite state controllers for representing the policy and finding the optimal solution for a given FSC size and for the infinite horizon case [8]. They formulate the problem as a nonlinear program (NLP) using this representation and they obtain the optimal policy by solving an NLP. Although solving the NLP is intractable for large problems and large FSCs, their results show that this new method outperforms all other methods when they are applied to the infinite horizon case.

2.4.2. Approximate Solution Techniques

Due to the high worst case complexity of the exact solution techniques, approximate solution techniques that can produce near optimal solutions for larger problems and larger horizons are desirable for real-world problems that are more complex than the DEC-POMDP test problems.

A policy P is an ϵ-optimal policy, if $Value(P)/Value(Optimal\ policy) \geq 1 - \epsilon$. Finding ϵ-optimal policies for DEC-POMDP problems has also been shown to be NEXP-complete [47]. Therefore, in order not to deal with NEXP-complete complexity, approximate solution techniques do not concentrate on ϵ-optimal solutions.

2.4.2.1. Bounded Policy Iteration. Bounded Policy Iteration (BPI) is an approximate solution technique for solving infinite horizon DEC-POMDP problems. It is an extended version of the BPI algorithm proposed to solve POMDP problems [6]. It uses finite state controllers (FSC) to represent the agent policy [19].

In some of the DEC-POMDP solution techniques, the memory requirement grows quickly, however this algorithm uses a fixed amount of memory. At each iteration, algorithm tries to improve the obtained FSC using linear programming and it is guaranteed that the obtained FSC at the end of the iteration has at least the same value as the previous FSC for all possible initial states.

In addition to using FSCs, another mechanism called *correlation device* is used in this algorithm. This is also a finite state machine, however the state transition happens at each time step without any input. It is claimed that using the correlation device helps to obtain better strategies. The algorithm either tries to change the state transition of an FSC node or the correlation device and aims a better strategy. If no improvement occurs for any FSC node and for the correlation device, it means that a local optimal solution is reached. Then, by increasing the FSC size or controller device size, another solution can be searched.

The results are reported for the Multi-access Broadcast Channel and the Meeting in a Grid problems. It is observed that increasing FSC size do not always result in a better solution and using correlation device helps in obtaining better solutions for most cases.

2.4.2.2. Optimal Fixed Size Controllers. This algorithm is similar to BPI method in the way they represent the policy, however a correlation device is not used in this technique [43]. On the other hand, the solution is not found for any initial state but for only a particular given initial state and the problem is formalized as a nonlinear program (NLP). The solution of the NLP gives the optimal solution for the given number of nodes and the given initial state.

In order to solve the NLPs, two programs called as *snopt* and *filter* are used. The obtained values are similar for these two methods. When compared to BPI algorithm, this algorithm obtains strategies having better values. The runtime is also comparable for small controller sizes. However, with the increasing controller size, the required time to solve NLPs grows quickly. Therefore, we expect the algorithm to have a difficulty in finding solutions for complex problems that require larger controllers.

2.4.2.3. Memory-Bounded Dynamic Programming (MBDP). The MBDP algorithm represents the policy as a tree and uses the combination of both the top-down and the bottom-up approaches [27]. The algorithm progresses mainly in a bottom-up manner. However, while growing the depth of the tree by one, instead of finding all possible transitions, it tries to find the transitions from the most likely belief states. While calculating the most likely belief states, it uses top down heuristics. With this technique they are able to obtain a linear time complexity with respect to the horizon length. For this reason, the algorithm is able to solve the Multi-access Broadcast Channel and the Dec-Tiger problem for much larger horizons when compared to previous algorithms by obtaining comparable reward values

2.4.2.4. Improved MBDP (IMBDP). The IMBDP algorithm is the improved version of the previous algorithm. The previous algorithm has linear time complexity with the hori-

zon, however it is exponential with the number of observations. The new technique eliminates this problem and the algorithm is expected run much faster. In order to do this, it calculates the most likely observations while calculating the most likely belief states and this significantly decreases the number of trees to process [28].

The results reported in the paper is said to be preliminary and the new algorithm is not tested with the old problems. The Cooperative Box Pushing problem is proposed to the DEC-POMDP world in this paper and is used for the experiments. It is said that none of the previous algorithms including MBDP can solve this problem due to the number of states and observations. Just the obtained reward values are reported for this problem, but no run-time is given.

2.4.2.5. MBDP with Observation Compression. Carlin *et al.* [21] proposed a value-based compression approach in order to reduce the running time and space requirements of the *MBDP* algorithm. The new algorithm called *MBDP-OC*, produces reward values even better than the *IMBDP* algorithm for the two problems tested.

2.4.2.6. Point Based Incremental Pruning. *PBIP* is another recent method that is based on the *MBDP* method which only searches promising policies by making use of some heuristics and produces better policies than the previous *MBDP* alternatives [22].

2.4.2.7. Incremental Policy Generation with PBIP. Amato *et al.* [48] proposed a method called *Incremental policy generation* (*IPG*) based on a reachability analysis of the state space. It can be used to produce an optimal solution and also is compatible with dynamic programming algorithms. It is incorporated into *PBIP* and a significant performance gain is obtained.

2.4.2.8. Point-Based Policy Generation. Wu *et al.* [49] offered an algorithm, *Point-Based Policy Generation (PBPG)*, that constructs the best joint policy for each reachable belief state directly, instead of creating a large set of candidates first and they have outperformed

the *IPG* approach.

2.4.2.9. Constraint Based Point Backup. This is one of the few studies for solving DEC-POMDP problems using evolutionary algorithms. The idea behind the algorithm is very simple. Each possible observation sequence for the given horizon and for all smaller horizons is encoded into the chromosome for each agent. Therefore, the size of the chromosome is exponential in terms of the horizon. This makes it impossible to store the chromosome in the memory after only a few horizons. Each gene has a corresponding observation sequence and the value of the gene is the optimal action for this observation sequence. Namely, while the agent is running on the environment with the given policy it finds the gene from the chromosome that corresponds to its observation history and chooses the value in the gene as its current action.

The fitness calculation process is not mentioned in the paper and experiments are only made on the Dec-Tiger problem. The obtained values are given for up to horizon 5 and the reward values are not impressive. On the other hand, there are exact solution algorithms that reaches this limit and an approximate algorithm is expected to handle larger horizons.

2.4.2.10. The Cross-Entropy Method for Policy Search in Decentralized POMDPs. *The Cross Entropy* (CE) is a recently introduced method for solving combinatorial optimization problems and it has also been applied to the solution of DEC-POMDP problems [25]. Since the CE method is able to search huge spaces efficiently and DEC-POMDP problems have huge policy spaces, using CE method for DEC-POMDP problems seems to be promising. One of the most time consuming parts of this method is where the expected reward of a generated policy is calculated. Since it takes too much time with the growing horizon, an approximate calculation technique that basically makes a number of simulations with the given policy is also used. Especially, the results reported with the exact reward calculation is good in terms of received reward, however the horizon that can be practically reached is small. Although approximate reward calculation helps to reach larger horizons, it results in much worse policies in terms of reward for some cases.

3. MODELING AS A PARTIALLY OBSERVABLE MARKOV DECISION PROCESS

The POMDP is widely studied and there are many efficient algorithms to solve POMDP problems. If we can convert a DEC-POMDP problem into a POMDP problem, we can solve it using one of the known efficient POMDP solution techniques and we can use the obtained solution to make decisions in DEC-POMDP environments. However, since the DEC-POMDP problem set is more general than the POMDP set, it is obvious that it may not be possible to make a perfect conversion while converting the DEC-POMDP problem into a POMDP problem and applying the obtained POMDP solution in the DEC-POMDP environment.

3.1. Proposed Approach

In this approach, we assume that there is a single global agent who can receive all the individual observations at a time step and use the agents in the environment as its actuators. At each time step, the agents receive individual observations and we assume that the global agent can receive all of them. Also, each agent can make different actions at each time step and the global agent controls the actions of them, and this can be thought as a joint action. By this way, we have a POMDP problem which has joint observations and joint actions. Then, we solve this POMDP problem and share the obtained strategy to the agents. While agents run on the environment, each one holds its own belief state and finds the best joint action for its belief state. Then it chooses the action that it should do to realize the joint action and updates its belief states approximately. For example, in the Dec-Tiger problem, if the $agent_1$ decides that the best joint action for its current belief is $(Listen, OpenLeft)$, it should submit $Listen$ action.

3.1.1. Belief Update in POMDPs

The belief is a probability distribution over the state space S and $b(s)$ is the probability that the environment state being s. If the agent submits action a and receives observation o from the environment, it can update its current belief $b(s)$ using the following equation:

$$b'(s') = Obs(o|s', a) \sum_{s \in S} T(s'|s, a)b(s) \tag{3.1}$$

In POMDP environments, the agents can update their belief perfectly, since they have access to all the information needed to make the belief update.

3.1.2. Belief Update in DEC-POMDPs

Equation 3.1 can also be used in DEC-POMDP environments, because the overall behavior is the same in the DEC-POMDP model. However, there is a single action and single observation in the POMDP model at each time step while there are multiple actions and observations in the DEC-POMDP model. An agent should have access to all actions submitted and all observations received for a perfect belief update. Since the DEC-POMDP model does not allow communication among the agents, each agent has access to only its action and observation, therefore making a perfect belief update is not possible. For this reason, we offer an approximate belief update procedure for DEC-POMDP environments and use this method after we obtain a strategy from the POMDP solver. The general equation is very similar to the POMDP case, except that instead of a single action and observation, there are joint actions and observations:

$$b'(s') = Obs((o_1, o_2, ..., o_n)|s', (a_1, a_2, ..., a_n)) \sum_{s \in S} T(s'|s, (a_1, a_2, ..., a_n))b(s) \tag{3.2}$$

The problem is that, Equation 3.2 requires the knowledge of the actions and the observations of the other agents. Therefore, each agent should estimate the actions and the observations of the other agents. The DEC-POMDP model allows the agents to know

the strategy of the others, therefore, if an agent knows the belief state of another agent it can calculate its best action. For this reason, we allow each agent to hold a belief state for every other agent and estimate its action using this belief. For the observations of the other agents, we assume that since the agents are in the same environment, their observations should be similar, therefore each agent assumes that the other agents also get the same observation. With that assumption, each agent updates its belief state using the Equation 3.2 as well as its belief estimation for all other agents.

In our experiments with this method, we realized that approximate belief update does not produce good results for some problems, therefore we wanted to test the case where there is a communication between agents. If the agents can share their action and observation, each agent can make a perfect belief update as in the POMDP case. For all of the problems, we also tested this case.

3.1.3. Policy Evaluation

It is hard to measure the quality of a DEC-POMDP solution. Although there are some formulas proposed for some policy types like finite state controllers, it is hard to find such formulas for each policy type. Since, the aim of the policy evaluation is to find an expected reward, if we make several simulations and calculate the average reward of them this can give an insight about the expected reward. The more number of simulations implies more accurate calculations. For this purpose, we developed a simulator that can model a given DEC-POMDP problem and simulate the world according to the actions of the agents and calculate the obtained reward. By repeating this operation several times, we calculate an approximate expected reward and this gives an indication about the success of the algorithm.

Amato [50] proposed a format for defining DEC-POMDP problems in a standard way and our simulator can run any problem given in this format. Besides the problem definition, it gets the policy of the team and the policy representation type as the input. We use the same simulator to calculate the expected reward for different policy types that

we use in the following sections. Depending on the policy type and the policy given, simulator finds the actions of the agents at each time step and changes the environment state accordingly. Then, it calculates the observations that will be received by the agents and updates the belief of the agents. Therefore, it simulates both the environment and the agents. In order to make it more flexible, it can be modified to work in a distributed manner and the simulation of the agents can be separated from the simulator in the future.

3.2. Experiments and Results

In order to solve POMDP problems, we use the ZMDP package that is developed by *Smith* [51] which is one of the best POMDP packages available and our experiments show that it can even handle large DEC-POMDP problems when converted to a POMDP problem.

We made our tests on the following DEC-POMDP problems:

- Dec-Tiger,
- Multi-access Broadcast Channel,
- Meeting in a Grid,
- Cooperative Box Pushing,
- Recycling Robots, and
- Fire Fighting.

For each problem, we made tests with several horizons and we tested the behavior in both cases where the communication is allowed and not allowed. While calculating the expected reward of a strategy, we made 100000 simulations and reported the average of the rewards obtained in these runs. Since the ZMDP package produces solutions for infinite horizon, we obtain a single solution for each problem and calculate the expected reward for each horizon using the same strategy. We compared our results with the exact approaches that can handle the largest horizons and the best approximate approaches in terms of reward and horizon. Since the optimal results are obtained under the DEC-POMDP assumptions

meaning that the agents act without communication, there are cases that we report better results than the optimal DEC-POMDP results when the agents can communicate. While reporting the results of the other studies, "-" sign in the result tables means that the result is not available in the reference paper and "X" means that the algorithm is not able to produce a solution for the corresponding horizon.

3.2.1. DEC-Tiger Problem

The Dec-Tiger problem is a difficult problem in the sense that the punishment of a wrong action is very high. In our technique, the agents learn a strategy with the assumption that they know the other agents' actions and observations and they can make a nearly perfect belief update. When there is a slight deviation from the correct belief, synchronization problems appear and this results in wrong actions and high punishments. This happens especially in the case that the agents do not communicate. For this reason, our results are worse than the previous studies for this case. On the other hand, when we assume that the agents can communicate, the results become even better than the ones in the literature, because the information stored in each individual agent is very important for this problem in order to have a global view of the environment. The results can be seen in Table 3.1 with the best exact and approximate algorithms that were tested on this problem [27, 22, 16].

3.2.2. Multi-access Broadcast Channel

We made tests for several horizons up to 1000 for this problem and present our results in Table 3.2 together with the best exact and approximate algorithms that were tested on this problem [27, 15, 16]. Both the results that we obtained and all the other results for this problem are very close to each other. For our algorithm, the results are slightly worse when there is no communication and the results are either better than the previous results or the same with them when the agents are allowed to communicate. We see the benefit of the communication also in this problem.

Table 3.1. Test results for the Dec-Tiger problem using the MAP approach.

	APPROXIMATE		OPTIMAL		
Horizon	MBDP	PBIP	GMAA*-ICE	MAP	
	[27]	[22]	[16]	(no comm.)	(comm.)
2	-4.00	-	-4.00	-14.18	10.78
3	5.19	-	5.19	-16.29	12.64
4	4.80	-	4.80	-28.03	22.57
5	5.38	-	7.03	-30.11	26.53
6	9.91	-	10.38	-42.50	34.95
8	9.42	-	X	-57.06	47.68
10	13.49	13.6	X	-71.05	60.19
100	93.24	147	X	-709.30	643.77

3.2.3. Meeting In a Grid

The tests in the DEC-POMDP literature with this problem are mostly made on a 2×2 grid, so we also made our tests on it. The results for this problem are reported only up to horizon 6 in the literature; however our algorithm is able handle much larger horizon values. Besides horizon 6, we also report our results for horizon 50 for future reference. Our results are given in Table 3.3 with the best approximate and exact results obtained so far [25, 16]. The results obtained for this problem are as expected. For all of the cases, the results obtained using communication outperforms the optimal results. The results without communication is worse than the previous studies and the difference with the optimal results is considerably high.

3.2.4. Recycling Robots Problem

For the finite horizon case, to the best of our knowledge, there are only two studies reporting results and they propose an exact solution algorithm [15, 16]. In the first study (*Lossless Clustering*), they use a discount factor of 0.9 and they can solve the problem for

Table 3.2. Test results for the Multi-access Broadcast Channel problem using the MAP approach.

	APPROX.	OPTIMAL			
Horizon	MBDP	Loss. Clust.	GMAA*-ICE	MAP	
	[27]	[15]	[16]	(no comm.)	(comm.)
2	2.00	2.00	-	1.90	2.00
4	3.89	3.89	-	3.70	3.89
8	7.49	7.49	-	7.31	7.49
10	9.29	9.29	-	9.09	9.30
20	18.29	18.31	18.31	18.08	18.29
50	45.29	X	45.50	45.10	45.44
100	90.29	X	90.76	90.13	90.71
250	-	X	226.50	225.14	226.48
500	-	X	452.79	450.06	452.83
900	-	X	814.71	810.27	814.65
1000	900.290	X	X	899.98	905.21

horizon 15 at most [15]. In the second study (*GMAA*-ICE*), they can solve this problem for horizon up to 70 and their discount factor is 1.0 [16]. Since the discount factor is a parameter that is not affecting the difficulty of the problem, we compare our results with the *GMAA*-ICE* algorithm. The results from this study and our results are given in Table 3.4. Our results is about $20 - 30\%$ worse than the optimal results when agents do not communicate. On the other hand, we obtain better results than the optimal ones when the agents communicate.

3.2.5. Cooperative Box Pushing Problem

This problem is one of the most complex problems in terms of the number of states and the ZMDP package is able to solve this problem. Our average reward is too low when the agents do not communicate. This is due to the problems in the belief update for this

Table 3.3. Test results for the Meeting in a Grid Problem using the MAP approach.

	APPROXIMATE	OPTIMAL		
Horizon	Cross Entropy	GMAA*-ICE	MAP	
	[25]	[16]	(no comm.)	(comm.)
2	0.91	0.91	0.82	0.96
3	1.55	1.55	1.29	1.60
4	2.24	2.24	1.75	2.32
5	2.96	2.97	2.14	3.10
6	3.64	3.72	2.62	3.84
50	X	X	21.77	40.34

problem. For some cases, the belief of the agents become invalid. When such a situation occurs, we rollback the update and the agent remains in the old belief state. If this happens in the initial steps, the agents may always stay in the same belief state and may never go behind a box and push it. In this case, they get a -2 reward at each time step and the results in the Table 3.5 are obtained. When they communicate, the situation is the same with horizon 2. For horizon 3, our result is better than the IMDP approach [28] and very close to the optimal [16]. For horizon 4, our result is better than the optimal. For the higher horizons, there is no optimal result reported yet, however our results are better than the IMDP approach for all test horizons.

3.2.6. Fire Fighting Problem

The Fire Fighting is the problem having the most number of states we have attempted to solve, namely 432 states. The ZMDP package is able to solve this problem also. (*GMAA*-ICE*) algorithm gives the optimal results for this problem up to horizon 6 and we compare our results with it in Table 3.6. The results obtained have a similar behavior to the other problems. We obtain better results than the optimal ones when agents communicate. However, we obtain much worse results than the optimal ones when they are not allowed to communicate. Therefore, this method is not suitable to be used in this

Table 3.4. Test results for the Recycling Robots problem using the MAP approach.

	OPTIMAL		
Horizon	GMAA*-ICE	MAP	
	[16]	(no comm.)	(comm.)
5	16.49	13.34	17.52
15	47.25	35.44	50.13
18	56.48	42.31	59.99
20	62.63	46.78	66.48
30	93.40	68.93	99.32
40	124.17	91.16	131.98
50	154.94	113.57	164.67
60	185.71	135.58	197.39
70	216.48	157.91	230.05
100	X	224.63	328.28

problem.

3.3. Discussion

The DEC-POMDP model is proposed for multiagent teams that work for the benefit of the team instead of the individual benefits and this requires a synchronization between the agents. While the easiest way to synchronize is the communication, the model assumes that the agents can not communicate. Therefore, the agents should develop their strategy with this assumption. On the other hand, in the MAP approach we create a strategy assuming that the agents can perfectly update their beliefs, and this requires the sharing of actions and observations among the agents. After the agents learn a strategy with this assumption, when they are not allowed to communicate, this causes them to act in a more individual way rather than a team and the average reward obtained for this case becomes worse than the optimal results. Depending on the characteristics of the problem, the results can be much worse and our experiments support this. The results can be acceptable for

Table 3.5. Test results for the Cooperative Box Pushing problem using the MAP approach.

	APPROXIMATE	OPTIMAL		
Horizon	IMBDP	GMAA*-ICE	MAP	
	[28]	[16]	(no comm.)	(comm.)
2	17.60	17.60	-0.40	-0.40
3	65.68	66.08	-0.60	65.99
4	97.91	98.59	-0.80	99.36
8	182.92	X	-1.60	191.94
10	189.32	X	-2.00	221.03
20	415.25	X	-4.00	476.89
50	1051.82	X	-10.00	1220.88
100	2112.05	X	-20.00	2462.03

only the Multi-access Broadcast Channel problem. The results with Dec-Tiger and the Cooperative Box Pushing problems are much worse than the optimal. On the other hand, when we allow the agents to communicate, we obtain better results than the optimal results for all of the problems as we expected, because the communication helps the agents to synchronize better.

As a result, we do not recommend using this method under the DEC-POMDP assumption. On the other hand, if the agents are allowed to communicate, the results look impressive, however, since previous DEC-POMDP algorithms worked without communication, we do not know how much benefit they can gain from the communication, so we can not make a fair comparison for the the case where the communication is allowed.

Table 3.6. Test results for the Fire Fighting problem using the MAP approach.

	OPTIMAL		
Horizon	GMAA*-ICE	MAP	
	[16]	(no comm.)	(comm.)
2	-4.38	-5.13	-4.38
3	-5.74	-7.04	-5.72
4	-6.58	-8.38	-6.36
5	-7.07	-10.22	-6.95
6	-7.18	-11.88	-7.13

4. USING EVOLUTION STRATEGIES TO SOLVE DEC-POMDP PROBLEMS

The *Modeling as POMDP* approach in Chapter 3 mainly depends on a POMDP solver. Therefore, the success of the underlying algorithm also determines the success of our algorithm. Additionally, the assumptions made while updating the belief may lead to unacceptable results in most of the problems when communication is not allowed. In order to overcome these problems, we looked for other techniques and we decided to use the algorithms from evolutionary algorithms family that are known to work well with optimization problems.

We used two different structure for decision making and for both of the cases we represent the genes using real numbers. Since the evolution strategies (ES) are known to work well with real valued chromosomes, we preferred to use this technique [52].

4.1. Evolution Strategies

Evolutionary algorithms are based on natural selection and genetics. A variety of evolutionary computational models have been proposed. Common to all is the concept of simulating the evolution of individual structures using genetic operators such as selection, mutation, reproduction, and recombination. Although available for more than 40 years in one form or another, they have become widely accepted as practical, robust optimization and search methods since 1990s [53]. They provide approximate solutions to problems that are difficult solve exactly. There are four main evolutionary computation models, namely, genetic algorithms [54], evolutionary programming [55], evolution strategies [56], and genetic programming [57].

The basic idea of the evolutionary algorithms is starting with an initial candidate solution and improving the solution over the generations. The set of candidate solutions is called as the *population* in the evolutionary algorithms terminology and each candidate

solution is called as *chromosome*. For a given set of chromosomes, it should be possible to tell how good a solution is for the problem at hand. This is called the *fitness* of the chromosome. After testing the fitness of the chromosomes in a population, a new population is generated using the genetic operators like crossover and mutation.

Evolution strategies (ES) is an evolutionary algorithm technique that uses real numbers for encoding. The main idea behind evolution strategies is the *principle of strong causality*, which means a small change to one encoding of the problem causes only a slight change on its optimality [58]. By using this principle, controlled small changes are made on the chromosomes and convergence is achieved in a more controlled way compared to the other evolutionary algorithms. To achieve this, another vector, called *strategy parameters* is used, which affects the mutation of the chromosomes.

4.1.1. Mutation

Mutation is the primary genetic operator in ES where both the chromosomes and the strategy parameters are mutated. Strategy parameters are used to control the mutation of the chromosomes. Let the chromosome *i*, which has m genes, be,

$$c_i = (g_1, g_2, ..., g_m), \tag{4.1}$$

and the strategy parameters for chromosome i ,

$$sp_i = (p_1, p_2, ..., p_m). \tag{4.2}$$

Then, the mutated chromosome is obtained by adding a normally distributed value to each gene of the chromosome:

$$c_i{}^{mut} = c_i + N_0(sp_i) = [g_1 + N_0(p_1), g_2 + N_0(p_2), ..., g_m + N_0(p_m)] \tag{4.3}$$

where $N_0(p_j)$ is the normal distribution with mean *0* and standard deviation p_j. The mutation of the strategy parameters is achieved via the following equation:

$$sp_i{}^{mut} = (p_1 \times A_1, p_2 \times A_2, ..., p_m \times A_m), \tag{4.4}$$

where A_j depends on the value of a uniformly distributed value *E* over *[0,1]* as follows:

$$A_j = \begin{cases} \alpha & \text{, if } E < 0.5 \\ 1/\alpha & \text{, if } E \geq 0.5 \end{cases} \tag{4.5}$$

The parameter α is called *the strategy parameters adaptation value* and if the number of parameters is less than 100, it is advised to be chosen as 1.3 [56]. For more parameters, α should be smaller. When mutating sp_i, new *E* values should be chosen for each A_j.

4.1.2. Crossover (Recombination)

There are two types of recombination operators used in the ES. The first one is discrete recombination which is similar to uniform crossover in genetic algorithms. In this

case, each child has equal probability of receiving each gene from each parent. Intermediate recombination is where each child parameter is calculated by getting the mean of the parent parameters.

4.2. Proposed Approach

4.2.1. Policy Representation

In our MAP approach, we represented the DEC-POMDP policies as belief vectors. In the ES approach, we also use the belief vector representation, however, due to its scalability problem with the increasing state count, we investigated alternative approaches. As mentioned in Section 2.3.2, policy trees are used in most of the finite horizon DEC-POMDP studies. Since the number of observation histories grows exponentially with the growing horizon and more recent observations give more information about the current state, we thought that we can use only the last n observations for decision making and feed them to a neural network that will help in finding the optimal action. We made our experiments with these two approaches. More details about how they are converted into a chromosome are given in the related sections.

4.2.2. Fitness Calculation

There are some problems that can not be solved in polynomial time but a solution proposed for such problems can either be tested in polynomial time or at least an indication about the fitness of the solution can be given. For example, in the Traveling Salesperson Problem, when you are given a solution you can not tell whether it is the optimal solution directly, but you can tell the cost of the solution easily [59]. After getting this cost, one can look for a better solution and this is the case for many such difficult problems. However, even fitness calculation itself is a very time consuming task for DEC-POMDP problems. When an agent starts from a belief state with a given policy, there are many possible paths it can traverse depending on the horizon and the nondeterminism in the environment. This calculation is exponential in terms of the horizon.

Whenever either an explicit fitness function is not available or calculating the fitness is very costly, approximate fitness calculation methods can be used. Approximation can be made at the functional level or at the problem level. If the problem can be approximated as another problem and its fitness function can then be calculated more efficiently, the problem can be defined in that way and solved. If this is not possible, the fitness function can be calculated approximately. There are several methods to approximate the fitness function such as polynomial methods, least square method, Kriging method, neural networks and support vector machines. No matter which method is used, it is generally very difficult to obtain a globally correct approximation model and the evolutionary algorithm may converge to a false optimum. In order to prevent this, the approximate model should be used together with the original fitness function. This is called as *evolution control* [60].

4.2.2.1. Statistical Approach to Fitness Evaluation. For the fitness calculation, we used the DEC-POMDP simulator developed for finding the expected reward in the MAP approach. To calculate the expected reward of a given policy, we run the simulation up to given horizon and get the reward taken. This is called a $sample$. Since the process is probabilistic, repeating this process as much as possible and calculating the average reward may give a better indication about the fitness of the solution. However during our simulations, we observed that just calculating the average reward for some number of samples and using it as the fitness is not a good idea, because even if you run the simulation with the same policy for the same number of samples, you sometimes obtain very different results, especially in the early generations, since the policy generated so far possibly consists of just random actions. Therefore, the solution that is chosen as the best solution may be disappointing.

To overcome this problem, we used a novel approach which uses statistical methods. Since, due to its probabilistic nature, there are many possible paths for simulation, there are also many possible reward values for each sample. We assume that, these reward values come from a normal distribution and we compute the confidence interval for the mean reward. The normal distribution assumption may seem doubtful at first, however, it has been shown that when estimating the confidence interval for the mean of non-normal distributed

data, there is no significant difference between approaches like Box-Cox transformation, bootstrapping or normal distribution [61] beyond a certain number of test samples; these tests give very close results for 50 samples. So after this many samples, the other methods do not have a significant advantage in terms of accuracy over the normal distribution assumption. On the other hand, their run times are significantly higher than the normal distribution based approach.

By getting n samples with our simulator, we obtain n numbers from these reward values. Using these values and Equation 4.6, we can calculate a confidence interval for the mean on the original reward population and this gives us the expected reward:

$$M - t\sigma \leq \mu \leq M + t\sigma \tag{4.6}$$

where μ is the mean of the reward population, i.e. the expected reward, M is the mean of the sample population, σ is the standard deviation of the samples, and t is the value given by t-table, that depends on degrees of freedom (number of trials) and the confidence desired [62]. We calculate this range with 99% confidence. Here we make a conservative approximation and choose the minimum value in this range as our fitness value. We say that, the expected reward is higher than this value with 99% confidence.

4.2.2.2. Determining the Number of Samples. Although increasing the sample count gives more realistic fitness values, the running time of our algorithm also increases. Therefore, we should repeat the simulation as few times as possible in order to converge quickly, but this number should also be sufficient to calculate the fitness appropriately. For this purpose, we designed an experiment. We started an evolution strategies run and stopped it at the 10th generation since in the earlier generations the effect of randomness is more pronounced. We calculated the fitness values of these chromosomes by getting several number of samples, changing from 10 to 500. We wanted to see the number of samples after which

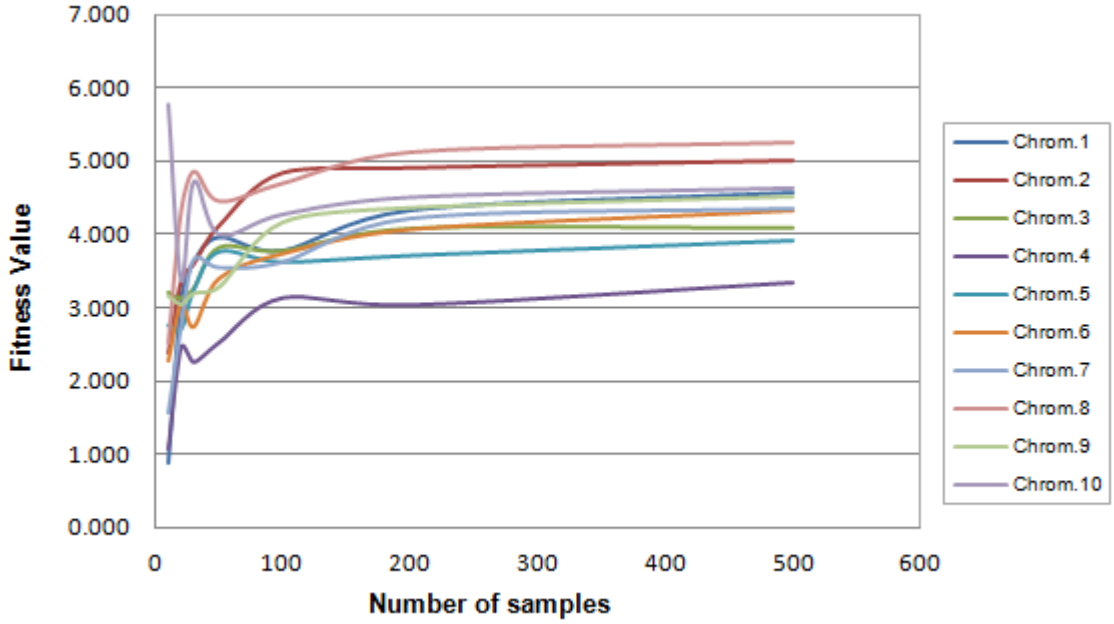

Figure 4.1. Fitness values of 10 chromosomes for several number of samples.

the ordering of chromosomes do not change. Since, the main job of the fitness calculation is to order the chromosomes from the best to the worst, this would give us the sufficient number of samples to calculate the fitness. The obtained curve can be seen in Figure 4.1. Although we used more chromosomes during the learning, we just included 10 of them in the figure to make the figure less complex, since this is sufficient to illustrate the idea.

As can be observed from the Figure 4.1, the ordering of the chromosomes changes very frequently for small number of samples and the ordering do not change significantly later. The ordering becomes stable after $150 - 200$ samples, so this number seems to be sufficient for evaluating the fitness approximately. On the other hand, as mentioned above, even 50 samples is enough to make the normal distribution assumption even if the underlying distribution is non-normal. We wanted to see the effect of this on the convergence and devised another experiment. We made 10 ES runs by evaluating the fitness with 50 samples and another 10 runs by calculating the fitness with 150 samples. We calculated the average of 10 runs and plotted them. The results are shown in Figure 4.2. It is clear from the figure that, there is no significant difference between these two cases and the obtained

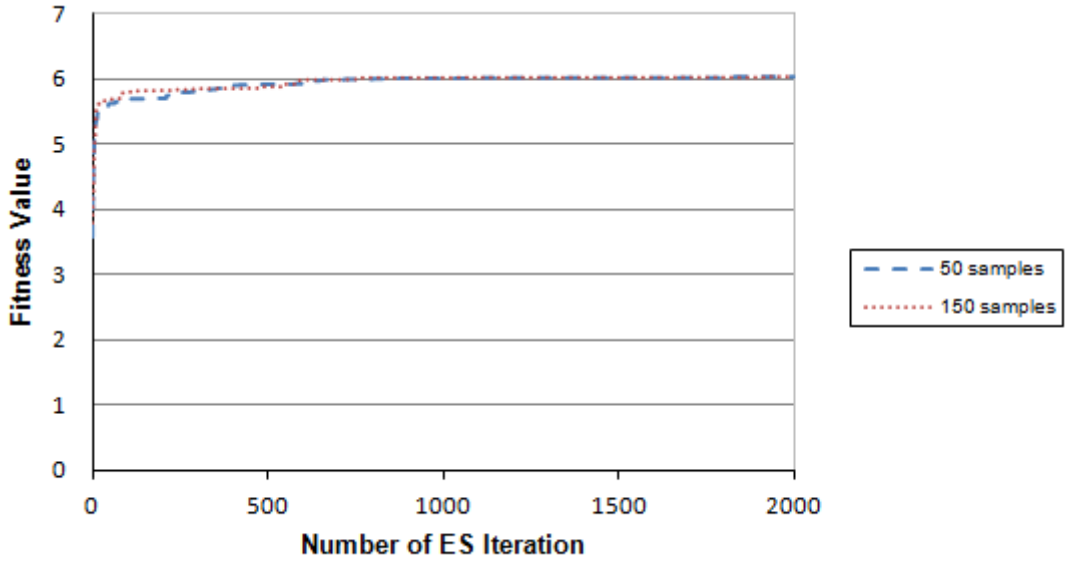

Figure 4.2. Learning curve of ES for 50 and 150 samples.

learning curves are almost the same. Therefore, we decided to calculate the fitness with 50 samples for a better run-time. On the other hand, depending on the characteristic of the problem, more number of samples may be needed.

As mentioned above, *evolution control* is necessary when an approximate model is used for the fitness calculation. On the other hand, it is almost impossible to calculate the exact expected reward for our case, but increasing the number of samples gives a better estimation. Instead of trying to calculate the fitness exactly, we calculated it by getting 1000 samples when needed. We calculated the fitness of all chromosomes in the first generation by getting 1000 samples in order to get a better estimate. After the first generation, if we obtain a chromosome which has a fitness value greater than one of the best chromosomes selected as a result of 50 samples, we re-calculate its fitness more accurately by getting 1000 samples. If its fitness value is still greater, we put it into the selected chromosomes. At the end of the evolution process, we re-calculate the fitness of the best chromosomes selected by getting 100000 samples and report the one having the highest fitness value as the best solution we obtained.

4.2.3. Convergence Criteria

Although our algorithm is an approximate solution technique, we try to converge to the optimal solution. On the other hand, there is no criteria that tell us whether a solution is optimal or not, therefore, deciding when to stop the evolution is a major problem. We initially relied only on the generation count and let the evolution continue for an empirically determined number of generations. However, we observed that improvements in the best fitness usually occur during the initial steps of the evolution and after some number of steps, the diversity in the population drops so that the probability of generating better solutions gets lower even with mutations. By combining these observations, we decided to let 2000 as the upper limit for the number of generations and to observe both the best fitness of the population and the standard deviation of fitnesses. If the best fitness is not improved for 10 successive generations and the standard deviation of the fitnesses has a low value, i.e. 0.01, we stop the evolution. By this way, the evolution usually stops after 50-100 generations. This may lead us to obtain worse values on the average, however, this enables us to make much more runs at the same time and explore the search space more efficiently.

4.2.4. Generation of the Next Generation

We tested several population sizes in our experiments and we populated our initial set with random chromosomes. We select the best ten chromosome and put them in a separate set, i.e. the set of potential solutions. These chromosomes do not take part in the evolution. If a chromosome having a better fitness value appears during the evolution, the worst one is taken out and the new chromosome is put into this set. At the end of the evolution process, the best chromosomes in this set are tested again in order to find the best one more accurately. The best chromosome obtained as such corresponds to the *team policy*. Each agent uses the part that is related with it from this policy, so, the individual agent policies may be different from each other. The general flow of our algorithm can be seen in Figure 4.3.

While creating the next generation, recombination and mutation operators are used.

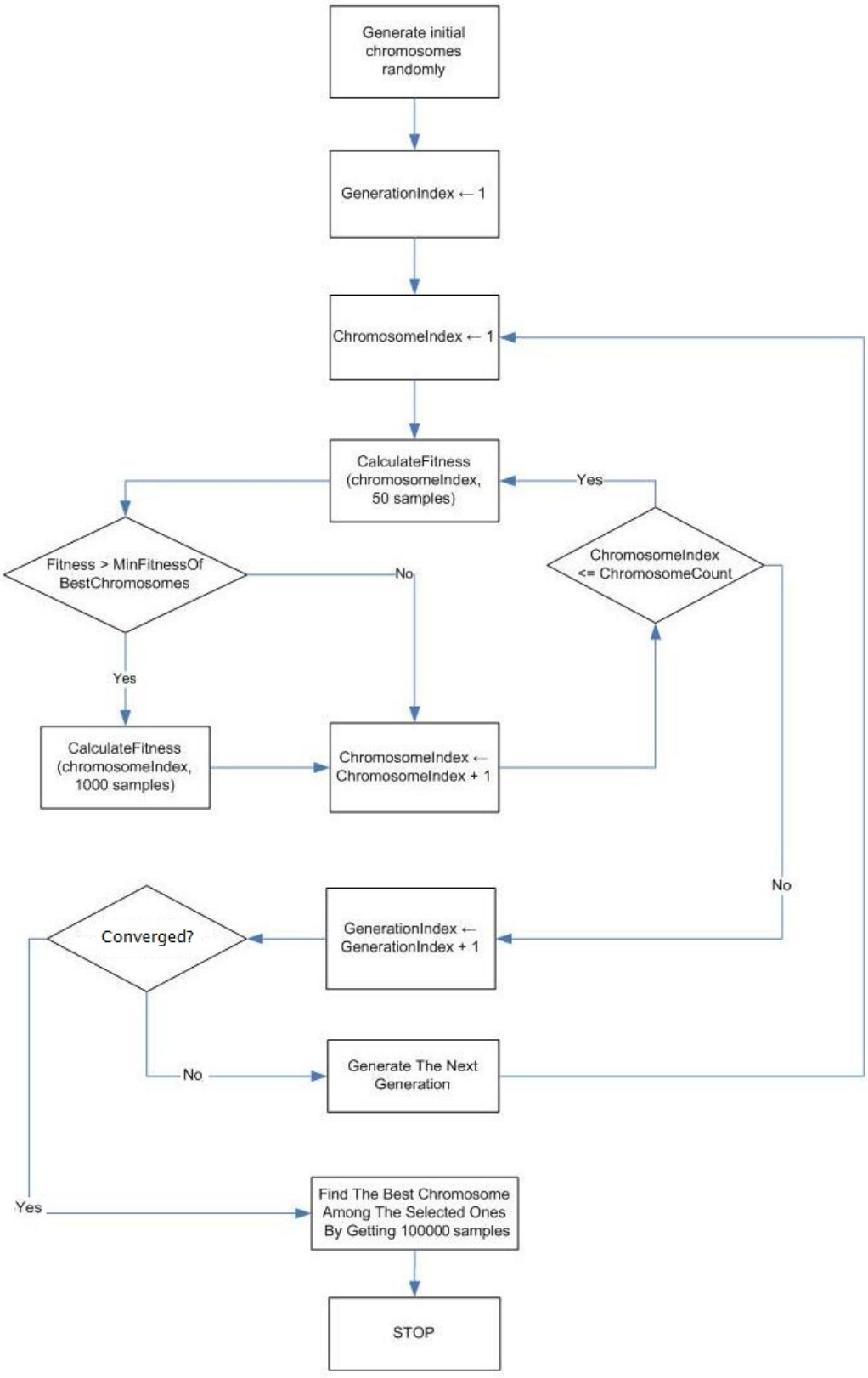

Figure 4.3. The general flowchart of our algorithm.

We used the same mutation operation that we explained in Section 4.1.1. For the recombination case, we used uniform crossover.

4.3. Solving DEC-POMDP Problems Using Belief Vectors

As mentioned above, we make simulations to test the quality of a generated chromosome. During the simulation, each agent keeps its own belief state about the environment. The belief state of an agent is an indication of its belief about the whole environment, not just its state. For example, in the Meeting in a 2×2 Grid problem with two agents, there are 16 environment states. So, each agent holds a belief vector of size 16, indicating its belief about these environment states. They update their belief using the method discussed in Section 3.1.2.

4.3.1. Encoding

POMDP solvers give us, as a solution, a set of vectors each corresponding to a different action. The agents hold a belief state and they multiply their belief vectors with each of these vectors. Then, they select the action that corresponds to the vector giving the highest value as their best action. We anticipated that, we can obtain the values of these vectors by using evolutionary algorithms.

In POMDP solutions, the size of the vector set produced by the solution algorithm is not determined initially. Since we use the same approach for DEC-POMDP problems, we can not know the size of the solution vector set. In general, more than one vector may correspond to the same action. If one of these vectors maximizes the product with the current belief, then the corresponding action of the vector is the best action. On the other hand, there may be no corresponding vector for an action. This means that this action is not optimal in any case. For the sake of simplicity, we assumed that the number of vectors in the solution set is equal to the number of actions. This may hinder our algorithm to reach the optimal solution but it is still possible to find near optimal solutions.

We represent the team policy as a chromosome in our algorithm. This chromosome consists of the individual agent policies and each agent policy in turn consists of several vectors. The number of vectors in an agent policy is equal to the number of actions available to this agent. The size of each vector is equal to the number of states in the environment. For example, in the Multi-access Broadcast Channel problem, there are four environment states and two actions for each agent. The policy of a single agent consists of 2 vectors and the size of these vectors is 4. Therefore, the policy of an agent can be represented with a vector of size $4 \times 2 = 8$. Since we have two agents, the team policy is a vector having size $8 \times 2 = 16$, in other words we have a chromosome with 16 genes. In Figure 4.4, we show a sample chromosome for this problem and how the best action calculation is realized for the corresponding belief vectors.

4.3.2. Complexity Analysis

Since we do not have any guarantee on how quickly the ES algorithm will converge, it is not possible to find the time complexity of the algorithm exactly. However, each generation of the ES algorithm can be analyzed and each can be investigated in two parts.

- *Complexity of generating a candidate population:* In the initial population, it takes polynomial time in terms of the population size, N_p, and the chromosome size. The chromosome size is $n \times N_a \times N_s$, where n is the agent count, N_a is the number of actions available to each agent and N_s is the number of environment states. Therefore, the complexity of this part is $O(N_p \cdot n \cdot N_a \cdot N_s)$. In the later generations, it again takes proportional time, but here, each member of this chromosome is accessed more than once.
- *Complexity of testing the fitness of each chromosome:* This part requires more time and it is determined by the complexity of a simulation step. Each step requires finding the best action and updating the belief state. Finding the best action takes time in terms of $n \times N_a \times N_s$. Belief update part takes time in terms of $n \times N_s^2$. Since the $state\ count$ is greater than $action\ count$ in most cases, belief update part mainly determines the complexity of the fitness calculation. This part is repeated for each

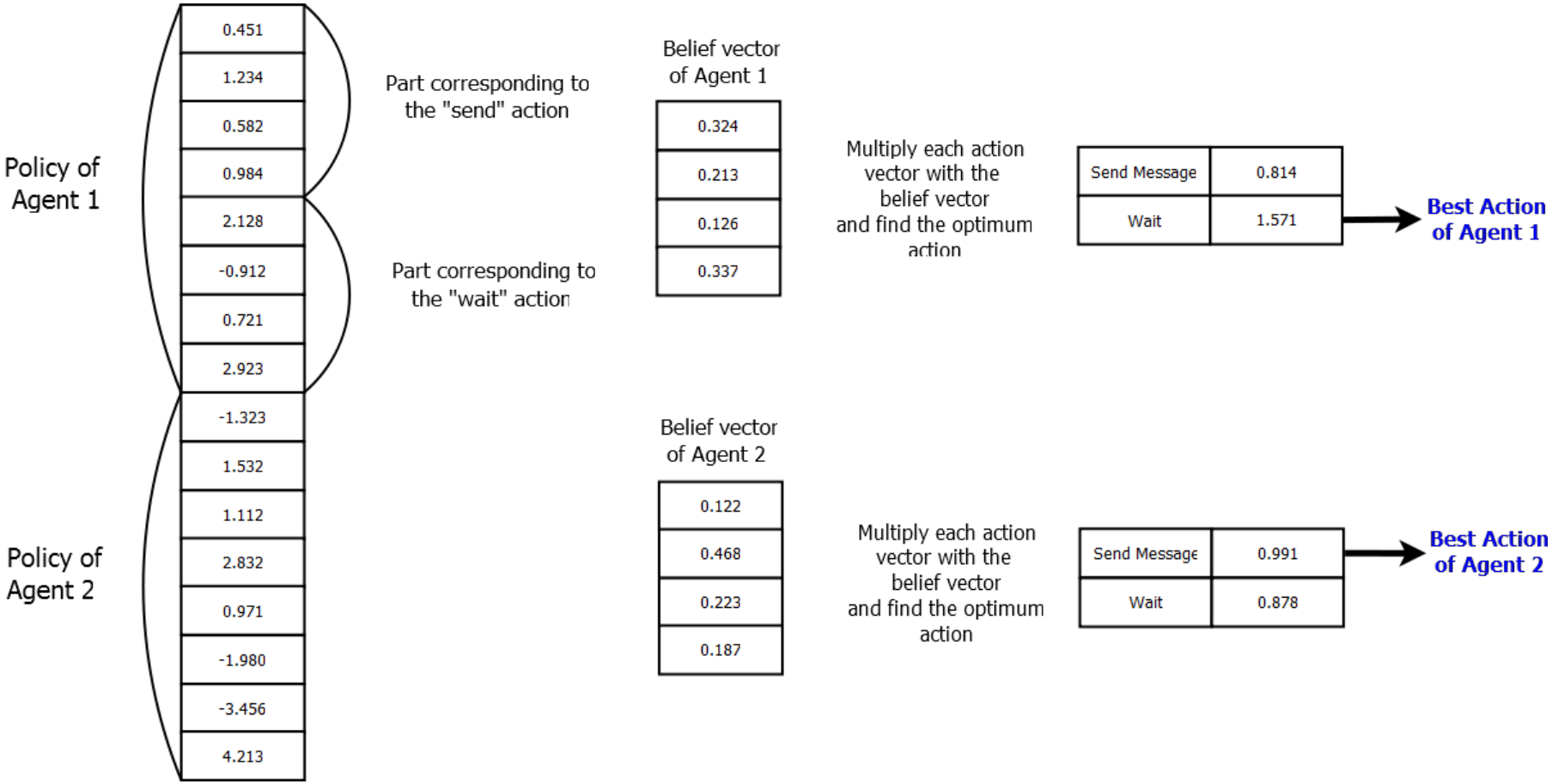

Figure 4.4. An example chromosome for the Multi-access Broadcast Channel problem using the belief vector representation.

chromosome in the population. Since the fitness calculation is the most time consuming part of ES generation for our case, it can be said that each evolution generation takes $O(N_p \cdot h \cdot N_{sc} \cdot n \cdot N_s^2)$ time, where N_p is the population size, h is the test horizon and N_{sc} is the number of simulations made for calculating the average reward.

4.3.3. Experiments and Results

We made our experiments on the problems mentioned in Section 2.2 for several horizons which changes from one problem to another. For each horizon, we run the ES algorithm 10 times and we choose the best policies of each run. Then, for each horizon, we report both the $best\ policy$ among 10 runs and the average reward and standard deviations of the 10 policy obtained as a result of these runs. We print the best results in ***bold*** wherever we are able to find the optimal solution. Since our preliminary concern is the case where communication is not allowed, we made all experiments just for the no-communication case.

4.3.3.1. Determination of Algorithm Parameters. The ES algorithm has several parameters that can affect the results of the runs. These parameters are:

- *Population size:* Increasing the population size may help to search the policy space better at the cost of increasing the run-time. We made our tests with three different population sizes: 40, 80 and 120.
- *Neighborhood distance:* While creating a new generation, i_{th} chromosome is initially created from its neighbors between $i - n_d$ and $i + n_d$ with the help of recombination, where n_d is the neighborhood distance, and then it is mutated. We made our tests with four different values: 2, 4, 8 and 20.
- *Min-max values of genes:* We multiply the belief vector with the policy vectors in order to find the best action. When we normalize these vectors such that all values are between -1 and 1, the choice of the best action does not change. Therefore, if we determine the min-max values as -1 and 1, we can consider all cases.

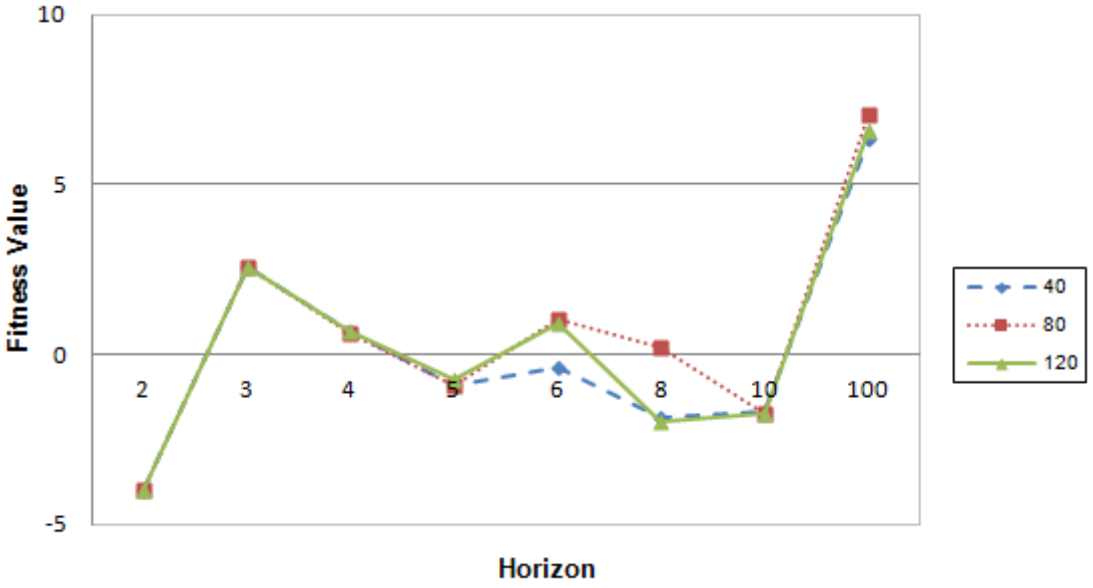

Figure 4.5. The obtained reward values for each horizon for different population sizes using the Belief Vectors and the ES for the DEC-Tiger problem.

We wanted to test the effect of the above mentioned parameters on one of the problems and use the same parameter set in all problems if the difference between results for the different parameters is not significantly different. For this purpose, we considered the DEC-Tiger and the Cooperative Box Pushing problems, since our initial results with these problems were much worse when compared to the other problems. Since, the DEC-Tiger problem is a smaller problem in terms of state count, we decided to use it for extensive tests in order to make faster runs.

4.3.3.2. DEC-Tiger Problem. In order to see the effect of the population size on this problem, we made 10 runs for each horizon by using 3 different population sizes and assuming the neighborhood distance as 8. We report the best results obtained for each case in Figure 4.5 with respect to the horizon. In general, the results are close to each other for all horizons except for horizon 6 and 8. For these horizons, population size 80 outperforms others. Therefore, we decided to choose the population size as 80 for our remaining tests.

After testing the effect of the population size on this problem, we made runs for each horizon by using 4 different neighborhood distances and assuming the population size as

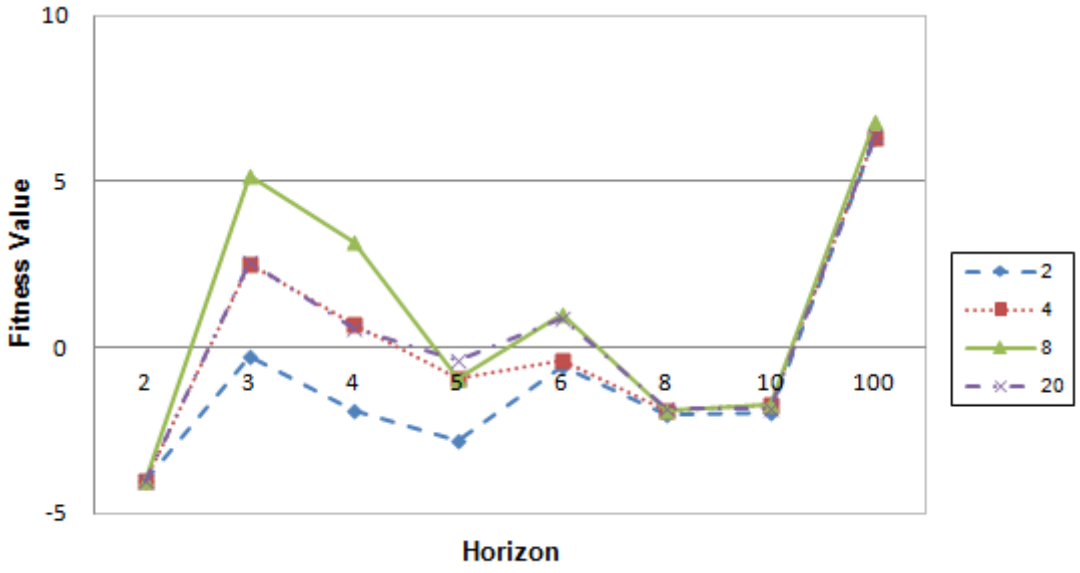

Figure 4.6. The obtained reward values for each horizon for different neighborhood distances using the Belief Vectors and the ES for the DEC-Tiger problem.

80. The obtained results are presented in Figure 4.6. According to the results, increasing the neighborhood distance has a positive effect on the average reward up to 8. When the distance is increased to 20, the results becomes worse for some horizons. For this reason, we set the neighborhood distance parameter as 8 for all the problems.

The summary of our results with population size 80 and neighborhood distance 8 are presented in Table 4.1. We compare our results with the same studies we cited in Section 3.2.1. The results show that our algorithm does not work well with the DEC-Tiger problem as in the MAP algorithm. We use the belief state approach also in the MAP algorithm and the approximate belief updates may not always result in correct belief states. Also, this problem is highly sensitive to wrong actions such that the punishment of opening the wrong door is too high. Therefore, opening the door the tiger is behind due to a wrong belief state affects the average reward seriously. According to our experiments, it seems to be difficult to produce a "successful" strategy for this problem using belief vector approach no matter which method is used to find the belief vectors. Except horizon 2, the results are much worse than the optimal ones using $ES-BV$ approach. On the other hand, the results are better than the MAP approach, since the policies are obtained with no-communication

Table 4.1. Test results for the Dec-Tiger problem using the Belief Vectors and ES.

	APPROXIMATE		OPTIMAL		
Horizon	MBDP	PBIP	GMAA*-ICE	ES-BV	
	[27]	[22]	[16]	Best	Average
2	-4.00	-	-4.00	-4.00	**-4.00** $\mp$ 0.00
3	5.19	-	5.19	2.54	1.93 $\mp$ 1.24
4	4.80	-	4.80	0.61	0.54 $\mp$ 0.07
5	5.38	-	7.03	-0.92	-1.71 $\mp$ 1.00
6	9.91	-	10.38	1.05	0.07 $\mp$ 0.81
8	9.42	-	X	0.20	-1.44 $\mp$ 0.92
10	13.49	13.6	X	-1.76	-1.84 $\mp$ 0.05
100	93.24	147	X	7.04	6.76 $\mp$ 0.40

assumption.

4.3.3.3. Multi-access Broadcast Channel Problem. The summary of all of our results are presented in Table 4.2. When compared to the DEC-Tiger problem, the results are more acceptable, however they are still worse than the optimal values for all the horizons as in the MAP approach. On the other hand, while the GMAA*-ICE algorithm is able to handle this problem up to horizon 900, we are able to generate a solution up to horizon 1000 as MBDP method and we can handle much larger horizons.

4.3.3.4. Meeting in a Grid Problem. The summary of all of our results are presented in Table 4.3. In general, we obtain comparable results with the optimal ones up to horizon 6 and this is the largest horizon reported for this problem so far. Our algorithm is able to handle larger horizons and as an example we report a solution for horizon 50. Producing the strategy with no-communication assumption helps us to obtain much better rewards when compared to MAP approach.

Table 4.2. Test results for the Multi-access Broadcast Channel problem using the Belief Vectors and ES.

	APPROXIMATE		OPTIMAL		
Horizon	MBDP	PBIP	GMAA*-ICE	ES-BV	
	[27]	[22]	[16]	Best	Average
2	2.00	2.00	-	1.90	1.90 ∓ 0.00
4	3.89	3.89	-	3.70	3.70 ∓ 0.00
8	7.49	7.49	-	7.30	7.30 ∓ 0.00
10	9.29	9.29	-	9.11	9.11 ∓ 0.00
20	18.29	18.31	18.31	18.11	18.11 ∓ 0.00
50	45.29	X	45.50	45.12	45.12 ∓ 0.00
100	90.29	X	90.76	90.12	90.11 ∓ 0.01
250	-	X	226.50	225.11	225.11 ∓ 0.00
500	-	X	452.74	450.13	450.13 ∓ 0.01
900	-	X	814.71	810.21	810.21 ∓ 0.00
1000	900.29	X	X	899.99	899.99 ∓ 0.00

Table 4.3. Test results for the Meeting in a Grid problem using the Belief Vectors and ES.

	APPROXIMATE	OPTIMAL		
Horizon	Cross Entropy	GMAA*-ICE	ES-BV	
	[25]	[16]	Best	Average
2	0.91	0.91	**0.91**	0.91 ∓ 0.00
3	1.55	1.55	1.53	1.53 ∓ 0.00
4	2.24	2.24	2.19	2.18 ∓ 0.00
5	2.96	2.97	2.87	2.86 ∓ 0.01
6	3.64	3.72	3.59	3.58 ∓ 0.01
50	X	X	37.91	37.24 ∓ 1.33

Table 4.4. Test Results for the Recycling Robots Problem using Belief Vectors and ES.

	OPTIMAL		
Horizon	GMAA*-ICE	ES-BV	
	[16]	Best	Average
5	16.49	16.11	16.11 $\mp$ 0.00
15	47.25	46.89	46.88 $\mp$ 0.00
18	56.48	56.13	56.12 $\mp$ 0.01
20	62.63	62.27	62.26 $\mp$ 0.01
30	93.40	93.03	93.02 $\mp$ 0.01
40	124.17	123.81	123.81 $\mp$ 0.00
50	154.94	154.61	154.59 $\mp$ 0.03
60	185.71	185.36	185.36 $\mp$ 0.00
70	216.48	216.12	216.12 $\mp$ 0.00
100	X	308.46	308.45 $\mp$ 0.01

4.3.3.5. Recycling Robots Problem. The summary of all of our results are presented in Table 4.4. The optimal results for this problem is calculated up to horizon 70 and our results are very close to the optimal for all the horizons. Our algorithm is able to handle larger horizons and we report a solution for horizon 100 as an example. As in the Meeting in a Grid Problem, we see the benefit of the new method for searching the policy.

4.3.3.6. Cooperative Box Pushing Problem. As we stated in Section 3.2.5, our belief update procedure is just an approximation and this resulted in invalid belief states for this problem. Due to this reason, we could not obtain successful solutions for this problem with the belief update method. The obtained results are given in Table 4.5. The results are same with the MAP approach, because the agents stay in the same belief state always.

4.3.3.7. Fire Fighting Problem. The obtained results are given in Table 4.6. $ES - BV$ approach gives acceptable results for small sized problems and it improves the results of the

Table 4.5. Test results for the Cooperative Box Pushing problem using the Belief Vectors and ES.

	APPROXIMATE	OPTIMAL		
Horizon	IMBDP	GMAA*-ICE	ES-BV	
	[28]	[16]	Best	Average
2	17.60	17.60	-0.40	-0.40 $\mp$ 0.00
3	65.68	66.08	-0.60	-0.60 $\mp$ 0.00
4	97.91	98.59	-0.80	-0.80 $\mp$ 0.00
8	182.92	X	-1.60	-1.60 $\mp$ 0.00
10	189.32	X	-2.00	-2.00 $\mp$ 0.00
20	415.25	X	-4.00	-4.00 $\mp$ 0.00
50	1051.82	X	-10.00	-10.00 $\mp$ 0.00
100	2112.05	X	-20.00	-20.00 $\mp$ 0.00

MAP approach. With the approximate belief update, making the belief estimate correctly becomes more difficult with the increasing state size and having wrong belief values causes to wrong actions. For this reason, the results are worse than the optimal results for all horizons with this problem.

4.4. Solving DEC-POMDP Problems Using Observation History

Since the belief update method we used is just an approximation, it may lead to misleading belief states in some cases. The belief state is the summary of the previous observations and actions. Therefore, using the observation history for decision making is an alternative to using the belief state approach. Mazurowski encoded all possible observation sequences up to a given horizon into a chromosome [1]. Each gene corresponds to a different observation sequence and the value of that gene is the best action for this observation history. An example encoding for a problem having 2 agents and 2 observations for horizon 2 can be seen in Figure 4.7. The state and the action count do not affect the chromosome length for this encoding. According to the example encoding, if $agent\ 1$ gets

Table 4.6. Test results for the Fire Fighting Problem using the Belief Vectors and ES.

	OPTIMAL		
Horizon	GMAA*-ICE	ES-BV	
	[16]	Best	Average
2	-4.38	-5.48	-5.52 ∓ 0.05
3	-5.74	-7.38	-7.50 ∓ 0.17
4	-6.58	-8.76	-8.92 ∓ 0.23
5	-7.07	-10.33	-10.66 ∓ 0.47
6	-7.18	-12.01	-12.45 ∓ 0.62

observation o_1 at time step 1, its best action is 3. If it gets observation o_1 in the next time step, its best action is 2, since the observation sequence becomes $o_1 o_1$ and the best action for this sequence for the agent 1 is 2. Being independent of the state and the action count is an advantage of this encoding, however the chromosome size increases exponentially with the horizon size and after a few horizons, the length of the chromosomes easily reaches to over millions of genes. Therefore, we can see that this encoding can only be used for very small horizons.

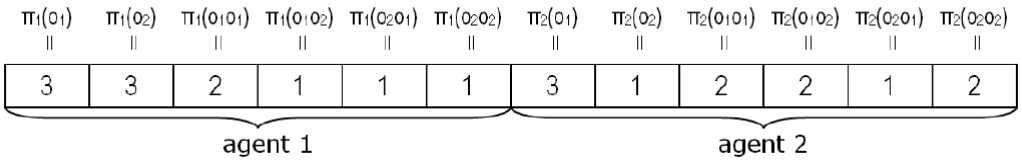

Figure 4.7. An example chromosome encoding using observation history [1].

Instead of keeping the observation sequences in a chromosome, we use a neural network that gets the observation sequence as an input. Also we prefer to use only the last n observations for decision making, because we think that the effect of the previous observations to the decision can be omitted and by this way our encoding does not depend on the horizon length. In our belief vector approach, we have a vector for each action. In this case, however, we have a different neural network for each action and each agent. Therefore our chromosome size depends on the observation, the action and the agent count.

There are several methods to train neural networks and the most commonly used one is the backpropagation. This technique is a supervised learning technique and it requires a set of example input and outputs. Since there is no such set for our problems, we can not use this technique. For this purpose, we used ES to learn the weights of the neural networks. The weights of the neural networks are mapped to a chromosome and the fitness of that chromosome is calculated by using our approximate fitness calculation method.

4.4.1. Neural Network Architecture

If an agent has m possible observations and we are using the last n observations for decision making, we form a neural network having $n \times m$ input nodes. Each of the last n observations is coded with a node group having m nodes. We also order the possible observations from 1 to m. If the observation corresponding to a node group is j, where $1 \leq j \leq m$, the $node\ j$ is given the value 1 and the other nodes are given the value 0. An example mapping can be seen in Figure 4.8. In this example, we assume that the agent has two possible observations and it uses last three observations to make a decision. If the last three observations, $O(t-2, t-1, t)$ is, $O_1, O_2\ and\ O_1$, the input of the neural network should be like that is shown in the figure.

We use a neural network having an input layer, a hidden layer and an output layer. The number of input nodes depends on the observation set of the agent and the length of the observation history. The number of hidden layer nodes is determined empirically. We have one output node. If the number of actions available to an agent is A, this agent uses A such neural networks for decision making and each neural network corresponds to a different action. To make a decision, an agent feeds the current observation history to all the networks and gets the outputs from them. It then selects the action corresponding to the network that gives the highest output.

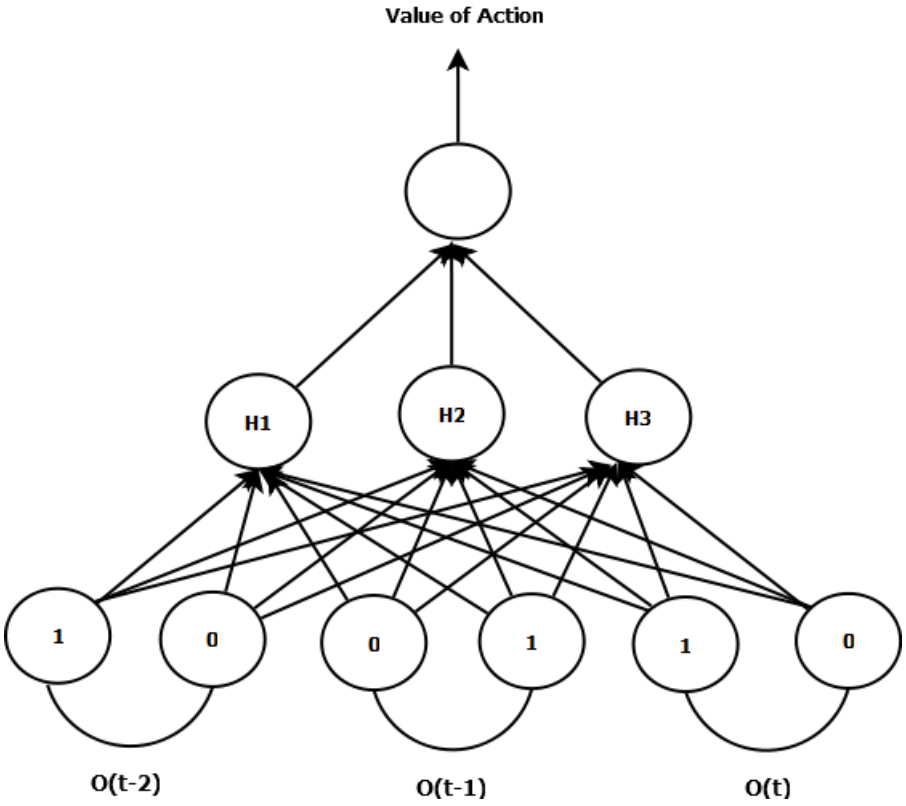

Figure 4.8. An example of mapping an observation history to input nodes.

4.4.2. Complexity Analysis

Similar to the belief vector approach, we analyze each generation of the ES algorithm and this can be investigated in two parts:

- *Complexity of generating a candidate population:* In the initial population, it takes polynomial time in terms of the population size and the chromosome size. The neural networks used for decision making are encoded in the chromosomes and the number of weights in a neural network is $(N_o \times N_{oh} + 1) \times N_{hid}$, where N_o is the number of observations available to an agent, N_{oh} is the observation history size and N_{hid} is the number of neurons in the hidden layer. Therefore, the chromosome size is $n \times N_a \times (N_o \times N_{oh} + 1) \times N_{hid}$. The generation of a candidate population takes proportional time to this number.
- *Complexity of testing the fitness of each chromosome:* This part requires more time and it is determined by the complexity of a simulation step. Each step requires finding

the best action and updating the observation history. Finding the best action for each agent depends on the complexity of neural network output calculation and the action count. The output calculation is polynomial in terms of the number of weights in the neural network. Updating the observation history takes constant time; the oldest observation is removed from the observation list and the new is inserted. Therefore, the complexity of the first and the second part of the ES iteration is the same in terms of complexity. As a result, each generation takes $O(N_p \cdot h \cdot N_{sc} \cdot n \cdot N_a \cdot N_o \cdot N_{oh} \cdot N_{hid})$ time.

4.4.3. Experiments and Results

In order to compare the results with the belief vector method, we solved the same problems. Similarly, we run 10 different ES learning and obtained 10 different strategy for each case. We obtained the average rewards of these strategies by making 100000 simulations. We reported the average and the standard deviations of the 10 best results that we obtained for each horizon and also the best of them.

4.4.3.1. Determination of Algorithm Parameters. In addition to the ES parameters that we tested in the belief vector method, there are two other parameters that are specific to this method:

- *Observation history size:* The number of recent observations used for decision making is important. Increasing it may help to make better decisions. On the other hand increasing it will also result in larger neural networks that will increase the run-time. For this reason, we should use the least number of observations possible to make a healthy decision. We tested 3 different history sizes for this purpose: 1, 3 and 5.
- *Neural network hidden node count:* Increasing the hidden node count may increase the expressiveness of the neural networks at the cost of increasing the run-time. We made our tests with three different node counts: 1, 3 and 5.

We wanted to test the effect of these parameters on the DEC-Tiger problem. After determining the most suitable parameter set in general, we used the same parameter set in all parameters.

4.4.3.2. DEC-Tiger Problem. In order to see the effect of the population size on this problem, we made runs for each horizon by using three different population sizes and assuming the neighborhood distance as 8. We set the observation history size and the hidden node count as 5. We report the best results obtained for each case in Figure 4.9 with respect to the horizon. The results are very close to each other for all horizons, however, the population size 80 slightly outperforms the others for some horizons, therefore we decided to use a population size of 80 as in the belief vector case.

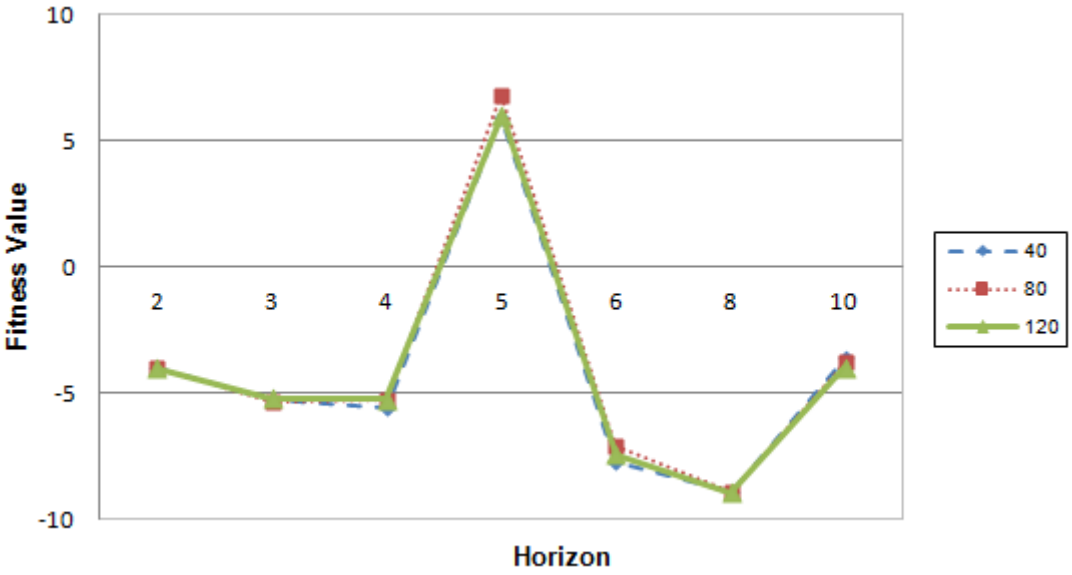

Figure 4.9. The obtained reward values for each horizon for different population sizes using the Neural Networks and the ES for the DEC-Tiger problem.

After testing for the effect of the population size on this problem, we made runs for each horizon by using four different neighborhood distances and setting the population size as 80. We set the observation history size and the hidden node count as 5. The obtained results are presented in Figure 4.10. In general, the results are close to each other, however the neighborhood distance 8 slightly outperforms the others for some horizons. For this reason, we set this parameter as 8 for all other problems.

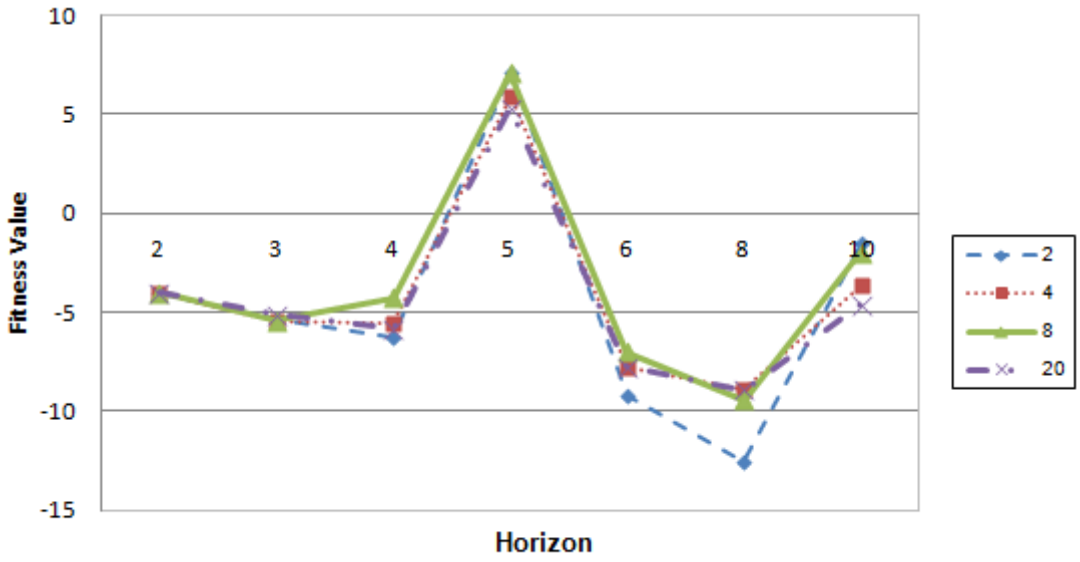

Figure 4.10. The obtained reward values for each horizon for different neighborhood distances using the Neural Networks and the ES for the DEC-Tiger problem.

One of the parameters that affects the neural network structure is the node count in the hidden layer. By holding other parameters constant, we made experiments with three node counts. As we expect, we obtain better results with the increasing node count and the best results are obtained with 5 nodes as can be seen in Figure 4.11. On the other hand, the results with 3 nodes and 5 nodes are very similar. Therefore, we did not try larger hidden node counts and put 5 nodes to the hidden layer of our neural networks for all the problems.

Another parameter that affects the neural network is the size of the observation history. We made experiments with three observation history sizes and present our results in Figure 4.12. For small horizons, the obtained results are almost the same. However, for larger horizons, best results are obtained by using the last 5 observations for decision making. For this reason, we set the observation history size as 5.

After these tests, we determined the parameter values and used the same parameters for all other problems. The summary of all of our results for the DEC-Tiger problem is presented in Table 4.7. We compare our results with the same studies that we referred in Section 3.2.1. We obtain the optimal results for horizon 2 and 5, however, the results for

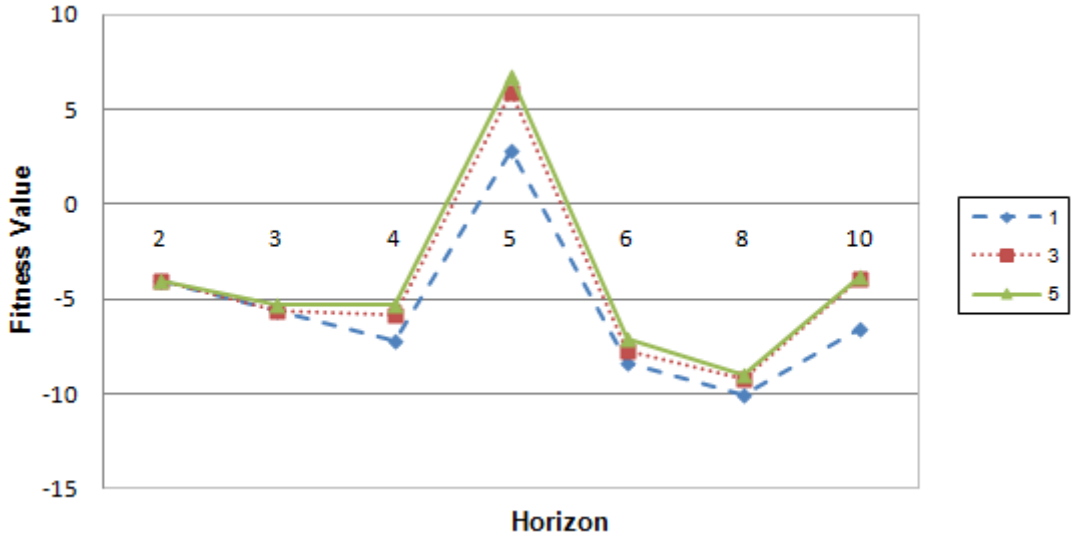

Figure 4.11. The obtained reward values for each horizon for different hidden node counts using the Neural Networks and the ES for the DEC-Tiger problem.

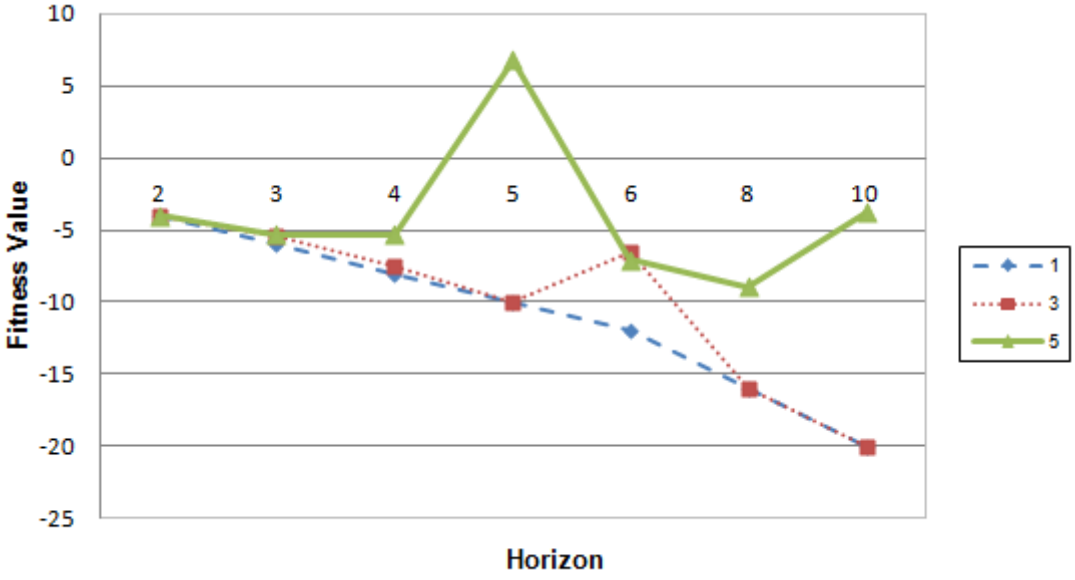

Figure 4.12. The obtained reward values for each horizon for different observation history sizes using the Neural Networks and the ES for the DEC-Tiger problem.

other horizons are much worse than the previously reported results. Obtaining the optimal value for horizon 5 shows the power of our method, however, we may need to make more runs to obtain better results for other horizons, due to the randomness in the algorithm and the difficulty of this problem.

Table 4.7. Test results for the Dec-Tiger problem using Observation History and ES.

	APPROXIMATE		OPTIMAL		
Horizon	MBDP	PBIP	GMAA*-ICE	ES-OH	
	[27]	[22]	[16]	Best	Average
2	-4.00	-	-4.00	**-4.00**	-4.00 $\mp$ 0.00
3	5.19	-	5.19	-5.41	-5.62 $\mp$ 0.25
4	4.80	-	4.80	-4.72	-5.92 $\mp$ 0.82
5	5.38	-	7.03	**7.03**	4.88 $\mp$ 1.86
6	9.91	-	10.38	-7.01	-9.18 $\mp$ 1.48
8	9.42	-	X	-9.45	-10.22 $\mp$ 1.32
10	13.49	13.6	X	-1.99	-2.46 $\mp$ 0.49
100	93.24	147	X	-60.64	-74.66 $\mp$ 8.74

4.4.3.3. Multi-access Broadcast Channel Problem. The summary of all of our results are presented in Table 4.8. As in the previous methods, the results with this problem are worse than the optimal values, however the difference is acceptable for all the horizons. The average values are the same with best values and this shows that the algorithm consistently finds similar policies. Also, we are able to produce a solution for horizon 1000 as in the belief vector method.

4.4.3.4. Meeting in a Grid Problem. The summary of all of our results are presented in Table 4.9. We are able to obtain the optimal results for horizon 2 and horizon 6. However, we obtain $10 - 20\%$ worse results than the optimal ones in general with low standard deviations as in the previous problem. 6 is the largest horizon solved in the literature for this problem and the advantage of our algorithm is being able to handle larger horizons and

Table 4.8. Test results for the Multi-access Broadcast Channel problem using Observation History and ES.

	APPROX.	OPTIMAL			
Horizon	MBDP	Loss. Clust.	GMAA*-ICE	ES-OH	
	[27]	[15]	[16]	Best	Average
2	2.00	2.00	-	1.90	1.90 ∓ 0.00
4	3.89	3.89	-	3.70	3.70 ∓ 0.00
8	7.49	7.49	-	7.31	7.30 ∓ 0.00
10	9.29	9.29	-	9.11	9.10 ∓ 0.00
20	18.29	18.31	18.31	18.11	18.11 ∓ 0.00
50	45.29	X	45.50	45.11	45.11 ∓ 0.00
100	90.29	X	90.76	90.11	90.11 ∓ 0.00
250	-	X	226.50	225.12	225.11 ∓ 0.00
500	-	X	452.74	450.13	450.13 ∓ 0.00
900	-	X	814.71	810.23	810.23 ∓ 0.00
1000	900.20	X	X	899.99	899.99 ∓ 0.00

as an example we report a solution for horizon 50.

4.4.3.5. Recycling Robots Problem. The summary of all of our results are presented in Table 4.10 and our results are very close to the optimal for all the horizons up to 70 and the results have a low standard deviation for all the cases. Our algorithm is able to handle larger horizons and we report a solution for horizon 100 as an example.

4.4.3.6. Cooperative Box Pushing Problem. The results that we obtained for this problem are much better than belief vector approach, however they are still far from the optimal results up to horizon 4. When compared to IMBDP approach, we obtain better results for horizon 10 and 20 in terms of best reward, however our best results are worse than this approach for other horizons. Also, the standard deviation of the results gets considerably

Table 4.9. Test results for the Meeting in a Grid problem using Observation History and ES.

	APPROXIMATE	OPTIMAL		
Horizon	Cross Entropy	GMAA*-ICE	ES-OH	
	[25]	[16]	Best	Average
2	0.91	0.91	0.87	0.86 ∓ 0.00
3	1.55	1.55	**1.55**	1.55 ∓ 0.00
4	2.24	2.24	2.00	1.99 ∓ 0.01
5	2.96	2.97	2.64	2.63 ∓ 0.01
6	3.64	3.72	**3.72**	3.71 ∓ 0.00
50	X	X	38.79	38.62 ∓ 0.21

larger with the growing horizon and our results becomes much worse than the IMBDP approach in terms of the average reward. The obtained results are given in Table 4.11.

4.4.3.7. Fire Fighting Problem. The obtained results are given in Table 4.12. The optimal results for this problem are known up to horizon 6 and we obtain the optimal result for horizon 2. When compared to the previous approaches, we obtain much better results for all the horizons and our results are close to the optimal ones in general. As in the small sized problems, the standard deviation with this problem is very low. The obtained results with the Cooperative Box Pushing problem and the Fire Fighting problem show that using the observation history for decision making instead of belief vectors help us to obtain better results even for the large problems.

Table 4.10. Test results for the Recycling Robots problem using Observation History and ES.

	OPTIMAL		
Horizon	GMAA*-ICE	ES-OH	
	[16]	Best	Average
5	16.49	15.39	15.39 $\mp$ 0.00
15	47.25	46.17	46.17 $\mp$ 0.01
18	56.48	55.41	55.40 $\mp$ 0.01
20	62.63	61.57	61.56 $\mp$ 0.01
30	93.40	93.06	93.05 $\mp$ 0.01
40	124.17	123.82	123.81 $\mp$ 0.01
50	154.94	154.59	154.58 $\mp$ 0.01
60	185.71	185.36	185.35 $\mp$ 0.01
70	216.48	216.14	216.12 $\mp$ 0.02
100	X	308.44	308.42 $\mp$ 0.02

Table 4.11. Test results for the Cooperative Box Pushing problem using Observation History and ES.

	APPROXIMATE	OPTIMAL		
Horizon	IMBDP	GMAA*-ICE	ES-OH	
	[28]	[16]	Best	Average
2	17.60	17.60	17.18	17.15 $\mp$ 0.04
3	65.68	66.08	22.96	22.91 $\mp$ 0.06
4	97.91	98.59	90.17	90.11 $\mp$ 0.08
8	182.92	X	174.68	173.76 $\mp$ 1.30
10	189.32	X	210.25	209.07 $\mp$ 0.25
20	415.25	X	440.35	361.27 $\mp$ 185.458
50	1051.82	X	902.21	702.29 $\mp$ 346.27
100	2112.05	X	1908.34	1472.68 $\mp$ 754.59

Table 4.12. Test results for the Fire Fighting problem using Observation History and ES.

	OPTIMAL		
Horizon	GMAA*-ICE	ES-OH	
	[16]	Best	Average
2	-4.38	**-4.38**	-4.43 $\mp$ 0.03
3	-5.74	-5.89	-5.90 $\mp$ 0.02
4	-6.58	-6.73	-6.84 $\mp$ 0.09
5	-7.07	-7.14	-7.18 $\mp$ 0.03
6	-7.18	-7.69	-7.81 $\mp$ 0.09

5. USING GENETIC ALGORITHMS TO SOLVE DEC-POMDP PROBLEMS

5.1. Genetic Algorithms

Like evolution strategies, genetic algorithms is also a member of evolutionary algorithms and it is very similar to evolution strategies. However, there are some small differences between them. In evolution strategies, the chromosomes are encoded with real numbers and it is known to work better for such cases. In genetic algorithms, the chromosomes are encoded in binary as strings of 0s and 1s traditionally, however other encodings are also possible. In evolution strategies, the mutation is controlled with another vector and it is also mutated. In genetic algorithm, there is no such vector. The mutation is made depending on the encoding. There are three main operations in genetic algorithms.

5.1.1. Crossover (Recombination)

Crossover is the process of combining the genetic materials of two parent chromosomes in order to produce new children chromosomes that have the characteristics of both of their parents. In the first phase of the process, two parent chromosomes are selected from the population. There are many possible methods to select them, such as randomly or selecting them with a probability weighted with their fitness, roulette wheel selection. The parents are next recombined to generate the children chromosomes. There are also several techniques for this operation such as *single point* crossover, *two-point* crossover and *uniform* crossover. In single point crossover, a point on the chromosomes is chosen randomly and the chromosomes exchange their content beyond that point and generate two new chromosomes. In two-point crossover, two points are randomly chosen and the content between these two points are exchanged and two new children are produced. In uniform crossover, a bitmap vector is generated randomly at the beginning and the elements of the vector determines whether there will be an exchange at that point or not.

5.1.2. Mutation

The mutation operator is applied to a single chromosome and its main aim is to maintain diversity in the generation. With the use of the crossover operator, the population may reach a local minimum by only exploring a small part of the search space. The mutation operator helps to explore unvisited parts of the search space with random modifications on the chromosomes. The mutation probability in the genetic algorithm determines how frequently a mutation occurs. The mutation implies a random change and it occurs according to the content of the chromosome. For example, if the chromosome contains real numbers, the values may be mutated by adding a random Gaussian variable or the value may be regenerated randomly. One important point to note is that the value obtained with the mutation operation should obey the constraints of the chromosome values whenever such constraints exist. As an example, a problem may require the chromosome values to be in between some limits and the mutated values should not exceed these limits.

5.1.3. Selection

The size of a population in genetic algorithms usually remains constant. On the other hand, with the help of the crossover and mutation, new chromosomes are generated and adding all of them to the population results in the population size exceeding the limits. Therefore, there should be a mechanism to select the chromosomes that will be placed in the next generation. Selecting the best n chromosomes may help to converge faster, however, this may reduce diversity. On the other hand, losing a chromosome having a good fitness value is not a good idea. Therefore, while some of the next generation chromosomes should be chosen from the best ones, all remaining chromosomes should have a nonzero chance to be selected.

5.1.4. Encoding

The chromosomes that are the candidate solutions to the optimization problem are typically represented as strings. For the mapping, a problem dependent encoding mech-

anism is used. In the original genetic algorithm, the mapping is a fixed length binary string. However, in the most general case both discrete and continuous variables can be represented as genes.

5.2. Proposed Approach (GA-FSC)

We initially represented the knowledge of the agents about the environment using belief vectors. Although belief vectors are successfully used in POMDP studies, due to the problems in updating the belief in DEC-POMDP environments, using them does not give effective results. For this reason, we tried using the observation history for making a decision. Although this helps us to obtain better results in general, the results do not get close to the optimal for some cases. Due to this reason, we considered using policy trees and finite state controllers to represent the agent policies. As mentioned in Section 2.3.3, FSCs are used in many infinite horizon DEC-POMDP studies to represent the policy. They are more compact structures than policy trees. While the size of the policy tree grows exponentially with the increasing horizon, the size of the FSC is constant. When compared to belief vectors and observation histories, these two approaches are just approximations, however FSCs are as expressive as policy trees. Also, the run-time performance of FSCs is expected to be much better, because finding the best action and updating the current FSC node according to the current observation is done in constant time. This means that if FSCs are used with approximate fitness calculation, we expect the algorithm to run faster. For all these reasons, we decided to represent agent policies using FSCs.

For an agent, if the size of the FSC is N_n and there are N_o observations and N_a actions in the DEC-POMDP problem, the number of possible FSC is ${N_a}^{N_n} \times {N_n}^{(N_n \times N_o)}$. As an example, even for a small problem having 3 actions and 2 observation, there are more than 2×10^9 different FSCs having 5 nodes. If there are 2 agents in the environment, the number of possible team policies is about 4×10^{18} . This shows that the policy search space is very huge even for such a problem and in general a solution based on complete enumeration is infeasible. For this reason, genetic algorithms known to handle such huge search spaces well are potential candidates.

In addition to the genetic algorithm parameters like crossover type and mutation rate, there are three important decisions that should be made in order to apply genetic algorithms:

(i) *Encoding strategy:* Since we decide to use FSCs to represent our policies, we should find a way to encode FSCs in a chromosome.

(ii) *Fitness calculation method:* Given a DEC-POMDP policy encoded in a chromosome, we should be able to calculate its expected reward or its fitness.

(iii) *Convergence criteria:* During the progress of a genetic algorithm run, many successive generations are produced. There should be some criteria to decide on whether the evolution should stop at the current generation or new generations should be generated. We use the same convergence criteria that we used in the ES algorithm that is mentioned in Section 4.2.3.

We introduce how we realize these in the following sections.

5.2.1. Encoding

In our approach, the agent policies are stored in FSCs. To represent the FSC of an agent as a chromosome, we need to store the node-action mappings and node transitions of the FSC. Since, each agent may have a different policy, we need to store different FSCs for each agent. If there are n agents, our chromosome should have n parts and there should be an FSC encoded in each part. In general, the FSC size for each agent can be different, however, for the sake of simplicity, we assume that the FSC sizes are the same for each agent. The length of the chromosome part that corresponds to an agent FSC is determined by the FSC node count, c, and the number of observations available to that agent. If we consider a part, the first c genes hold the node-action mappings and the values of these genes should be between $1\ and\ m$, where m is the number of actions available to the corresponding agent. Then, for each FSC node N_i and possible observation o_k combination, we have a gene and the value of this gene gives the next node N_j, if the observation o_k is received in the node N_i. The mapping for the FSC given in Figure 2.4

can be seen in Figure 5.1; the action and transition mappings for node N_2 are pointed out as an example.

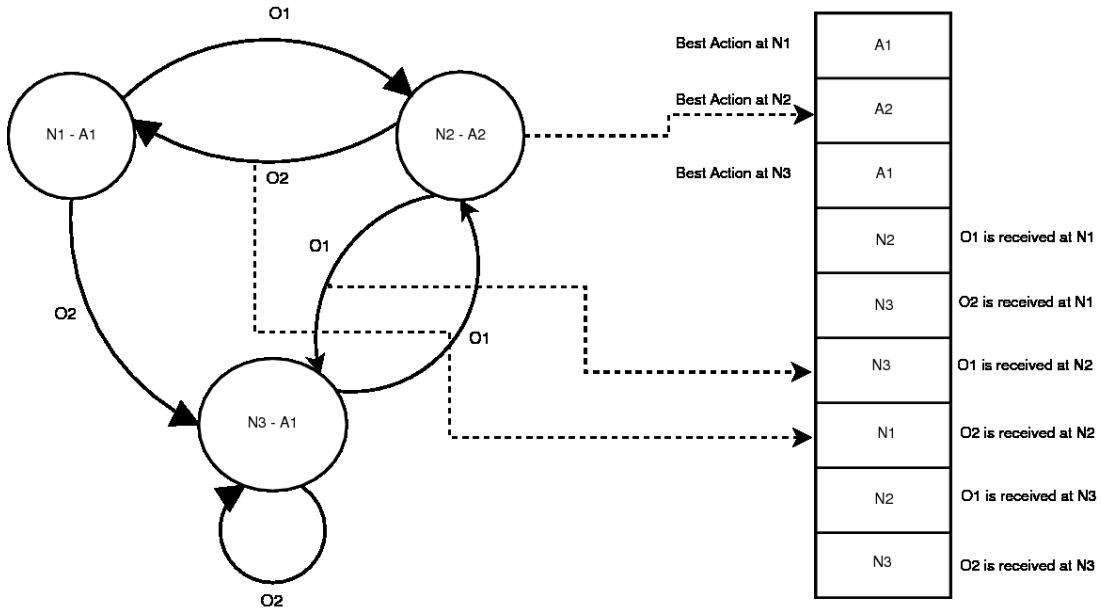

Figure 5.1. Encoding of the given FSC.

5.2.2. Fitness Calculation

The fitness of a chromosome for DEC-POMDP problems is the expected reward of the corresponding policy. We previously calculated expected reward approximately and it is mentioned in Section 4.2.2. If the policy is represented with FSCs it is possible to calculate the expected reward exactly.

5.2.2.1. Exact Fitness Calculation. The equation for calculating the expected reward depends on the structure used for representing the policy. For the infinite horizon case, the expected reward calculation equation is given in [43] for a 2-agent problem using the FSC structure. This equation can easily be adapted for the finite horizon case as follows where the horizon is t:

$$V_t(q_1, q_2, s) = \sum_{a_1,a_2} P(a_1|q_1)P(a_2|q_2)[R(s,a) +$$

$$\sum_{s'} P(s'|a,s) \sum_{o_1,o_2} Obs(o_1,o_2|s',a) \sum_{q_1',q_2'} P(q_1'|q_1,a_1,o_1)P(q_2'|q_2,a_2,o_2)V_{t-1}(q_1',q_2',s')] \tag{5.1}$$

Here, the expected reward is calculated for a given starting state s and joint action $a = (a_1, a_2)$. It is assumed that the initial node of the $agent_1$'s FSC is q_1 and the initial node of the $agent_2$'s FSC is q_2. This equation is general in the sense that it can also be applied for the stochastic policies. Since, we use deterministic policies, $P(a_1|q_1)$, meaning that the probability of a_1 being the best action of the $agent_1$ in the FSC node q_1, is 1 only for one action for q_1 and 0 for all other actions. Similarly, the transition probability $P(q_1'|q_1, a_1, o_1)$ in $agent_1$'s FSC is 1 only for one observation and 0 for all other observations, while a_1 has no effect on node transition in FSC.

If there are more than one possible starting states in the problem, the expected reward for all such states is calculated using the following equation:

$$V_t(q_1, q_2) = \sum_{s'} P(s')V_t(q_1, q_2, s') \tag{5.2}$$

where $P(s')$ is the probability of s' being the starting state.

For a given DEC-POMDP problem, a given strategy and a given horizon, the expected reward can be calculated using Equation 5.2 starting from V_0 with a bottom-up approach. This method is easy to implement and efficient in terms of memory usage because while calculating all V_t values, only V_{t-1} values need to be in the memory; V values for smaller horizons may be released. On the other hand, our final aim is to calculate only the $V_t(s, q_1, q_2)$ value, and we may not need all V_{t-1} values; however this method requires all of them to be calculated. Therefore, it is not efficient in terms of the number of calculations made.

Using a top-down approach and dynamic programming, only the necessary V values can be calculated and the performance can be increased significantly. On the other hand, in order to prevent re-calculations, all calculated V values should be kept in memory for all horizons. Therefore, this approach is not efficient in terms of memory. However, since the memory requirements with that approach for our test problems do not exceed our practical limits; we preferred this approach due to its runtime efficiency.

5.2.3. Complexity Analysis

The complexity of the approximate fitness calculation depends on the complexity of the simulation steps. At each simulation step, the agents submit their action, the environment state is updated, the observations are determined and the agents update their FSC states using these observations. The agents find their best actions and update their FSC states in $O(1)$ time. The next state is determined according to the current state and the current joint action. Each state is tested as a possible next state, therefore its complexity is $O(N_s)$. Similarly, each joint observation is tested as a possible observation in $O(N_o \cdot n)$ time. Therefore, the complexity of the algorithm using the approximate fitness calculation is $O(N_p \cdot h \cdot N_{sc} \cdot (N_s + (N_o \cdot n)))$. When Equation 5.1 is used, the complexity of the algorithm using the exact fitness calculation is $O(N_p \cdot h \cdot N_s^2 \cdot N_{fsc}^2 \cdot N_o^2)$, where N_{fsc} is the number of nodes in the FSC.

5.2.4. General Flow of Our Algorithm

Figure 5.2 shows how the fitness of a single chromosome is calculated without going into much detail and Figure 5.3 shows how our algorithm flows in general. While calculating the fitness, the flow changes according to the fitness calculation type. For the exact fitness calculation, Equation 5.1 is used. In the case of the approximate fitness calculation, an estimated fitness is calculated by making 50 simulations in the step 6 and if it is better than the best fitness so far, its fitness is calculated more accurately by making 1,000 simulations in the step 8. For both calculation type, the calculated fitness is compared against the best fitness and if the new chromosome is better, the best chromosome is replaced with

it.

```
1: FCT ← Fitness Calculation Type
2: fitness = 0
3: if FCT is exact then
4:     fitness = calcFitnessExactly(c)
5: else {FCT is approximate}
6:     fitness = calcFitnessApproximately(c, 50)
7:     if fitness > bestFitness then
8:         fitness = calcFitnessApproximately(c, 1000)
9:     end if
10: end if
11: if fitness > bestFitness then
12:     bestFitness = fitness
13:     bestChromosome = c
14: end if
15: return fitness
```

Figure 5.2. The algorithm of the fitness calculation of a single chromosome.

Figure 5.3 starts with the initialization of the population. Each chromosome in the population is generated randomly in step 8 and its fitness is calculated in step 9. Then, the evolution continues until the population is converged. Crossovers are made between steps $12 - 16$. The fitness of the chromosomes generated with crossover is calculated and they are added to the population. A similar process happens for mutation between steps $17 - 22$. After that, via a selection process, the size of the population is reduced to the PSize, which is the actual population size.

5.3. Experiments and Results

In order to compare the results with our previous methods, we solved the same problems. Before starting to make experiments with all the problems, we decided to make extensive tests on one of the problems and see the effect of the algorithm parameters.

```
1: P ← Population
2: PSize ← Population Size
3: CrossoverCount ← Number of new chromosome to produce with crossover
4: MutationCount ← Number of new chromosome to produce with mutation
5: bestFitness = 0
6: bestChromosome = null
7: for c in P do
8:     Initialize c randomly
9:     Calculate fitness of c
10: end for
11: while Population is not converged do
12:     for i = 1 to CrossoverCount do
13:         cNew ← Create new chromosome with crossover
14:         Calculate fitness of cNew
15:         add cNew to P
16:     end for
17:     for i = 1 to MutationCount do
18:         cNewTmp ← Select a chromosome randomly
19:         cNew ← Mutate cNewTmp
20:         Calculate fitness of cNew
21:         add cNew to P
22:     end for
23:     Reduce the population size to PSize with selection
24: end while
```

Figure 5.3. General flow of the GA-FSC algorithm.

5.3.1. Determination of Algorithm Parameters

There are several parameters that can be changed in our algorithm which may affect the solution quality and performance:

- *Population size* is one of the GA parameters such that increasing the population size may help to search the policy space better at the cost of increasing the run-time. We

made our tests with three different population sizes: 40, 80 and 120.

- *Crossover type* is another GA parameter. We tested one point crossover and uniform crossover.
- *Node count in a FSC* determines both the capability and the complexity of it and also the size of the chromosome in our representation. Since we keep the chromosome size fixed, FSC node count should be determined at the beginning. We empirically determined the node counts for each problem. Since each node corresponds to an action in the FSCs, we determined the minimum FSC size as the number of actions for a particular problem and the maximum FSC size is chosen as 10.
- *Fitness calculation technique* is also important. We used both *exact* and *approximate* calculation techniques for evaluating the fitness.
- *Mutation probability* is also a parameter in GA, in order not to increase the parameter space too much we used it as 0.1 for all cases.

In our preliminary experiments, we realized that our algorithm gives worse results for the *Dec-Tiger Problem* when compared to other test problems. For this reason, we decided to test all parameters on this problem and make a detailed analysis on the affect of these parameters on the solution quality. For all other problems, we kept the population size fixed, since we realized that population size 80 is sufficient for obtaining satisfactory results. On the other hand, we tried all other parameters for the remaining problems.

We made our tests on these problems for various horizons. For each of the parameter combinations mentioned above, we made 10 different runs with random initial populations and we report the best expected rewards obtained in all of these runs. We use these results to compare our algorithm with the exact solution algorithms. We also report the average reward and the standard deviation of these 10 runs. These are used to compare with the studies, especially approximate solutions, that report their average results for these problems. Since there are many studies that report results for these problems, we compared our results with only the best exact and approximate approaches when available for the simplicity of the result tables.

5.3.2. Dec-Tiger Problem

We present our best and average results for several horizons depending on the fitness calculation type in Table 5.1. From these results, it can be inferred that we obtain better results using the exact fitness calculation as expected. When we compare our results with the other approaches, we observe that we are able to obtain the optimal results up to horizon 6 for the best case, except for horizon 5. Similarly, our average results are comparable with the *MBDP* algorithm. For horizons larger than 6, the optimal values are not available yet in the literature. For horizon 10, our best result is better than that of the *MBDP* approach. For horizons 8 and 100, our results are worse. Also, the *PBIP* approach that reports for only horizon 10 and 100 have better results than ours. When we compare our results for horizons 10 and 100 we see that the reward value for horizon 100 is less than the horizon 10 value whereas we expect a much better value.

This observation hints that instead of starting with a completely random initial population, taking at least one of the members of the initial population as the solution obtained for a lower horizon while initializing the rest randomly might improve the solution. With this change, we made another set of test runs for this problem using the exact fitness calculation and added the results obtained into Table 5.1 by indicating them as *Case 2*. As a result of this, we have a slight improvement for horizon 8 and a major improvement for horizon 100. In this case, in addition to the improvement of the best results, our average results are also comparable with both *MBDP* and *PBIP* approach. Therefore, it is a good idea to inject the solution for a lower horizon to the initial population whenever possible. However, we made our tests for all other problems using a completely random initial population.

5.3.2.1. Effect of the FSC Size. One of the most important parameters that can affect the expected reward is the FSC size. Intuitively, we can say that if the FSC size is too small it may not represent the optimal policy, on the other hand, if it is too large, it may be hard to learn a near optimal policy with a complex FSC. As an example, we present how the obtained reward changes according to the FSC size using exact fitness calculation and one-

Table 5.1. Test results for the Dec-Tiger problem using the GA-FSC approach.

	APPROXIMATE		OPTIMAL	GA-FSC Exact		GA-FSC Exact (Case 2)		GA-FSC Approximate	
Horizon	MBDP	PBIP	GMAA*-ICE	Best	Average	Best	Average	Best	Average
2	-4.00	-	-4.00	**-4.00**	-4.00 $\mp$ 0.00	**-4.00**	-4.00 $\mp$ 0.000	**-4.00**	-5.48 $\mp$ 1.59
3	5.19	-	5.19	**5.19**	5.19 $\mp$ 0.00	**5.19**	5.19 $\mp$ 0.000	**5.19**	4.37 $\mp$ 1.32
4	4.80	-	4.80	**4.80**	4.39 $\mp$ 0.67	**4.80**	3.89 $\mp$ 0.638	3.99	-3.25 $\mp$ 2.69
5	5.38	-	7.03	3.75	-12.61 $\mp$ 26.11	3.76	2.11 $\mp$ 0.903	-0.52	-5.14 $\mp$ 3.43
6	9.91	-	10.38	**10.38**	4.74 $\mp$ 5.19	**10.38**	5.97 $\mp$ 2.380	1.72	-8.15 $\mp$ 6.77
8	9.42	-	X	6.94	-0.92 $\mp$ 2.87	8.00	6.96 $\mp$ 0.430	-2.75	-19.12 $\mp$ 35.76
10	13.49	13.6	X	13.57	2.39 $\mp$ 6.18	13.57	13.57$\mp$ 0.000	-1.97	-13.03 $\mp$ 8.17
100	93.24	147	X	10.06	-6.56 $\mp$ 8.87	169.30	169.30$\mp$ 0.000	7.68	-44.64 $\mp$ 81.63

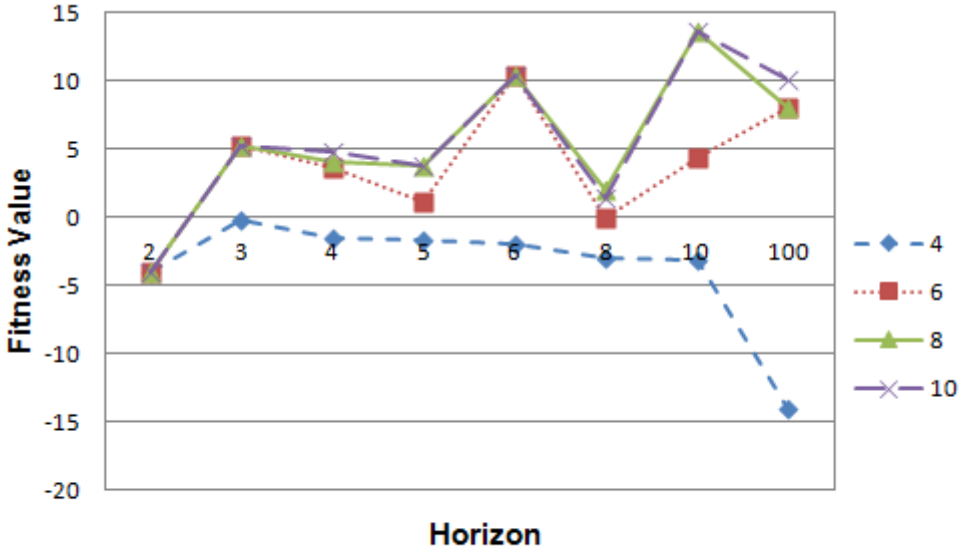

Figure 5.4. Effect of the FSC Size on the reward for the Dec-Tiger problem using exact fitness calculation and one-point crossover.

point crossover in Figure 5.4. In order to make the figure more readable we include the results of four FSC sizes. The obtained behavior by changing FSC size is similar to what we expected. FSC having the 4 nodes has the worst results. Increasing the node count to 6 has a considerable impact on the reward. When the node count is increased to 8, there is a slight change for all the horizons except horizon 10 which we have a considerable increase. However, when the node count is increased to 10, the change is very small.

5.3.2.2. Effect of the Population Size. We also wanted to see the effect of the population size on the best reward obtained. To illustrate this behavior, we examined how the reward changes for different population sizes in the case where the fitness is calculated exactly and one-point crossover is used and we chose the horizon 10 as an example. We present the obtained graphs in Figure 5.5. It can be concluded from the graph that there is no clear trend with the increasing population size, however we can say that the best reward did not decrease from population size 40 to 80 and it increased for some of the cases. On the other hand, the reward can decrease, increase or remain constant from population size 80 to 120. Therefore, we can say that population size 80 is a good choice because it helps

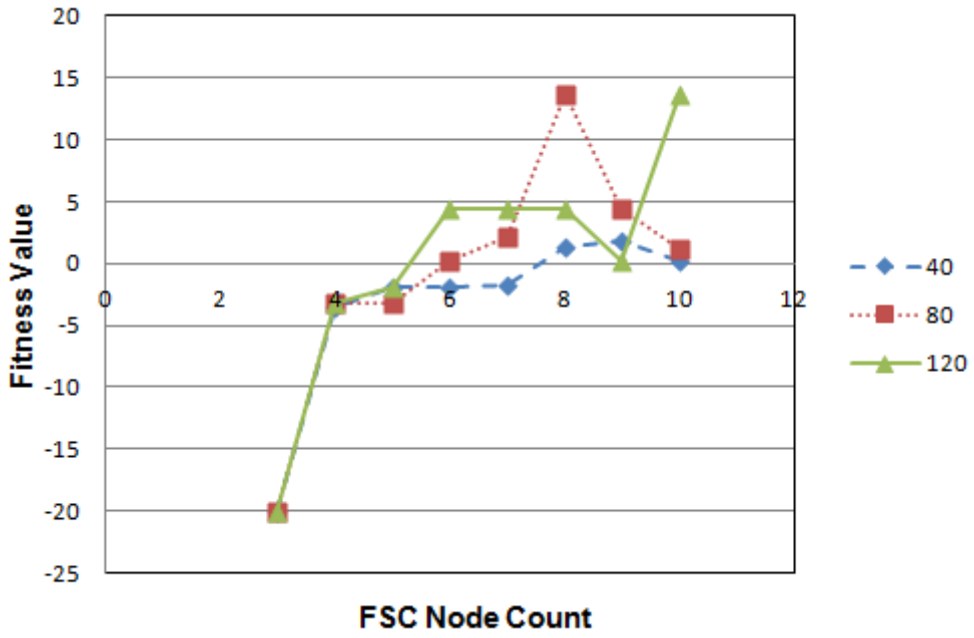

Figure 5.5. Reward change according to population size and the FSC Size for the Dec-Tiger problem when the fitness is calculated exactly and one-point crossover is used for horizon 10.

us to obtain better rewards than population size 40 and it is not worth spending extra time for population size 120 since it has no clear advantage. For this reason, we choose our population size as 80 for all of the following test problems.

5.3.2.3. Effect of the Crossover Type. To see the effect of the crossover type, we compared our best results obtained for several horizons both using exact and approximate fitness calculation and we present the obtained graphs in Figure 5.6. When the exact fitness calculation is used, the obtained best results for both crossover types are the same for horizons 2, 3, 4, 6, and 10. For horizons 5 and 100, one-point crossover produces slightly better results whereas uniform crossover produces a better result only for horizon 8. When the approximate fitness calculation is used, the best obtained results are the same for horizon 2 and 3. One-point crossover produces better results for horizons 10 and 100, and uniform crossover produces better results for horizon 4,5,6 and 8. Therefore, it can be said that the results obtained using both crossover type are similar in terms of the best results in general. In some cases, one may perform better than the other, however, the results are comparable.

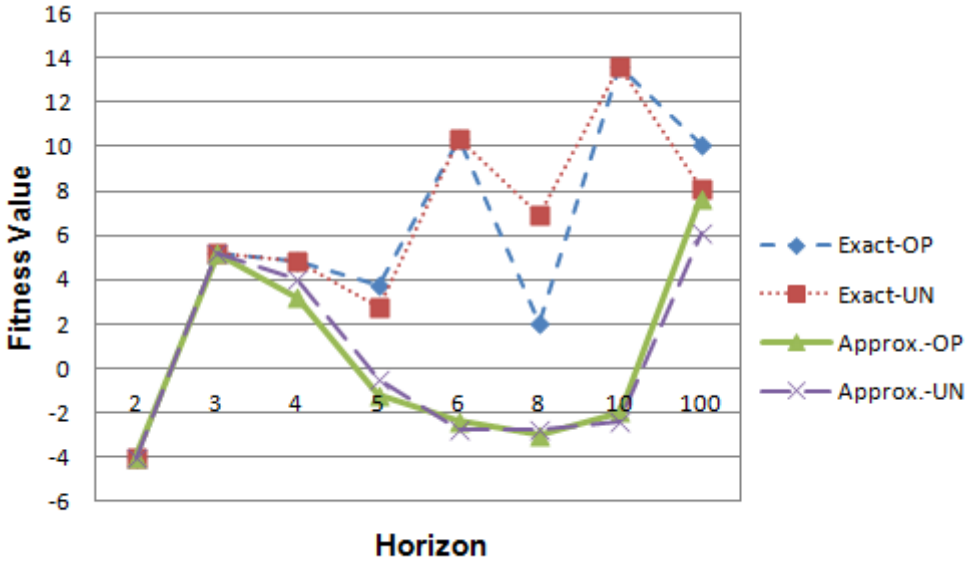

Figure 5.6. Reward change according to the crossover type for the Dec-Tiger Problem with respect to number of horizons.

5.3.2.4. Difficulty of the Dec-Tiger Problem. The Dec-Tiger problem is a small problem in terms of the number of states, actions and observations. For this reason one may be tempted to consider it as a simple problem. However, the difficulty of a problem is determined not only by the size of the problem, the characteristics of the problem are also important. Having a correct information about the environment state is very important for the agents to make meaningful actions. In this respect, the information gained due to an observation taken from the environment is very important. In the Dec-Tiger problem an agent must submit several consecutive "Listen" actions in order to have a "sufficient" belief about the correct door. What constitutes a "sufficient" belief should be determined by the agent and it is also influenced by the problem definition. The punishment of a wrong action is higher than a correct action, therefore the agent should be pretty sure about the correct door. It should make its choice as soon as possible, since the "Listen" action also has a punishment. These requirements point to a complex policy and a bigger search space than expected at the first sight.

5.3.3. Multi-access Broadcast Channel

The results show that, we are able to obtain the optimal results up to horizon 10 with exact fitness calculation. The obtained results for larger horizons are slightly less than the optimal values. The optimal *GMAA*-ICE* algorithm is able to solve up to horizon 900. We show that we are able to obtain a solution for horizon 1000 like the *MBDP* algorithm. The obtained results with the approximate fitness approach is optimal up to horizon 4 and for larger horizons, they are slightly worse than that of the exact fitness approach. At the same time, the standard deviation in our runs is very low meaning that we are able to obtain similar results to the best ones in most of the runs.

5.3.4. Meeting In a Grid

The results for this problem are reported only up to horizon 6; however our algorithm is able handle much further horizon values than those reported in these papers. In addition to horizon 6, we also report our results for horizon 50 for both comparison purposes among our techniques and for future reference. Our results are given in Table 5.3.

By using the exact fitness approach we are able to obtain the optimal results up to horizon 6 and our average reward values are close to the optimal. For the approximate fitness approach, we could obtain the optimal result for only horizon 2. However, the obtained results for larger horizons are still comparable with the exact fitness approach. The standard deviation is a little higher in the approximate fitness case.

5.3.5. Recycling Robots Problem

Our results for this problem are presented in Table 5.4. In order to show that our algorithm can handle larger horizons than GMAA*-ICE algorithm, we also give a result for horizon 100. We do not currently know whether or not we have obtained an optimal solution for horizon 100, but for all other horizons, we could obtain the optimal solutions using both exact and approximate fitness calculations with a low standard deviation.

Table 5.2. Test results for the Multi-access Broadcast Channel problem using the GA-FSC approach.

	APPROXIMATE	OPTIMAL		GA-FSC Exact		GA-FSC Approximate	
Horizon	MBDP	Loss. Clust.	GMAA*-ICE	Best	Average	Best	Average
2	2.00	2.00	-	**2.00**	2.00 $\mp$ 0.00	**2.00**	2.00 $\mp$ 0.00
4	3.89	3.89	-	**3.89**	3.84 $\mp$ 0.05	**3.89**	3.83 $\mp$ 0.08
8	7.49	7.49	-	**7.49**	7.39 $\mp$ 0.03	7.40	7.35 $\mp$ 0.05
10	9.29	9.29	-	**9.29**	9.19 $\mp$ 0.03	9.20	9.15 $\mp$ 0.05
20	18.29	18.31	18.31	18.29	18.19 $\mp$ 0.06	18.20	18.17 $\mp$ 0.05
50	45.29	X	45.50	45.20	45.20 $\mp$ 0.000	45.20	45.14 $\mp$ 0.05
100	90.29	X	90.76	90.29	90.19 $\mp$ 0.03	90.20	90.13 $\mp$ 0.05
250	-	X	226.50	225.29	225.18 $\mp$ 0.04	225.20	225.10 $\mp$ 0.04
500	-	X	452.74	450.29	450.19 $\mp$ 0.06	450.20	450.09 $\mp$ 0.08
900	-	X	814.71	810.29	810.19 $\mp$ 0.03	810.20	810.10 $\mp$ 0.07
1000	900.29	X	X	900.29	900.17 $\mp$ 0.07	910.20	900.10 $\mp$ 0.08

Table 5.3. Test results for the Meeting in a Grid problem using the GA-FSC approach.

	APPROX.	OPTIMAL	GA-FSC Exact		GA-FSC Approximate	
Horizon	Cross Ent.	GMAA*-ICE	Best	Average	Best	Average
2	0.91	0.91	**0.91**	0.91 $\mp$ 0.00	**0.91**	0.88 $\mp$ 0.04
3	1.55	1.55	**1.55**	1.55 $\mp$ 0.00	1.55	1.47 $\mp$ 0.04
4	2.24	2.24	**2.24**	2.23 $\mp$ 0.03	2.23	2.11 $\mp$ 0.07
5	2.96	2.97	**2.97**	2.95 $\mp$ 0.05	2.93	2.71 $\mp$ 0.13
6	3.64	3.72	**3.72**	3.68 $\mp$ 0.02	3.62	3.45 $\mp$ 0.10
50	X	X	40.49	40.30 $\mp$ 0.40	40.18	37.80 $\mp$ 1.57

Table 5.4. Test results for the Recycling Robots problem using the GA-FSC approach.

	OPTIMAL	GA-FSC Exact		GA-FSC Approximate	
Horizon	GMAA*-ICE	Best	Average	Best	Average
5	16.49	**16.49**	16.17 $\mp$ 0.16	**16.49**	16.13 $\mp$ 0.13
15	47.25	**47.25**	46.90 $\mp$ 0.12	**47.25**	46.85 $\mp$ 0.23
18	56.48	**56.48**	56.21 $\mp$ 0.19	**56.48**	56.13 $\mp$ 0.12
20	62.63	**62.63**	62.33 $\mp$ 0.16	**62.63**	62.13 $\mp$ 0.51
30	93.40	**93.40**	93.13 $\mp$ 0.19	**93.40**	92.69 $\mp$ 0.68
40	124.17	**124.17**	123.86 $\mp$ 0.16	**124.17**	123.83 $\mp$ 0.12
50	154.94	**154.94**	154.67 $\mp$ 0.19	**154.94**	154.45 $\mp$ 0.35
60	185.71	**185.71**	185.44 $\mp$ 0.19	**185.71**	184.94 $\mp$ 0.81
70	216.48	**216.48**	216.21 $\mp$ 0.19	**216.48**	215.85 $\mp$ 1.01
100	X	308.79	308.48 $\mp$ 0.16	308.79	307.98 $\mp$ 0.74

5.3.6. Cooperative Box Pushing

This problem is one of the most complex problems in terms of the number of states. The exact fitness calculation suffers from the number of states and it takes too much time with the growing number of horizons when compared to the approximate fitness calculation. The recent *GMAA*-ICE* approach reports the optimal results, but just up to horizon 4. We are able to obtain the optimal result for only horizon 2 for both exact and approximate fitness calculations. For horizons 3 and 4, our reward values are close to the optimal. *Improved MBDP* algorithm is an approximate technique that can produce solutions for larger horizons, however, for both approximate and exact fitness calculations, we obtained better results except for horizon 3 in terms of the best results and average results.

One interesting observation with this problem is that the standard deviation for horizon 3 is very high when compared to the other horizons and it gets much lower when the horizon becomes 4. This is contrary to the intuition that obtaining the strategy for a larger horizon should be harder. The best the agents can do for horizon 3 is to push the large box to the target and this does not change when it horizon is 4, because a single extra action is not sufficient for obtaining extra positive reward. On the other hand, they have the opportunity to follow an alternative strategy since they have an extra action. Therefore, there are more strategy alternatives in this case. For this reason, it is more probable to obtain a better strategy for horizon 4 and the standard deviation is lower. Since the environment is reset in this problem after the large or small box reaches the target, it has a repetitive characteristic. Therefore, the strategy for larger horizons should be similar to the previous horizons. Also, a standard deviation of 0 value may appear as counterintuitive. As mentioned before, we try several parameters making 10 runs for each of them and report the best value for all these runs. The reported average and the standard deviation corresponds to the 10 run of the parameter set giving the best reward. Different parameters, especially different FSC sizes, may produce different reward values in general. However, the obtained strategies for the same parameter set is very similar. Therefore, the standard deviation is low in this case.

Table 5.5. Test results for the Cooperative Box Pushing problem using the GA-FSC approach.

	APPROXIMATE	OPTIMAL	GA-FSC Exact		GA-FSC Approximate	
Horizon	IMBDP	GMAA*-ICE	Best	Average	Best	Average
2	17.60	17.60	**17.60**	17.60 $\mp$ 0.00	**17.60**	17.60 $\mp$ 0.00
3	65.68	66.08	65.27	22.98 $\mp$ 14.81	64.87	22.68 $\mp$ 14.41
4	97.91	98.59	98.17	98.13 $\mp$ 0.04	98.10	94.07 $\mp$ 3.43
8	182.92	X	189.05	188.64 $\mp$ 0.14	188.60	183.74 $\mp$ 4.30
10	189.32	X	216.98	216.60 $\mp$ 0.15	216.50	216.49 $\mp$ 0.04
20	415.25	X	468.15	468.15 $\mp$ 0.00	467.89	467.80 $\mp$ 0.16
50	1051.82	X	1201.04	1201.05 $\mp$ 0.00	1200.22	1199.52 $\mp$ 0.20
100	2112.05	X	2420.26	2420.26 $\mp$ 0.00	2418.55	2418.53 $\mp$ 0.06

5.3.7. Fire Fighting Problem

The Fire Fighting is the problem with the most number of states we have attempted to solve so far, namely 432 states. The *GMAA*-ICE* approach reported the optimal results up to horizon 6 [16]. We obtain the optimal results for all horizons except horizon 5 using the exact fitness calculation. The approximate fitness approach produced the optimal results for only horizon 2 and the results are close to the optimal for all horizons. On the other hand, when we compare the run times, we observe that the approximate fitness calculation takes much less time. Therefore, it may be better to use approximate fitness calculation for higher horizons.

Table 5.6. Test results for the Fire Fighting problem using the GA-FSC approach.

	OPTIMAL	GA-FSC Exact		GA-FSC Approximate	
Horizon	GMAA*-ICE	Best	Average	Best	Average
2	-4.38	**-4.38**	-4.38 $\mp$ 0.00	**-4.38**	-4.48 $\mp$ 0.08
3	-5.74	**-5.74**	-5.77 $\mp$ 0.01	-5.76	-5.94 $\mp$ 0.11
4	-6.58	**-6.58**	-6.61 $\mp$ 0.02	-6.63	-6.91 $\mp$ 0.20
5	-7.07	**-7.07**	-7.09 $\mp$ 0.01	-7.10	-7.83 $\mp$ 0.53
6	-7.18	**-7.18**	-7.25 $\mp$ 0.11	-7.55	-8.52 $\mp$ 0.91

5.3.8. Discussion on the Determination of the Algorithm Parameters

There are several parameters that can be adjusted in our algorithm and we made several experiments to see their effect on the solution quality. We presented our results for each test problem above. Since the problems given above have been previously solved to some horizon, we can comment on the quality of our solutions by comparing our results with theirs and decide when to stop experimenting with the parameters. On the other hand, if a new DEC-POMDP problem is tried to be solved, our algorithm does not give any indication about whether our results close to the optimal or not. Therefore, for each problem, one should decide herself how to manage the parameters and determine when the obtained solution is "sufficiently good". Nevertheless, here we propose a guideline about

how to attack a new DEC-POMDP problem and tune the parameters.

5.3.8.1. Fitness Calculation Type. According to our experiments, the fitness calculation type is one of the most important parameters and it may affect both the solution quality and the run-time performance significantly. Our experiments show that exact fitness calculation should be preferred to approximate fitness calculation whenever possible. However, when the problem has many states, e.g. over 100, exact calculation may have performance problems and approximate fitness should be considered. Using approximate fitness requires setting the number of simulations to be made. Our experiments show that, 50 is enough for most of the problems, however, there may be some problems like Dec-Tiger where this number of simulations will not be sufficient. In order to decide on that, it would be useful to make some tests on several strategies generated during evolution process and calculate their fitness approximately several times. If these calculations give similar results consistently, this gives an indication about the sufficiency of the number of simulations. If the results are considerably different, the number of simulation should be increased. Our experiments with the Dec-Tiger problem show that, whenever a policy for a smaller horizon is available, it can be put directly into the initial population as a chromosome and the remaining can be determined randomly. This should be considered for both exact and approximate fitness calculation cases.

5.3.8.2. FSC Size. The FSC size is another important parameter, because it directly determines the limits of the policy. Our experiments show that, the reward value increases with the increasing the FSC size up to some node count, because the small FSCs are not enough to express the optimal policy and increasing node count also increases the capability of the FSC. There is an optimal node count and in the ideal case even if we increase the node count further, the reward value is not expected to drop because larger FSC is also able to express the smaller FSC. On the other hand, when the structure becomes more complex and the chromosome becomes longer, it becomes harder to learn the optimal policy using the GA and the reward may indeed decrease. For comparison purposes, we tried all FSC sizes up to 10 starting from the number of actions in the corresponding DEC-POMDP problem. For a new problem, the test intervals may be chosen higher and more than 10 node FSCs

can be tested. For example, if we have a 5 action problem, instead of testing all the FSCs having 5 to 10 nodes; 5, 8, 10, 15 and 20 node FSCs can be tested and a better coverage may be obtained. Also, since it may be harder to learn a larger FSC, as the FSC size gets larger, the number of test runs with the same configuration may also be increased.

5.3.8.3. Population Size. We tested 3 different *population sizes* with the Dec-Tiger problem and we realized that 80 chromosomes are sufficient to obtain satisfactory results, therefore, for all other problems we used it as the population size. We suggest starting with a population size of 80 and if the results are not satisfactory we recommend increasing the population size, since it may help searching the policy space better.

5.3.8.4. Crossover Operator. We tested both one point crossover and uniform crossover for all problems and none of them has any significant advantage over the other. For a particular problem and particular test case, one of them may produce better results, however, for some other cases the other one may lead. Besides, the differences between the reward values observed are insignificant, therefore, we can say that the crossover type does not have a significant effect on the solution quality in general. Therefore, any of the crossover type can be selected at the beginning, however, for fine-tuning purposes the other one may also be tried.

5.3.8.5. Mutation Operator. We take mutation rate constant in our experiments as 0.1. It may be useful to have mutations when the population converges to a local optimum. On the other hand, since the purpose of a mutation is to jump to the unvisited parts of the search space, making several runs from different initial populations may be an alternative to trying several mutation rates.

6. COMPARISON OF THE ALGORITHMS

In the previous chapters, we developed four novel algorithms to solve finite horizon DEC-POMDP problems approximately. We provided experimental results in the corresponding sections and we compared our results with the ones in the literature. In this part, we compare the expected reward performance and the run-time performance of these algorithms.

6.1. Expected Reward Performance of the Algorithms

We solved six different DEC-POMDP problems with each of our algorithms. Depending on the algorithm, we tried different cases. For example, with the MAP algorithm we tested two cases: communication between the agents is allowed and not allowed. Since there is no communication between the agents in the DEC-POMDP definition, we only report the results with the no-communication case here. With the $ES-BV$ and the $ES-OH$ algorithms, there are several parameters, but we determine a parameter set empirically and we report the results in terms of this set. We calculated the fitness both exactly and approximately in the $GA-FSC$ algorithm and we report these also here. Except the MAP approach, we made 10 runs for each case due to the randomness in the methods. We compare here both the best of these runs and the average of them. For each horizon, we print the best rewards in bold.

6.1.1. Dec-Tiger Problem

For the DEC-Tiger problem, we tested the effect of using previous horizon solutions with the $GA-FSC$ algorithm and we see its contribution to the results for some horizons. However, in this part we want to see how the algorithms perform in their standard forms, so we do not give these results. Despite this, the $GA-FSC$ (exact) approach outperforms the others for most horizons both in terms of the average and the best reward. The $ES-BV$ algorithm has the closest performance to the $GA-FSC$ (exact) and the MAP algorithm

has the worst performance. The $GA - FSC$ (approximate) approach is better than the $ES - OH$ approach in terms of the best reward, however, when the average reward is considered there is no clear winner. The results can be seen in Table 6.1.

6.1.2. Multi-access Broadcast Channel

The summary of the results for this problem can be seen in Table 6.2. The $GA-FSC$ (exact) algorithm has the best performance and the $GA - FSC$ (approximate) approach has the closest performance. All other methods are slightly worse than these methods and they have a similar performance both in terms of the best reward and average reward.

6.1.3. Meeting In a Grid

The summary of the results for this problem can be seen in Table 6.3. In general, the $GA - FSC$ (exact) approach has the best performance and the $GA - FSC$ (approximate) is the next. Both in terms of the average and the best reward, there is not a clear winner among the $ES - BV$ and the $ES - OH$ approaches for this problem; the winner changes from one horizon to another. The MAP algorithm has the worst performance.

6.1.4. Recycling Robots Problem

In terms of the best reward, the $GA - FSC$ approach outperforms the other methods both using the exact and approximate fitness calculations. The $GA-FSC$ (exact) approach is also the best in terms of average reward, however there is no clear outperformer among the $ES-BV$, $ES-OH$ and $GA-FSC$ (approximate) approaches. The MAP algorithm has the worst performance once again. The summary of the results for this problem can be seen in Table 6.4.

Table 6.1. Expected reward performance of our algorithms for the Dec-Tiger problem.

Horizon	MAP	ES-BV		ES-OH		GA-FSC			
	comm.					Exact		Approximate	
	Best	Best	Average	Best	Average	Best	Average	Best	Average
2	-14.18	**-4.00**	**-4.00 ∓ 0.00**	**-4.00**	**-4.00 ∓ 0.00**	**-4.00**	**-4.00 ∓ 0.00**	**-4.00**	-5.48 ∓ 1.59
3	-16.29	2.54	1.93 ∓ 1.24	-5.41	-5.62 ∓ 0.25	**5.19**	**5.19 ∓ 0.00**	**5.19**	4.37 ∓ 1.32
4	-28.03	0.61	0.54 ∓ 0.07	-4.72	-5.92 ∓ 0.82	**4.80**	**4.39 ∓ 0.67**	3.99	-3.25 ∓ 2.69
5	-30.11	-0.92	-1.71 ∓ 1.00	**7.03**	**4.88 ∓ 1.86**	3.75	-12.62 ∓ 26.11	-0.52	-5.14 ∓ 3.43
6	-42.50	1.05	0.07 ∓ 0.81	-7.01	-9.18 ∓ 1.48	**10.38**	**4.74 ∓ 5.19**	1.72	-8.15 ∓ 6.77
8	-57.06	0.20	-1.44 ∓ 0.92	-9.45	-10.22 ∓ 1.32	**6.94**	**-0.92 ∓ 2.87**	-2.75	-19.12 ∓ 35.76
10	-71.05	-1.76	-1.84 ∓ 0.05	-1.99	-2.46 ∓ 0.49	**13.57**	**2.39 ∓ 6.18**	-1.97	-13.03 ∓ 8.17
100	-709.30	7.04	**6.76 ∓ 0.40**	-60.64	-74.66 ∓ 8.74	**10.06**	-6.56 ∓ 8.87	7.68	-44.64 ∓ 81.63

Table 6.2. Expected reward performance of our algorithms for the Multi-access Broadcast Channel problem.

Horizon	MAP	ES-BV		ES-OH		GA-FSC			
	comm.					Exact		Approximate	
	Best	Best	Average	Best	Average	Best	Average	Best	Average
2	1.90	1.90	1.90 $\mp$ 0.00	1.90	1.90 $\mp$ 0.00	**2.00**	**2.00 $\mp$ 0.00**	**2.00**	**2.00 $\mp$ 0.00**
4	3.70	3.70	3.70 $\mp$ 0.00	3.70	3.70 $\mp$ 0.00	**3.89**	**3.84 $\mp$ 0.05**	**3.89**	3.83 $\mp$ 0.08
8	7.31	7.30	7.30 $\mp$ 0.00	7.31	7.30 $\mp$ 0.00	**7.49**	**7.39 $\mp$ 0.03**	7.40	7.35 $\mp$ 0.05
10	9.09	9.11	9.11 $\mp$ 0.00	9.11	9.10 $\mp$ 0.00	**9.29**	**9.19 $\mp$ 0.03**	9.20	9.15 $\mp$ 0.05
20	18.08	18.11	18.11 $\mp$ 0.00	18.11	18.11 $\mp$ 0.00	**18.29**	**18.19 $\mp$ 0.06**	18.20	18.17 $\mp$ 0.05
50	45.10	45.12	45.12 $\mp$ 0.00	45.11	45.11 $\mp$ 0.00	**45.20**	**45.20 $\mp$ 0.00**	**45.20**	45.14 $\mp$ 0.05
100	90.13	90.12	90.11 $\mp$ 0.01	90.11	90.11 $\mp$ 0.00	**90.29**	**90.19 $\mp$ 0.03**	90.20	90.13 $\mp$ 0.05
250	225.14	225.11	225.11 $\mp$ 0.00	225.12	225.11 $\mp$ 0.00	**225.29**	**225.18 $\mp$ 0.04**	225.20	225.10 $\mp$ 0.04
500	450.06	450.13	450.13 $\mp$ 0.01	450.13	450.13 $\mp$ 0.00	**450.29**	**450.19 $\mp$ 0.06**	450.20	450.09 $\mp$ 0.08
900	810.23	810.21	810.21 $\mp$ 0.00	810.23	**810.23 $\mp$ 0.00**	**810.29**	810.19 $\mp$ 0.03	810.20	810.10 $\mp$ 0.07
1000	899.98	899.99	899.99 $\mp$ 0.00	899.99	899.99 $\mp$ 0.00	**900.29**	**900.17 $\mp$ 0.07**	910.20	900.10 $\mp$ 0.08

Table 6.3. Expected reward performance of our algorithms for the Meeting in a Grid problem.

Horizon	MAP	ES-BV		ES-OH		GA-FSC			
	comm.					Exact		Approximate	
	Best	Best	Average	Best	Average	Best	Average	Best	Average
2	0.82	**0.91**	**0.91 ∓ 0.00**	0.87	0.86 ∓ 0.00	**0.91**	**0.91 ∓ 0.00**	**0.91**	0.88 ∓ 0.04
3	1.29	1.53	1.53 ∓ 0.00	**1.55**	**1.55 ∓ 0.00**	**1.55**	**1.55 ∓ 0.00**	**1.55**	1.47 ∓ 0.04
4	1.75	2.19	2.18 ∓ 0.00	2.00	1.99 ∓ 0.01	**2.24**	**2.23 ∓ 0.03**	2.23	2.11 ∓ 0.07
5	2.14	2.87	2.86 ∓ 0.01	2.64	2.63 ∓ 0.01	**2.97**	**2.95 ∓ 0.05**	2.93	2.71 ∓ 0.13
6	2.62	3.59	3.58 ∓ 0.01	**3.72**	**3.71 ∓ 0.00**	**3.72**	3.681 ∓ 0.02	3.62	3.45 ∓ 0.10
50	21.77	37.91	37.24 ∓ 1.3	38.79	38.62 ∓ 0.21	**40.49**	**40.30 ∓ 0.40**	40.18	37.80 ∓ 1.57

Table 6.4. Expected reward performance of our algorithms for the Recycling Robots problem.

Horizon	MAP	ES-BV		ES-OH		GA-FSC			
	comm.					Exact		Approximate	
	Best	Best	Average	Best	Average	Best	Average	Best	Average
5	13.34	16.11	16.11 $\mp$ 0.00	15.39	15.39 $\mp$ 0.00	**16.49**	**16.17 $\mp$ 0.16**	**16.49**	16.13 $\mp$ 0.13
15	35.44	46.89	46.88 $\mp$ 0.00	46.17	46.17 $\mp$ 0.01	**47.25**	**46.90 $\mp$ 0.12**	**47.25**	46.85 $\mp$ 0.23
18	42.31	56.13	56.12 $\mp$ 0.01	55.41	55.40 $\mp$ 0.01	**56.48**	**56.21 $\mp$ 0.19**	**56.48**	56.13 $\mp$ 0.12
20	46.78	62.27	62.26 $\mp$ 0.01	61.57	61.56 $\mp$ 0.01	**62.63**	**62.33 $\mp$ 0.16**	**62.63**	62.13 $\mp$ 0.51
30	68.93	93.03	93.02 $\mp$ 0.01	93.06	93.05 $\mp$ 0.01	**93.40**	**93.13 $\mp$ 0.19**	**93.40**	92.69 $\mp$ 0.68
40	91.16	123.81	123.81 $\mp$ 0.00	123.82	123.81 $\mp$ 0.01	**124.17**	**123.86 $\mp$ 0.16**	**124.17**	123.83 $\mp$ 0.12
50	113.57	154.61	154.59 $\mp$ 0.03	154.59	154.58 $\mp$ 0.01	**154.94**	**154.67 $\mp$ 0.19**	**154.94**	154.45 $\mp$ 0.35
60	135.58	185.36	185.36 $\mp$ 0.00	185.36	185.35 $\mp$ 0.01	**185.71**	**185.44 $\mp$ 0.19**	**185.71**	184.94 $\mp$ 0.81
70	157.91	216.12	216.12 $\mp$ 0.00	216.14	216.12 $\mp$ 0.02	**216.48**	**216.21 $\mp$ 0.19**	**216.48**	215.85 $\mp$ 1.01
100	224.63	308.46	308.45 $\mp$ 0.01	308.44	308.42 $\mp$ 0.02	**308.79**	**308.48 $\mp$ 0.16**	**308.79**	307.98 $\mp$ 0.74

6.1.5. Cooperative Box Pushing

The summary of the results for this problem can be seen in Table 6.5. The $GA-FSC$ algorithm outperforms all the others for this problem both in terms of the best reward and average reward. The $ES-OH$ approach has slightly worse results in terms of the best reward, however the average rewards of this method are considerably less than the $GA-FSC$ method. This means that the $GA-FSC$ produces consistently similar results, while the $ES-OH$ approaches produce good results sometimes. The $ES-BV$ and the MAP algorithms have similar results and they are not comparable with the other ones in terms of the expected reward.

6.1.6. Fire Fighting Problem

The summary of the results for this problem can be seen in Table 6.6. Once again, the $GA-FSC$ (exact) method has the best performance both in terms of the average and best rewards. While the $GA-FSC$ (approximate) is better than $ES-OH$ in terms of the best reward, the $ES-OH$ approach slightly outperforms it in terms of the average reward. The MAP and the $ES-BV$ approaches have worse results and they have a similar performance in general.

6.1.7. Evaluation of the Expected Reward Performance

According to our experiments, using the $GA-FSC$ algorithm with exact fitness calculation produces the best results in general. The $GA-FSC$ (approximate) gets close to the $GA-FSC$ (exact) when the best rewards are considered, however, the $GA-FSC$ (exact) has certainly better average reward performance with a very low standard deviation in general. The $ES-OH$ has a performance similar to that of the $GA-FSC$ (approximate), however, the $GA-FSC$ (approximate) appears to be better. For small size problems, the $ES-BV$ and the $ES-OH$ have a similar performances. While one is better than other for some cases, the other one is better for some other cases. However, for the Cooperative Box Pushing and the Fire Fighting problem, the $ES-OH$ is much better.

Table 6.5. Expected reward performance of our algorithms for the Cooperative Box Pushing problem.

Horizon	MAP	ES-BV		ES-OH		GA-FSC			
	comm.					Exact		Approximate	
	Best	Best	Average	Best	Average	Best	Average	Best	Average
2	-0.40	-0.40	-0.40 $\mp$ 0.00	17.18	17.15 $\mp$ 0.04	**17.60**	**17.60** $\mp$ **0.000**	**17.60**	**17.60** $\mp$ **0.00**
3	-0.60	-0.60	-0.60 $\mp$ 0.00	22.96	22.91 $\mp$ 0.06	**65.27**	**22.98** $\mp$ **14.86**	64.87	22.68 $\mp$ 14.41
4	-0.80	-0.80	-0.80 $\mp$ 0.00	90.17	90.11 $\mp$ 0.08	**98.17**	**98.13** $\mp$ **0.04**	98.10	94.07 $\mp$ 3.43
8	-1.60	-1.60	-1.60 $\mp$ 0.00	174.68	173.76 $\mp$ 1.30	**189.05**	**188.64** $\mp$ **0.14**	188.60	183.74 $\mp$ 4.30
10	-2.00	-2.00	-2.00 $\mp$ 0.00	210.25	209.07 $\mp$ 0.25	**216.98**	**216.60** $\mp$ **0.15**	216.50	216.49 $\mp$ 0.04
20	-4.00	-4.00	-4.00 $\mp$ 0.00	440.35	361.27 $\mp$ 185.458	**468.15**	**468.15** $\mp$ **0.00**	467.89	467.80 $\mp$ 0.16
50	-10.00	-10.00	-10.00 $\mp$ 0.00	902.21	702.29 $\mp$ 346.27	**1201.04**	**1201.05** $\mp$ **0.00**	1200.22	1199.52 $\mp$ 0.20
100	-20.00	-20.00	-20.00 $\mp$ 0.00	1908.34	1472.68 $\mp$ 754.59	**2420.26**	**2420.26** $\mp$ **0.00**	2418.55	2418.53 $\mp$ 0.06

Table 6.6. Expected reward performance of our algorithms for the Fire Fighting problem.

Horizon	MAP	ES-BV		ES-OH		GA-FSC			
	comm.					Exact		Approximate	
	Best	Best	Average	Best	Average	Best	Average	Best	Average
2	-5.13	-5.48	-5.52 $\mp$ 0.05	**-4.38**	-4.43 $\mp$ 0.03	**-4.38**	**-4.38 $\mp$ 0.00**	**-4.38**	-4.48 $\mp$ 0.08
3	-7.04	-7.38	-7.50 $\mp$ 0.17	-5.89	-5.90 $\mp$ 0.02	**-5.74**	**-5.77 $\mp$ 0.01**	-5.76	-5.94 $\mp$ 0.11
4	-8.38	-8.76	-8.92 $\mp$ 0.23	-6.73	-6.84 $\mp$ 0.09	**-6.58**	**-6.61 $\mp$ 0.02**	-6.63	-6.91 $\mp$ 0.20
5	-10.22	-10.33	-10.66 $\mp$ 0.47	-7.14	-7.18 $\mp$ 0.03	**-7.07**	**-7.09 $\mp$ 0.01**	-7.10	-7.83 $\mp$ 0.53
6	-11.88	-12.01	-12.45 $\mp$ 0.62	-7.69	-7.81 $\mp$ 0.09	**-7.18**	**-7.25 $\mp$ 0.11**	-7.55	-8.52 $\mp$ 0.91

The MAP algorithm clearly has the worst performance among the four for all of the cases.

Having correct information about the current state has a vital importance in making good decisions. FSCs are widely used in the DEC-POMDP literature in order to represent the infinite horizon policies. We think that the success of the $GA - FSC$ algorithm is due to the success of FSCs in representing the current state information close to the actual one. It summarizes the observation history in the FSC nodes. The $ES - OH$ algorithm has a similar approach since it also uses recent observations to make a decision and its success with larger state problems is due to this. The MAP and the $ES - BV$ approaches use belief vectors to represent the state and updating the belief exactly is not possible for the DEC-POMDP model. For large state problems, the effect of this becomes clearer. The $ES-BV$ approach performs better than the MAP, because the MAP produces a strategy with the assumption that the agents can communicate, however they use the strategy in an environment where they cannot communicate. On the other hand, the $ES - BV$ approach produces strategy with no-communication assumption and acts in such an environment, therefore the results are better.

6.2. Run-Time Performance of the Algorithms

We compared performance of our algorithms with the best ones in the literature in terms of the average reward obtained. However, comparing the run-time performance is not straightforward since there are several algorithms that we make comparison with and not all of them report their run-time performances. Additionally, most of the studies do not report their platforms , for this reason, we do not know whether we make the tests using the same processing capabilities with the ones that report run-time performance. For this reason, we chose not to compare the run-time with other algorithms, but, compare the run-time performance of our algorithms with each other. In order to give a clue about the performance of our algorithms, we report the run-time performance of the Multi-access Broadcast Channel problem, representing small size problems, in Table 6.7 and the performance of the Fire Fighting problem, representing large size problems, in Table 6.8. We report the average running time for each horizon. Since, we obtain an infinite horizon so-

Table 6.7. Run time comparison of our algorithms for the Multi-access Broadcast Channel Problem (in seconds).

Horizon	ES-BV	ES-OH	GA-FSC	
			Exact	Approximate
2	522	490	0.56	3.71
4	886	607	0.99	5.12
8	1320	840	1.59	12.06
10	1571	1037	2.11	19.38
20	2691	1760	3.84	33.18
50	5994	4211	12.94	65.99
100	11159	7687	29.7	125.23

lution with ZMDP in the MAP approach, we obtain a single solution and test it on all test horizons. Due to this reason, we do not report these values in the result tables. The run-time of the ZMDP for the Multi-access Broadcast Channel problem is 21 sn and for the Fire Fighting problem run-time is 28 sn. The runs are made on an Intel Xeon 2.50 based platform with 4 GB RAM and the durations are reported in seconds with respect to the horizon.

For the Multi-access Broadcast Channel problem, the $GA-FSC$ approach with the exact fitness calculation is the fastest. The remaining three algorithms are based on simulation and the performance of the decision making method is important. The decision is made in constant time using FSC, therefore, FSC based approach is faster than the belief vector and neural network based approaches. The runtime of the $ES-BV$ and the $ES-OH$ are close to each other, but the $ES-OH$ is slightly faster.

When a large size problem, the Fire Fighting problem, is considered, the exact fitness calculation gets much slower. The $GA-FSC$ approach with the approximate fitness calculation still outperforms ES-based approaches and it is much faster than the exact fitness calculation. The $ES-OH$ is the second best performer and the $ES-BV$ is the worst

Table 6.8. Run time comparison of our algorithms for the Fire Fighting Problem (in seconds).

Horizon	ES-BV	ES-OH	GA-FSC	
			Exact	Approximate
2	23412	187	5004	18
3	40623	243	7890	34
4	48198	303	11887	54
5	63290	482	17583	71

performer.

As a result of these runs, it can be said that the $ES - BV$ approach is the worst in all cases. The $GA - FSC$ approach with the exact fitness calculation can be used in small problems, however, for large problems the run-time gets considerably large. For such cases, approximate fitness calculation can be preferred depending on whether the priority is better policies or better run-times. The performance of the $ES - OH$ approach does not depend on the state count of the problems, therefore it is not effected from the problem size significantly.

7. CONCLUSION

Many real world problems require agent teams to work in a collaborative way and this in turn requires the agents planning their actions accordingly. The DEC-POMDP is a recent model addressing the multiagent planning problem from a decentralized perspective. The agents can develop their strategies in a centralized way, but, they must act without communicating with other agents and the team should have a benefit as a result of agents' actions. Since the DEC-POMDP model has a high computational complexity, solving large problems, like the Fire Fighting problem, for large horizons, beyond 6 for this problem, exactly is not feasible currently due to the high memory and processing requirements. When the complexity of real world problems are considered, approximate solution algorithms that can produce acceptable solutions, depending on the problem context, with less memory and processing requirements should be considered. This thesis focuses on finding approximate and efficient solution techniques to solve finite horizon DEC-POMDP problems.

The MAP algorithm proposes to convert a given DEC-POMDP problem into a POMDP problem as if there is a single agent who can control all the agents in the environment and receive their observations. The obtained POMDP model is solved with a POMDP solver, which is ZMDP in our case, and the obtained strategy is distributed to all of the agents. The POMDP solver assumes that the agent has a belief vector, which is the summary of all past observations, and the POMDP model enables the agent to update its belief perfectly. Therefore, the agents in the DEC-POMDP model should also have a belief vector and be able update it. However, making the belief update perfectly requires the knowledge of the actions and observations of the other agents in the environment, which is not allowed in the DEC-POMDP model. For this reason, we offer an approximate belief update method and made our experiments with this approach. We obtained near optimal results only for the Multiaccess Broadcast Channel problem with that approach. For all other cases, the results were less than desirable. We wanted to see the effect of the communication and tested also this case for all the problems. We see that when the agents can communicate, we can obtain better results than the best ones obtained for the DEC-

POMDP model.

In the MAP approach, we obtain the best policies under the assumption that the agents can communicate. However, when the agents act in the environment, they cannot communicate and this results in synchronization problems. We thought that we can use the same belief approach for decision making, but, we can search for the policy vectors with no-communication assumption. We expected that, if the agents learn their strategy with this assumption, they can act better in the DEC-POMDP environment. We decided to use evolutionary algorithms which are known to work well with optimization problems. We map the belief vectors of the agents to a chromosome and use the ES approach to find a policy since the chromosomes consist of real numbers and we call this approach $ES - BV$. We calculate the fitness of the chromosome, namely the expected reward of the corresponding policy, by making simulations with that policy and by computing the average of the rewards obtained in these simulations. We also use a statistical method to obtain a more conservative fitness value. Our experiments show that this approach helps us to obtain better policies than the MAP approach for all problems, except the Cooperative Box Pushing problem. We have a problem in the approximate belief update using this problem and we obtain invalid belief vectors. Due to this reason, both the MAP and the $ES - BV$ approach failed in this problem. Also, the results with this approach were far from the optimal for most of our test problems.

It can be seen that the success of the belief vector approach mainly depends on the perfect belief updates. However, we cannot update the belief perfectly under the DEC-POMDP constraints. For this reason, we wanted to find alternative ways of making a decision instead of using belief vectors and we decided to use observation histories. Since the belief is a summary of the observation history and recent observations has a greater impact on the current belief, we chose to use the last n observations and feed them into several neural networks each of which corresponds to a different action. We train the neural networks using ES and named this approach as $ES - OH$. The average reward performance of the $ES - BV$ and the $ES - OH$ approaches are similar for small sized problems. For the Cooperative Box Pushing problem, the $ES - OH$ approach produces

much better solutions, since the $ES - BV$ approach has problems with belief update. However, the results are not close enough to the optimal. For the Fire Fighting problem, the $ES - OH$ outperforms the $ES - BV$ and it gets close to the optimal results.

Since we did not obtain completely satisfactory results with some problems using the $ES - OH$ approach, we looked for alternative policy representations. We wanted to use FSCs due to their compact structure compared to policy trees. Since the chromosome values are not real numbers anymore, we decided to use genetic algorithms in order to search the optimal FSC for the given node count and named the method $GA-FSC$. In our previous approaches, we calculated the expected reward of a given policy approximately. Since the expected reward of a given FSC can be calculated exactly, we wanted to test both exact and the approximate fitness calculations. We observe that the $GA - FSC$ approach performs better when the exact fitness calculation is used and we are able to obtain the optimal results for many cases. The main disadvantage of the exact fitness calculation is its scalability with the number of states in the DEC-POMDP model. For example, the algorithm runs almost 200 times faster when approximate fitness calculation is used in the Fire Fighting problem. On the other hand, for small problems exact fitness calculation may run faster. Therefore, the need should be analyzed well. When a small state problem is solved, the exact fitness calculation can easily be used. When solving a large state problem, the optimality expectation is important. If obtaining a high quality solution is the primary concern and sufficient time is available, the exact fitness calculation should be preferred. However, if a quick solution is needed, approximate fitness calculation is a good alternative. The main advantage of the $GA - FSC$ approach is its solution quality and its scalability with the number of horizons. We report solutions for higher horizons than the currently available ones for many problems. The approximate fitness calculation is scalable also with the number of states.

During our study, we concentrated on solving DEC-POMDP problems in the literature. On the other hand, constructing a DEC-POMDP model for every problem is very hard. Using approximate fitness calculation in the $ES - BV$, $ES - OH$ and $GA - FSC$ approaches allow their application to the problems without an exact DEC-POMDP model

when there is a simulator for the given problem. For example, there are several simulators for football environments or search and rescue environments. Instead of using our DEC-POMDP simulator, these simulators can be used to calculate the fitness and all these three algorithms can be used to generate a team strategy in such environments.

The current DEC-POMDP studies concentrate on discrete spaces, however many environments are in fact continuous. There is a huge research area in this subject. As a future work, a formal model and test problems should be defined for the new model should be developed. Then, alternative decision making and learning methods should be tested with it.

REFERENCES

1. Mazurowski, M. and J. Zurada, "Solving decentralized multi-agent control problems with genetic algorithms", *IEEE Congress on Evolutionary Computation*, pp. 1029–1034, 2007.

2. LaValle, S., *Planning Algorithms*, Cambridge University Press, Cambridge, U.K., 2006.

3. Lopez, E., R. Barea, L. Bergasa and M. Escudero, "Visually augmented POMDP for indoor robot navigation", *21st IASTED International Multi-Conference on Applied Informatics*, 2003.

4. Spaan, M. T. J. and N. Vlassis, "A point-based POMDP algorithm for robot planning", *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2399–2404, 2004.

5. Nair, R., D. Pynadath, M. Yokoo, M. Tambe and S. Marsella, "Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings", *Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003.

6. Bernstein, D., S. Zilberstein and N. Immerman, "The Complexity of Decentralized Control of Markov Decision Processes", *Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.

7. "TeamBots (tm)", `http://www.teambots.org/`, 2011.

8. Amato, C., D. Bernstein and S. Zilberstein, "Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs", *Autonomous Agents and Multi-Agent Systems*, Vol. 21, pp. 293–320, 2010.

9. Aras, R. and A. Dutech, "An investigation into mathematical programming for finite

horizon decentralized POMDPs", *Journal of Artificial Intelligence Research*, Vol. 37, pp. 329–396, 2010.

10. Bernstein, D. S., C. Amato, E. A. Hansen and S. Zilberstein, "Policy Iteration for Decentralized Control of Markov Decision Processes", *Journal of Articial Intelligence Research*, Vol. 34, pp. 89–132, 2009.

11. Boularias, A. and B. Chaib-draa, "Exact dynamic programming for decentralized POMDPs with lossless policy compression", *International Conference on Automated Planning and Scheduling*, 2008.

12. Hansen, E. A., D. S. Bernstein and S. Zilberstein, "Dynamic Programming for Partially Observable Stochastic Games", *19th National Conference on Artificial Intelligence (AAAI-04)*, San Jose, CA, 2004.

13. Oliehoek, F. A. and N. Vlassis, "Q-value Functions for Decentralized POMDPs", *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-agent Systems, 2007*, pp. 833–840, 2007.

14. Szer, D., F. Charpillet and S. Zilberstein, "MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs", *21st Conference on Uncertainty in Artificial Intelligence (UAI)*, Edinburgh, Scotland, 2005.

15. Oliehoek, F., S. Whiteson and M. Spaan, "Lossles Clustering of Histories in Decentralized POMDPs", *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pp. 577–584, 2009.

16. Spaan, M. T. J., F. A. Oliehoek and C. Amato, "Scaling Up Optimal Heuristic Search in DEC-POMDPs via Incremental Expansion", *Proceedings of International Joint Conference on Artificial Intelligence*, 2011.

17. Amato, C., D. S. Bernstein and S. Zilberstein, "Optimizing Memory-Bounded Controllers for Decentralized POMDPs", *Proceedings of the 23rd Conference on Uncer-*

tainty in Artificial Intelligence (UAI-07), pp. 1–8, 2007.

18. Amato, C. and S. Zilberstein, "Heuristic Policy Iteration for Infinite-Horizon Decentralized POMDPs", *Proceedings of the Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains (MSDM)*, 2008.

19. Bernstein, D. S., E. Hansen and S. Zilberstein, "Bounded Policy Iteration for Decentralized POMDPs", *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1287–1292, Edinburgh, Scotland, 2005.

20. Boularias, A. and B. Chaib-draa, "Planning in Decentralized POMDPs with Predictive Policy Representations", *ICAPS'08 Multiagent Planning Workshop (MASPLAN'08)*, 2008.

21. Carlin, A. and S. Zilberstein, "Value-based observation compression for DEC-POMDPs", *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 501–508, 2008.

22. Dibangoye, J. S., A. Mouaddib and B. Chai-draa, "Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs", *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 569–576, 2009.

23. Dibangoye, J. S., B. Chai-draa and A. I. Mouaddib, "Policy Iteration Algorithms for DEC-POMDPs with Discounted Rewards", *Eighth International AAMAS Workshop on MSDM*, 2009.

24. Oliehoek, F. A., F. P. Kooij and N. Vlassis, "A Cross-Entropy Approach to Solving Dec-POMDPs", *1st International Symposium on Intelligent and Distributed Computing*, 2007.

25. Oliehoek, F. A., F. P. Kooij and N. Vlassis, "The Cross-Entropy Method for Policy Search in Decentralized POMDPs", *Informatica*, Vol. 32, pp. 341–357, 2008.

26. Oliehoek, F., M. Spaan and N. Vlassis, "Optimal and Approximate Q-value Functions for Decentralized POMDPs", *Journal of Artificial Intelligence Research*, Vol. 32, pp. 289–353, 2008.

27. Seuken, S. and S. Zilberstein, "Memory Bounded Dynamic Programming for Decentralized POMDPs", *20th International Joint Conference On Artificial Intelligence (IJCAI)*, 2007.

28. Seuken, S. and S. Zilberstein, "Improved Memory Bounded Dynamic Programming for Decentralized POMDPs", *23rd Conference on Uncertainty in Artificial Intelligences (UAI-07)*, Vancouver, BC, Canada, 2007.

29. Becker, R., S. Zilberstein and C. V. Goldman, "Solving transition independent decentralized markov decision processes", *Journal of Artificial Intelligence Research*, Vol. 22, pp. 423 – 455, 2004.

30. Nair, R., P. Varakantham, M. Tambe and M. Yokoo, "Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs", *Proceedings of the 20th National Conference on Artificial Intelligence*, pp. 133–139, 2005.

31. Besse, C. and B. Chaib-draa, "Quasi-Deterministic POMDPs and Dec-POMDPs", *Proceedings of 5th International Workshop On Multiagent Sequential Decision Making in Uncertain Domains*, 2010.

32. Kumar, A. and S. Zilberstein, "Constraint-based dynamic programming for decentralized POMDPs with structured interactions", *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '09, pp. 561–568, 2009.

33. Wu, F. and X. Chen, "Solving Large-Scale and Sparse-Reward DEC-POMDPs with Correlation-MDPs", *RoboCup 2007: Robot Soccer World Cup XI*, pp. 208–219, 2008.

34. Wu, F., S. Zilberstein and X. Chen, "Multi-Agent Online Planning with Commu-

nication", *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 321–329, Thessaloniki, Greece, 2009.

35. Wu, F., S. Zilberstein and X. Chen, "Online planning for multi-agent systems with bounded communication", *Artificial Intelligence*, Vol. 175, pp. 487–511, 2011.

36. Jimenez, F., G. Sanchez, P. Vasant and J. L. Vergeday, "A Multi-Objective Evolutionary Approach for Fuzzy Optimization in Production Planning", *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 3120–3125, 2006.

37. Sanchez, G., F. Jimenez and P. Vasant, "Fuzzy Optimization with Multi-Objective Evolutionary Algorithms: a Case Study", *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Multicriteria Decision Making (MCDM)*, pp. 58–64, 2007.

38. Lin, A. Z., J. C. Bean and C. White, *Genetic Algorithm Heuristics for Finite Horizon Partially Observed Markov Decision Problems*, Tech. rep., 1998.

39. Cassandra, A., "A Survey of POMDP Applications", *AAAI Fall Symposium*, 1998.

40. Pynadath, D. and M. Tambe, "The communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories, Models", *Journal of Artificial Intelligence Research*, 2002.

41. Kaelbling, L. P., M. L. Littman and A. R. Cassandra, "Planning and Acting in Partially Observable Stochastic Domains", *Artificial Intelligence*, Vol. 101, pp. 99–134, 1995.

42. Szer, D. and F. Charpillet, "Point-based Dynamic Programming for DEC-POMDPs", *21st National Conference on Artificial Intelligence (AAAI)*, 2006.

43. Amato, C., D. Bernstein and S. Zilberstein, "Optimal Fixed-Size Controllers for Decentralized POMDPs", *AAMAS 2006 Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, 2006.

44. Kube, C. R. and H. Zhang, "Task Modelling in Collective Robotics", *Autonomous Robots*, Vol. 4, pp. 53–72, 1997.

45. Astrom, K., "Optimal control of Markov decision process with incomplete state estimation", *Journal of Mathematical Analysis and Applications*, 1965.

46. Oliehoek, F. A., "Decentralized POMDPs", M. Wiering and M. v. Otterlo (Editors), *Reinforcement Learning: State of the Art*, Springer, Berlin, Germany, 2011, (to appear).

47. Rabinovich, Z., C. V. Goldman and J. S. Rosenschein, "The Complexity of Multiagent Systems: The Price of Silence", *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1102–1103, 2003.

48. Amato, C., J. S. Dibangoye and S. Zilberstein, "Incremental Policy Generation for Finite-Horizon DEC-POMDPs", *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 2–9, 2009.

49. Wu, F., S. Zilberstein and X. Chen, "Point-Based Policy Generation for Decentralized POMDPs", *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, pp. 1307–1314, 2010.

50. Amato, C., "DEC-POMDP Page", `http://rbr.cs.umass.edu/~camato/decpomdp/`, 2011.

51. Smith, T., "ZMDP Software for POMDP and MDP Planning", `http://www.cs.cmu.edu/~trey/zmdp/`, 2009.

52. T. Back, H. S. H., "An overview of evolutionary algorithms for parameter optimization", *Evolutionary Computation*, pp. 1–24, 1993.

53. Akın, H. L., "Evolutionary Computation: A Natural Answer to Artificial Questions",

ANNAL, pp. 41–52, 1994.

54. Holland, J., *Adaptation in Natural, Artificial Systems*, Univ. Michigan Press, 1975.

55. Fogel, I., A. J. Owens and M. Walsh, *Artificial Intelligence Through Simulated Evolution*, John Wiley, New York, 1966.

56. Rechenberg, I., *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Fromman-Holzboog Verlag, Stuttgart, 1973.

57. Koza, J., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press. ISBN 0-262-11170-5, 1992.

58. B. Sendhoff, W. v. S., M. Kreutz, *Causality and the Analysis of Local Search in Evolutionary Algorithms*, Tech. rep., Internal Report IRINI 97-16, Institut fr Neuroinformatik, Ruhr-Universitt Bochum, Germany, 1997.

59. Ignat, D. B., *Genetic Algorithm With Punctuated Equilibria: Analysis of the Traveling Salesperson Problem Instance*, M.S. Thesis, School of Engineering and Applied Science University of Virginia, 1998.

60. Jin, Y., "A Comprehensive Survey of Fitness Approximation in Evolutionary Computation", *Soft Computing*, Vol. 9, No. 1, pp. 3–12, 2005.

61. Wang, F., "Confidence Interval For The Mean Of Non-Normal Data", *Quality, Reliability Engineering International*, Vol. 17, No. 4, pp. 57–267, 2001.

62. Cohen, P. R., *Empirical Methods for Artificial Intelligence*, MIT Press, Cambridge, MA, USA, 1995.