DISTRIBUTED ALGORITHMS BASED ON FICTITIOUS PLAY FOR NEAR OPTIMAL SEQUENTIAL DECISION MAKING

by

Esra Sisikoglu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Industrial and Operations Engineering)
in The University of Michigan
2009

Doctoral Committee:

Associate Professor Marina A. Epelman, Co-Chair
Professor Robert L. Smith, Co-Chair
Associate Professor Hyun-Soo Ahn
Assistant Professor Dushyant Sharma

To my love, Mustafa

ACKNOWLEDGEMENTS

I am heartily grateful to my academic advisors, Marina Epelman and Robert Smith, for their encouragement and guidance from the beginning to the end of my doctoral studies. Professor Epelman's and Professor Smith's energy and enthusiasm for research had motivated me all these years. They are and will always be role models for me through my academic career. I especially would like to thank both of them for providing the financial support during my Ph.D. studies and helping me to get a 50% Graduate Student Research Assistantship (GSRA) appointment during my 4^{th} year to help me concentrate on my research. I also would like to thank them for supporting me on my decision to accept an adjunct position at the University of Missouri in my last year. I am thankful to Professor Smith for signing all the work permit paperwork for my internships. I also thank Professor Epelman for nominating me for Rackham Graduate School travel grant and for lending me her laptop so that we were able to continue our weekly meetings online in my last year. I am grateful to both of my advisors for diligently reading my thesis and going over my all presentations. I have learnt a great deal from their comments and suggestions. Today, considering the last five years my life, I realize once again how lucky I have been to have both Professor Epelman and Professor Smith as my advisors and mentors.

I am also thankful to my committee members, Professor Hyun-Soo Ahn and Professor Dushyant Sharma, for their insightful comments on my dissertation. I greatly

appreciate Professor Ahn's recommendation to look into unigraphs formed by the induced Markov Chain from Markov Decision Processes and suggesting possible journals to submit our work. I also thank Professor Sharma for his insightful comments on the computational complexity of the algorithms developed in my dissertation.

I would like to acknowledge National Science Foundation (NSF) as this work was supported in part by grants CCF-0830092, CMMI-0422752 and CMMI-0244291. I also would like to acknowledge the Graduate Student Instructor (GSI) support I received from the Department of Industrial and Operations Engineering (IOE) and the travel grant I received from Rackham Graduate School.

I am indebted to IOE staff in many ways. Tina Blay, not only helped me with all the questions and problems I had in the last five years, but she also has been the source of joy with her smiles. I remember how talking to her in the mornings immediately changed my mood and made me smile. Matt Irelan helped me with all the graduate school and work permit paperwork. His help during my last year surely saved me several trips between Missouri and Michigan. Mary Winter helped me with all the complications about my Research Assistantship (RA) support because of my decision to move to another state in my last year. Wanda Dobberstein helped me to process my grade for one of the classes I had taken for my Curricular Practical Training (CPT). Liz Fisher and Nancy Murray processed all my time sheets even when I was away and helped me to resolve the problems with my health insurance. Mint Rahaman and Chris Konrad helped me with all my computer/technology related problems. Chris also helped me setup the conference room for my defense presentation. Thank you all very much for your patience and support.

I would like to express my gratitude to Professor Occeña, Professor Noble and Professor Klein from the University of Missouri for their support and encouragement.

I especially thank Professor Occeña for being very understanding and accommodating and agreeing to decrease my teaching load towards the end of my Ph.D. studies. Without his help I would not be able to finish on time. I also thank the staff at the University of Missouri, Sally Schwartz and Paula McDonald, for helping me and my husband to adjust to our new life in Missouri and also for patiently helping us with all the paperwork. Our conversations with Paula and her comforting smiles have definitely decreased my stress level during my doctoral studies.

I owe my deepest gratitude to my parents, Sukran and Hasan Sisikoglu. I would not be able to finish this adventure without their love, support and prayers. Every week, I was looking forward to our phone conversations which gave me the inspiration I needed the most. Mom and Dad: knowing that you were always with me and feeling your constant support has been and will always be invaluable for me and words are not sufficient to show my gratitude. I am also thankful to my brother Selman, sister-in-law Esra and of course little Bertug for enriching my life. I am grateful to my mother-in-law Kerime, father-in-law Mehmet and brother in laws Fatih and Mirac for their prayers and support. I am grateful to all of them for welcoming me as if I had been part of the family all my life and for supporting me during this journey. Special thanks to Mirac for all the SMS messages he sent on my birthdays and on our wedding anniversaries. He did not miss a single occasion so far.

I also would like to express my gratitude for my uncle Hadi Adanali. Following his example, I decided to pursue a Ph.D. degree in US. His dedication to science and his passion for teaching has been my inspiration. I also would like to thank him for all the witty emails he sent me. I will never forget waiting for his emails and impatiently reading them as I knew his emails would make me smile and change my mood when I needed.

I am thankful to our wonderful friends Selen, Burak, Kubra, Serkan, Burcin and Yasin for being our true "dost" and always being there for us. Our rafting trips are surely among the unforgettable memories of my Ph.D. years. I also will never forget the friendship of Burde, Melih, Yasemin, Mahmut, Tubanur and Seyda. Our paths have crossed once in Ann Arbor and I sure hope they will cross again in the future. I am also thankful to Irina with whom I shared a lot of memories as office mates, as neighbors and as classmates. I will never forget all the wonderful time I spent with Sarah, Shankara, Betzabe, Raul, Fernando, Eren, Blake, Tim, Stan, Ada, Archis, Tara, John, Iva, Oben, Gayu, Reza, Damon, Shervin, Pierre and Joy. An extra thanks to Shervin for being a mentor to me in my first semester of GSI experience. I had to spend my last year away from these great friends; however, I was also lucky to meet new great friends in Missouri: Nilufer, Muserref, Mehmet, Lale, Munevver, Sebnem, Sinan and Ayse. Thank you all for your kindness and warmth. Muserref Abla and Mehmet Abi, thank you very much for calling me every day before my defense and for all your support and prayers.

The last but not the least, I feel indebted to our dear friends Suleyman and Emel and their lovely daughter Zeynep for hosting us for one month in Ann Arbor before my defense. I will never forget the dinners we prepared together, the Turkish soap operas we watched and most importantly the long heart-warming conversations we had during that month. I am also grateful to Suleyman for listening to my defense presentation and making the suggestions which significantly improved my presentation.

Finally, my love, the meaning of my life, my husband, Mustafa. Thank you very much for always being there for me, for trusting me, for believing in me, for supporting me, for teaching me to be patient and pious, for making my life joyful

and rich, for making me feel loved, for cheering me up, for putting up with my frustrations, for sparing me your shoulder every time I needed you, for holding my hand and for promising to hold it forever. I love you.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

CHAPTER I

Introduction

Development of the theory of Dynamic Programming (DP) started following the period after World War II when it was realized that there were a large number of practical problems that can be modeled as multistage (sequential) decision making problems [11]. One important aspect of sequential decision making problems is that a decision with a low immediate cost (*i.e.*, the cost of the decision at the present time) might lead to situations where decisions available in the future are costly. Therefore, the decisions at different time periods cannot be viewed in isolation. DP methods take the effect of current decisions on the future states of a system into account by finding a decision at each time period that minimizes the sum of the immediate cost and the "cost-to-go" (*i.e.*, expected cost that will be accrued in the future) [12].

Well known examples of DP applications include inventory control [40], machine replacement, resource allocation [11] and job scheduling [12]. DP techniques not only break such complex problems into smaller and simpler subproblems but also make it possible to model complex constraints and objective functions which would be hard to handle with other optimization models such as linear programming [38]. Despite all the advantages that DP techniques offer, computationally, they are not effective on large-scale problems due to "curse of dimensionality." Curse of dimensionality is

a term, first used by Bellman [9], that refers to the exponential increase in the state space as the dimensionality of the problem increases. For example, an inventory control problem of a single product with storage capacity of K units can be represented by a DP model with $K \times T$ states. However, the DP model of an inventory control problem of N products will have $O(K^N \times T)$ states. Powell extends the notion of curse of dimensionality to include the growth of the state space, the outcome space and the action space as the dimensionality of the problem increase and refers to it as "three curses of dimensionality" [39]. As a result of the curse(s) of dimensionality, the computational requirements to solve DP problems increases tremendously as the dimensionality of the problem increases. In Chapter II of this thesis, we will discuss this issue in greater detail.

DP models cover a wide of range of problems with different model characteristics such as: discrete/continuous state space and decision space, discrete/continuous time, finite/infinite time horizon or deterministic/stochastic problem data. **Our contribution** in this well studied area is to design computationally efficient new algorithms using game theory concepts for specific classes of large-scale DP problems. We mainly focus on the following two subclasses of problems:

- Deterministic, discrete and finite time, large-scale DP problems.
- Stochastic, infinite horizon, discounted DP problems (also called Markov Decision Processes or MDPs).

The first subclass of problems have traditionally been solved with label correcting algorithms such as Dijkstra's Algorithm [20], which lose their effectiveness for large-scale problems. We propose an alternative approach in Chapter II to find "good" (if not optimal) solutions very quickly. In addition, the algorithms developed in

Chapter II are also applicable to problems in which the costs of decisions are not readily available but can be obtained via complex calculations or simulations.

The second subclass of problems, namely MDPs, are studied in Chapter IV. The small-scale MDPs can be effectively solved with methods such as Value Iteration and Policy Iteration [13]. For large-scale and complex MDPs, several methods such as Approximate Dynamic Programming [39] and Neuro-dynamic Programming [13] have been proposed. Most of these methods, however, assume that the probability distribution governing the random disturbance in an MDP problem is known. In many practical applications this assumption may be invalid. The decision maker may have a limited information about a system and may have to learn the system dynamics through observation. Therefore, in addition to determining optimal decisions, one needs to estimate the unknown parameters in the system. We achieve this (*i.e.*, simultaneously learn and optimize a system) by using game theory concepts. We compare our approach with other approaches from the Reinforcement Learning literature reviewed in Chapter IV.

The idea of using game theory concepts in an optimization framework is motivated by the convergence results obtained for an algorithm called Fictitious Play (FP). FP is a learning algorithm, at each step of which players compute their best replies based on the assumption that their opponents' decisions follow a probability distribution in agreement with the historical frequency of their past decisions [16]. FP has been proven to converge to Nash Equilibria [36] on identical interest games [35]. Nash Equilibrium (NE) is a strategy from which none of the players can unilaterally deviate and obtain a better payoff. By modeling an optimization problem as a game (see Chapter II for details) and defining the objective function of the optimization problem to be the identical payoff each player receives, it is easy

to see that the optimal solution of the optimization problem is also a NE of the game. Therefore concepts from FP can be used to develop algorithms to find the optimal solution of an optimization problem. This idea was first studied by Lambert *et al.* for solving an optimization problem with box constraints [33]. To improve the computational efficiency of FP, they implemented a sampling scheme and named this modified algorithm Sampled Fictitious Play (SFP). The convergence results of SFP, proven by Lambert *et al.* , and its successful applications to various problems such as traffic routing [24], coordinated traffic signal control [19], manufacturing [6] and network flow problems [23] are what motivated us to use a similar approach to solve DP problems.

Our contribution in combining SFP with DP are two fold. First of all, we provide several SFP based algorithms to quickly find good solutions for large-scale, deterministic, finite horizon DP problems. Further, we show that our algorithms converge to the optimal solution of the DP problem. We also present promising empirical results on the Traveling Salesman Problem (see Chapter III) and provide possible modifications that can improve the performance of the algorithms.

Our second contribution is in the field of Markov Decision Processes. We use SFP concepts to design an efficient online learning algorithm that simultaneously learns the system dynamics and finds the optimal solution of a given MDP. We prove that the algorithm converges to the optimal solution and empirically show that the convergence rate of our algorithm is less sensitive to the discount rate parameter given in a problem (compared to other learning algorithms such as Q-Learning and SARSA).

The dissertation is organized as follows: In Chapter II, we discuss the details of the Fictitious Play and Sampled Fictitious Play algorithms and present our motivation

for designing SFP based algorithms to solve deterministic, discrete, large-scale DP problems. In particular, we present two different algorithms, prove their convergence to an optimal solution, and compare their performances on deterministic inventory control problems. In Chapter III, we provide a brief literature review of the Traveling Salesman Problem (TSP) and present the numerical results obtained from applying the two algorithms developed in Chapter II to TSP. Furthermore, we discuss possible modifications to improve the performance of these algorithms. In Chapter IV, we provide a literature review of MDPs, with a focus on discounted, infinite horizon problems and discuss how SFP can be applied to such problems. We also highlight the differences between the algorithms that have been studied in the literature and our algorithm. Following the presentation of the theoretical convergence results, we provide several numerical experiments using the windy gridworld and dynamic location problems as examples. Finally, in Chapter V we conclude and discuss future research directions.

CHAPTER II

Sampled Fictitious Play Based Stochastic Search Algorithms for Dynamic Programming Problems

2.1 Introduction

Sequential (or multi stage) decision problems have been extensively studied in the Operations Research literature. Shortest path, knapsack and dynamic lot sizing problems are just a few examples of sequential decision problems which have been solved using dynamic programming techniques. Dynamic programming is a useful mathematical technique for making a sequence of interrelated decisions. The main advantage of dynamic programming is that it divides a large problem into smaller subproblems which are easier to solve. However, due to the curse of dimensionality, a phenomenon first described by Bellman [9], the practical applications of dynamic programming are somewhat limited; they involve certain problems in which the cost-to-go function has a simple analytical form or problems with a manageable state space [45]. To overcome this problem, we propose new algorithms based on concepts of Fictitious Play (FP) and Game Theory.

Fictitious Play is an iterative procedure designed to mimic the behavior of players engaged in a repeated game. At each step of SFP, players compute their best replies based on the assumption that their opponents' decisions follow a probability distribution in agreement with the historical frequency of their past decisions [16]. Monderer

et al. showed that historical frequencies of decisions made by players in Fictitious Play converge to Nash equilibrium for identical interest games [35]. Lambert *et al.* extended the work of Monderer *et al.* by proving the convergence of Sampled Fictitious Play (SFP) in which players compute their best replies to a set of strategies they sample from the histories of prior plays [33].

Sampled Fictitious Play algorithms have been successfully applied to several large-scale optimization problems such as computing optimal routings in a dynamic traffic network [24], finding efficient coordinated signal timing plans for large number of traffic signals [19], finding the optimal decisions in a manufacturing system [6] and optimization in network flow problems [23]. In these examples, the proposed SFP based algorithms are designed to solve a specific optimization problem. However, in this chapter, we use SFP to provide a general framework to solve any generic problem that can be modeled as a DP. Such problems arise in many important contexts such as resource allocation [20], production scheduling [30], vehicle routing [34] and equipment replacement [8], [7]. Therefore, our algorithms can be used effectively to solve a large variety of problems.

In order to apply equilibrium concepts of Game Theory and Fictitious Play to DP problems, we first need to define players, actions, and payoff/cost parameters of a game that represents the DP problem under consideration. In this chapter we concentrate on discrete DP problems that can be represented as a network such that nodes and arcs correspond to states and decisions available in each state, respectively. Moreover, the length of each arc is defined to be the cost/payoff of the decision it represents. We define an identical payoff game on this network assuming the players and actions available to each player are represented by nodes and arcs emanating from each node, respectively. The identical payoff/cost that players receive by play-

ing a certain action is the total length of the path that is formed by the arcs that correspond to the actions of all players. We introduce the concept of *local optimal solution* of a DP as the collection of actions that form an equilibrium solution of the game. Equilibrium of a game, a notion introduced by Nash [36], is the collection of actions corresponding to each player such that no player has incentive to unilaterally deviate from his action, given the actions of the other players. Based on this definition, it is easy to see that the globally optimal solution of the DP network is also a Nash equilibrium of the defined identical payoff game. Therefore, in theory, one can enumerate all the local optimal solutions of DP network and identify the globally optimal solutions. In this chapter we propose two algorithms called Repeated Sampled Fictitious Play (RSFP) and Sampled Fictitious Play Based Local Search (SFPLS) that find the globally optimal solutions. RSFP identifies a globally optimal solution by *intelligently* enumerating local optimal solutions such that in each run an equilibrium point with the same or lower objective value (higher objective value if it is maximization problem) then the current solution is identified. SFPLS, on the other hand, is a modified version of RSFP and it finds the optimal solution without identifying the local optimal solutions. In the remainder of the chapter we will use the term "action" for the decision of a single player and the term "strategy" to refer to the collection of actions of all players.

The convergence of SFP algorithms to a globally optimal solution on staged DP networks was first analyzed by Ghate *et al* [26]. They designed an SFP based algorithm in which the players have the flexibility of sampling from their set of all feasible actions instead of just sampling from their history of past plays, with a positive probability. As a result, they have proved that their SFP algorithm converges to a globally optimal solution with probability one.

Our approach differs from that of Ghate *et al* in the following ways:

1. We restrict players to sample only from their history of past plays. In doing so, we aim at identifying the Nash Equilibrium (local optimal) strategies very quickly.

2. We use a multi-start approach in which the SFP algorithm is restarted with different initial strategies, whereas Ghate *et al.* 's algorithm finds globally optimal strategies in one run. By restarting the SFP, we obtain "improving" local optimal strategies which eventually converge to the globally optimal solution.

Our motivations for applying SFP based algorithms to large-scale DP problems are as follows:

- SFP does not depend on the structure of the DP network or the type of the objective function. Therefore, such algorithms can be used for solving black-box optimization problems that require simulation of a system or computationally intensive calculations to evaluate the value (or cost) of different decisions.

- With an SFP based approach, improving solutions are obtained through relatively simple best reply calculations. Therefore, instead of using complex algorithms requiring intensive calculations, we aim at finding the solutions that are close to the optimum via simpler (and therefore faster) calculations. Moreover, these calculations can easily be parallelized and therefore the running time of these algorithms can be significantly reduced.

- The best reply calculations of SFP algorithms only require the knowledge of a small portion of the system. For example, to calculate the best reply for a player only the strategies of the relevant players need to be known. Since SFP based

algorithms do not require the knowledge of the entire system initially, they can be easily designed to optimize and learn (or discover) the system simultaneously.

- In SFP, players calculate best replies based on their beliefs about what other players are playing. These beliefs are formed using the historical data on the previous observations in the system. Therefore, our approach can be extended to solve stochastic DP problems by generating the beliefs that take into account the randomness within the system.

The chapter is organized as follows. In the next section the basic problem is introduced and notation is defined. In Section 2.3 the SFP algorithm is formally defined and then in Section 2.4 two new algorithms that use SFP to find globally optimal solutions are presented. Finally, numerical results are discussed in Section 2.5. Section 2.6 concludes the chapter.

2.2 Problem Formulation

We first present the notation and basic problem formulation:

- $(\mathcal{N}, \mathcal{A})$ is a directed acyclic network, where
  - $\mathcal{N}$ is the set of nodes and $n = |\mathcal{N}| < \infty$ is the number of nodes, and
  - $\mathcal{A}$ is the set of arcs.
- If there exists an arc from node i to node j, we denote it by the ordered pair (i, j). That is,

$$\mathcal{A} \equiv \{(i, j) : i, j \in \mathcal{N} \text{ and } \exists \text{ an arc from node } i \text{ to node } j\}.$$

- For a node $i \in \mathcal{N}$, $\mathcal{A}_i$ is the set of arcs emanating from that node:

$$\mathcal{A}_i \equiv \{(i, j) : j \in \mathcal{N}, (i, j) \in \mathcal{A}\}.$$

- c_{ij} is the cost of the arc $(i, j) \in \mathcal{A}$.
- We assume, without loss of generality, that there is a unique root node, denoted by $r \in \mathcal{N}$: $\{i \in \mathcal{N} : (i, r) \in A\} = \emptyset$.
- $\mathcal{N}_l$ is the set of leaves. That is,

$$\mathcal{N}_l \equiv \{i : i \in \mathcal{N} \text{ and } \mathcal{A}_i = \emptyset\}.$$

- **Connectivity Assumption:** There exists a sequence of nodes that connects the root node to any other node in the network $(\mathcal{N}, \mathcal{A})$, i.e. $\exists(r, i_1, \ldots, i_m, j)$, $\forall j \in \mathcal{N}$ such that,
  - $m \in \mathbb{Z}_+$
  - $(r, i_1), (i_{k-1}, i_k)$ and $(i_m, j) \in \mathcal{A}$, $\forall k = 2, \ldots, m$.
- A path is a sequence of nodes, starting with the root node r and ending at one of the leaves, of the form $p = (i_1, i_2, \ldots, i_m)$, where
  - m is some positive integer specific to each path,
  - there exist an arc $(i_{k-1}, i_k) \in \mathcal{A}$ for $k = 2, 3, \ldots, m$, and
  - $i_1 = r$ and $i_m \in \mathcal{N}_l$.

  Since in this chapter we consider directed acyclic networks, root node r exists, $\mathcal{N}_l \neq \emptyset$ and nodes in a path are not repeated. In the remainder of the chapter, we use the term "path" to refer to "directed path". Therefore, the sequence of nodes in a path is sufficient to define the path, and this sequence also provides the information on the sequence of arcs that form the path.
- $\mathcal{P}$ is the set of all paths.

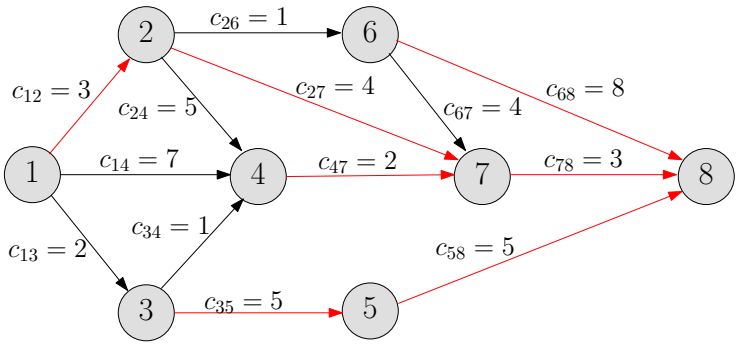


Figure 2.1: Acyclic Network Example

- The cost of a path $p = (i_1, i_2, \ldots, i_m)$ is given by

$$c(p) = \sum_{k=2}^{m} c_{i_{k-1} i_k}.$$

Finding the optimal solution on a deterministic, discrete DP network can be viewed as a deterministic shortest-path problem on a finite acyclic directed network which can be formulated as:

$$\begin{aligned} \min_{p} \quad & c(p) \\ \text{s.t.} \quad & p \in \mathcal{P}. \end{aligned} \tag{2.1}$$

2.3 Sampled Fictitious Play (SFP) Algorithm

The shortest-path problem on an acyclic network described above can be viewed as a strategic game with identical payoff by considering the nodes as players and the arcs emanating from each node as the actions available to that player. In this context, $\mathcal{N}$ represents the set of players and $\mathcal{A}_i$ represents the set of feasible actions available to player $i \in \mathcal{N}$.

Let $a_i \equiv (i, j) \in \mathcal{A}_i$ denote an action played *i.e.*, arc chosen, by player $i \in \mathcal{N}$. Let $\mathcal{S} = (a_1, a_2, \ldots, a_n)$ represent an n-tuple of actions played by all players. $\mathcal{S}$ defines a unique path $p \in \mathcal{P}$ from the root node r to one of the leaves due to the connectivity

assumption [26]. We define the negative of the cost of this path, $c(p)$, to be the identical payoff [1] that each player receives. Therefore, the payoff player i receives by playing the action a_i depends not only on a_i but also the actions played by all other players.

For example, Figure 2.1 shows an acyclic network with 8 nodes/players. The cost of each arc/action is shown next to it. Node 1 is the unique root node and node 8 is the only leaf node of the network. The red arcs in the figure represent the actions played by each player. Note that, the actions played by players 1, 2 and 7 define a unique path from the root node 1 to the leaf node 8. The cost of this path is 10, which is the payoff that all players including ones not on this path receive.

In this chapter, we analyze a Sampled Fictitious Play (SFP) algorithm for identifying the Nash equilibrium (NE) strategies of the identical payoff game described above. A NE strategy is a strategy such that such that no player can unilaterally switch to another action and obtain a lower cost. In other words, no improvement can be obtained via local changes. Therefore, we refer to a NE strategy as a *local optimal solution* of the underlying shortest-path problem. On the example in Figure 2.1, the red arcs represent a strategy that forms a Nash equilibrium with a payoff of 10. If player 1 deviates and plays (1,3) or (1,4), the cost of the path will increase to 12. Thus, player 1 cannot unilaterally deviate and obtain a lower payoff. Since this is true for all other players, the strategy represented by red arcs is a Nash equilibrium. It is important to note that a local optimal strategy is not necessarily globally optimal. For example, in Figure 2.1, the global optimal path with a cost of 8 is formed by the arcs (1,3), (3,4), (4,7) and (7,8).

[1] In this chapter, we set up the optimization problem to minimize the cost of the path to be consistent with the standard "shortest-path" problem, however, in the game theory literature the players maximize the payoff they receive. To be consistent with the game theory terminology we defined the negative of the path length to be the payoff of the game defined on the network.

Each iteration of SFP Algorithm is composed of *sampling an action, calculating a best reply* and *updating the history* for each player. At the beginning of each iteration all players sample an action from their corresponding histories. As mentioned before, the list of sampled actions (*i.e.*, arcs) defines a path. It is important to differentiate between *on-the-path* and *off-the-path* players, since the arcs selected by the latter do not contribute to the identical payoff (the path length) of the game. Therefore, assuming everybody is playing the action they sampled at the beginning of an iteration, *on-the-path* players calculate their best replies and these best reply actions are added to their history of past plays. *Off-the-path* players, on the other hand, do not make any best reply calculations and their histories are updated with the latest entry from their histories. In other words, since they are indifferent to the action they play, the action they played the last time they were on a path is chosen as the best reply for them. Finally, if the history size of any player exceeds a predefined fixed value, say L, then the earliest entry in the history is deleted. As we will explain in the subsequent sections, a fixed finite history size is necessary to prove the convergence of the algorithm.

The following notation is used to describe the algorithm:

- $x_i^k = (i, j) \in A_i$ is the action sampled by player i in iteration k.
- $\mathcal{S}^k = (x_1^k, x_2^k, \ldots, x_n^k)$ is the n-tuple of actions sampled by all players in iteration k.
- $p(\mathcal{S}^k) = (i_1^k, i_2^k, \ldots i_{m(k)}^k)$ is the sequence of nodes that form the unique path defined by $\mathcal{S}^k$. The components of $p(\mathcal{S}^k)$ are referred to as *on-the-path players*. For example, in Figure 2.1, if the red arcs are the sampled actions, then the players 1, 2, 7 and 8 are the *on-the-path players*, whereas the players 3, 4, 5

and 6 are the *off-the-path* players. In an abuse of notation, we use $i \in p(\mathcal{S}^k)$ to indicate the player i is on the path formed by $\mathcal{S}^k$.

- $y_i^k = (i, j) \in A_i$ is the action played by player i in iteration k, *i.e.*, the best reply.
- $\mathcal{H}_i^k$ is the history of past plays of player i at iteration k.
- L is the size of history kept in the memory for each player.

In the next two sections, we formally define the SFP algorithm and prove that it converges to a NE, and therefore to a locally optimal solution of the underlying shortest-path problem.

2.3.1 SFP Algorithm

Initialization

- Set $k \leftarrow 0$
- Randomly sample an action $x_i^0 \in A_i$ for each player i.
- Set $y_i^0 = x_i^0 \ \forall i \in \mathcal{N}$ and add it to the history, *i.e.*, $\mathcal{H}_i^1 = [y_i^0]$ (*i.e.*, the histories of all players are initialized with a random action).
- Set $k \leftarrow k + 1$.

Iteration $k \geq 1$

Step 1 Sample:

Step 1a. For each player i, sample an action $x_i^k \in A_i$ from the history $\mathcal{H}_i^k$ uniformly at random.

Step 1b. Set $\mathcal{S}^k = (x_1^k, x_2^k, \ldots, x_n^k)$.

Step 1c. Identify the unique path, $p(\mathcal{S}^k) = (i_1^k, i_2^k, \ldots, i_m^k)$, defined by $\mathcal{S}^k$.

Step 2 Play:

For each player i,

- If $i \in p(\mathcal{S}^k)$, player i plays the best reply strategy calculated by

$$y_i^k = \arg\min_{x \in A_i} c(x_1^k, x_2^k, \ldots, x_{i-1}^k, x, x_{i+1}^k, \ldots, x_n^k).$$

  If there are multiple actions that minimize the function above (*i.e.*, multiple best replies), choose one according to a lexicographic tie breaking rule. To elaborate further, suppose that all the arcs (or actions) are indexed either arbitrarily or according to a certain rule. If there are two or more actions which result in the same minimum cost, then choose the one with lowest index number.

- If $i \notin p(\mathcal{S}^k)$, player i plays the last entry in his history, *i.e.*,

$$y_i^k = y_i^{k-1}.$$

Step 3 Update History:

- Add the played action to the history: $\mathcal{H}_i^{k+1} = [\mathcal{H}_i^k; y_i^k]$.[2]

Step 4 Termination Criteria:

- If $1 \leq k \leq L$ and $y_i^0 = y_i^1 = y_i^2 = \ldots = y_i^k$ for all $i \in \mathcal{N}$, then terminate.
- If $k \geq L + 1$ and $y_i^{k-L} = y_i^{k-L+1} = y_i^{k-L+2} = \ldots = y_i^k$ for all $i \in \mathcal{N}$, then terminate.
- Otherwise
  - If the size of $\mathcal{H}_i^{k+1} > L$, then delete the earliest entry of $\mathcal{H}_i^{k+1}$.

[2] $[\mathcal{H}_i^k; \cdot]$ denotes the concatenation function.

– Set $k \leftarrow k+1$

– Go to *Step 1.*

2.3.2 Convergence of SFP to a Nash equilibrium

Without loss of generality, we assume that the network $(\mathcal{N}, \mathcal{A})$ has a unique sink node and that the nodes in the network are numbered in such a way that, if an arc from node i to node j exists then $j > i$. Under these assumptions, node n is the unique sink node and node $n-1$ is connected only to node n. Thus, player $n-1$ has only one feasible action and it always samples and plays this action. For example, in Figure 2.1, player 7 will always sample and play the action represented by arc $(7, 8)$.

Below, we prove that the SFP algorithm converges to a NE in finite time.

Theorem II.1. *There exists an n-tuple of actions* $\mathcal{S}^* = (a_1^*, a_2^*, \ldots, a_n^*)$ *and* $M \in \mathbb{Z}_+$, $M < \infty$ *such that*

$$Prob\{x_i^k = a_i^*\} = 1 \quad \forall k \geq M,\ \forall i = 1, 2, \ldots, n,$$

and the n-tuple of actions $\mathcal{S}^* = (a_1^*, a_2^*, \ldots, a_n^*)$ *is a Nash equilibrium (i.e., a local optimal solution) of the game defined on the acyclic network* $(\mathcal{N}, \mathcal{A})$ *that represents a finite horizon DP problem.*

Proof. At the sampling step of the SFP algorithm (*Step 1*), a sample is drawn from the histories of players, $\mathcal{H}_i^k$, $i = 1, 2, \ldots, n$. Therefore, to prove the first part of the theorem, it is sufficient to show that $\exists M < \infty$ such that:

$$Prob\{H_i^k = [a_i^*, \ldots, a_i^*]\} = 1, \quad \forall k \geq M,\ \forall i = 1, \ldots, n. \tag{2.2}$$

Suppose an infinite sequence of actions for each player i, denoted by T_i, is obtained by running the SFP indefinitely without the termination criteria. Since player $n-1$

always plays the only action available to him, we have

$$T_{n-1} = (y^0_{n-1}, y^0_{n-1}, \ldots) \tag{2.3}$$

$$H^k_{n-1} = [y^0_{n-1}, \ldots, y^0_{n-1}] \tag{2.4}$$

where $y^0_{n-1} = (n-1, n)$. Thus letting $a^*_{n-1} = y^0_{n-1}$,

$$Prob\{H^k_{n-1} = [a^*_{n-1}, a^*_{n-1}, \ldots, a^*_{n-1}]\} = 1, \quad \forall k \geq M_{n-1} = L$$

Now assume that $\exists M_i < \infty$ such that,

$$Prob\{H^k_i = [a^*_i, \ldots, a^*_i]\} = 1, \quad \forall k \geq M_i, \ \forall i = j+1, \ldots, n.$$

This implies that the actions sampled by the players $j+1, j+2, \ldots, n$ will be fixed after iteration $M' = \max\{M_{j+1}, M_{j+2}, \ldots, M_{n-1}\}$. Then, one of the following statements holds for player j:

Case 1: player j is never on a sampled path after iteration M' and he plays the last entry in his history for all iterations $k \geq M'$,

Case 2: player j becomes *on-the-path* for the first time in an iteration $k' \geq M'$, he plays the best reply action, denoted by $\bar{a}_j$, to the (fixed) strategies of players $j+1, j+2, \ldots, n$, in all iterations $k \geq k'$.

Therefore, the following holds for the T_j:

$$T_j = \begin{cases} (y^{M'-1}_j, y^{M'-1}_j, \ldots) & \text{if player } j \notin p(\mathcal{S}^k) \quad \forall k \geq M' \\ (y^{M'-1}_j, y^{M'-1}_j, \ldots, \bar{a}_j, \bar{a}_j, \ldots) & \text{otherwise} \end{cases}$$

This implies:

$$Prob\{H^k_j = [a^*_j, \ldots, a^*_j]\} = 1, \quad \forall k \geq M_j$$

where $a^*_j = y^{M'-1}_j$ and $M_j = M' + L$ or $a^*_j = \bar{a}_j$ and $M_j = k' + L$.

Therefore, by induction, theorem holds for all players $1, \ldots, n$ with

$$M = \max\{M_1, M_2, \ldots M_{n-1}\}.$$

This proves the first part of the theorem. To prove the second part, note that equation (2.2) implies $\exists M \in \mathbb{Z}_+$ and $\exists \mathcal{S}^* = (a_1^*, \ldots, a_n^*)$ such that,

$$Prob\{x_i^k = a_i^*\} = 1, \quad k \geq M \quad \forall i = 1 \ldots n$$

which in turn implies

$$Prob\{y_i^k = a_i^*\} = 1, \quad k \geq M \quad \forall i = 1 \ldots n.$$

Thus, at iteration $M + L + 1$ every player i will sample and play the action a_i^*. This implies that a_i^* is the best reply of player i when all other players play a_j^* , $j \neq i$, thus player i cannot obtain a lower cost strategy by unilaterally deviating from the action a_i^* and this is true for all players. Therefore, $\mathcal{S}^*$ is a Nash equilibrium.

Note that, in iterations $k \geq M + L + 1$ all players sample and play the same strategy. Thus, the algorithm can be terminated in finite time. □

2.4 Using SFP to Find an Optimal Solution

A DP network may have many Nash Equilibrium strategies, the path lengths of which are not necessarily close to the optimal path length. One way to improve the quality of solutions found by SFP is to run SFP several times with randomly initialized strategies and identify the "good" NEs. Figure 2.2 shows the histogram of the equilibrium strategies returned by the SFP algorithm for an inventory control problem, the details of which will be given in section 2.5, by restarting the algorithm with random strategies 2000 times.

As it can be seen from the histogram in Figure 2.2, in many cases a NE is far from the optimal solution as measured by the objective function value. Large number of

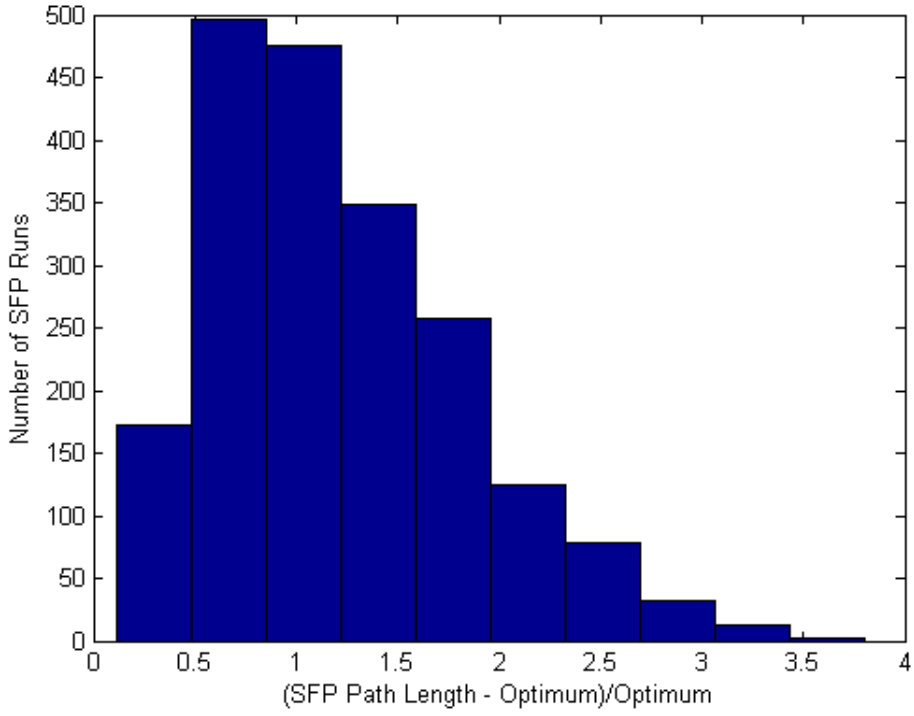


Figure 2.2: Histogram of NE path lengths returned by SFP for an inventory control problem. SFP is run 2000 times with random initial strategies.

NEs makes it difficult to identify the optimal solution by enumerating all NEs. In the next section, we show how the SFP algorithm can be recursively used to *learn* the dynamics of the problem and find solutions that are closer to the optimum.

2.4.1 Repeated Sampled Fictitious Play (RSFP)

In this section, we develop an algorithm referred to as Repeated Sampled Fictitious Play (RSFP) that runs SFP in a repeated fashion and uses the information gathered from each SFP run in the subsequent iterations in order to find NEs with shorter path lengths than the ones discovered in previous iterations. Note that, each iteration of RSFP is a full run of SFP.

In the initialization phase of RSFP (*i.e.*, the first run of SFP), a NE is obtained by running SFP with randomly initialized set of actions. Next, *on-the-path* and *off-the-*

path players from this NE are identified. In the iterations following the initialization phase (*i.e.*, the subsequent SFP runs), an SFP algorithm is run with initial strategies determined as follows:

- If a player is *on-the-path* in the NE from the previous iteration, add his action to the n-tuple of initial actions.
- If a player is *off-the-path* in the NE from the previous iteration, randomly sample an action for this player and add this sampled action to the n-tuple of the initial actions.

Once the initial strategy is determined, regular steps of SFP are carried out. Intuitively, RSFP tries to find a NE with a shorter path length by improving the previous one via best reply calculations.

The following notation is used in the definition of RSFP:

- $\mathcal{E}^t$ is the NE returned at the end of iteration t.
- $p(\mathcal{E}^t)$ is the list of players that are *on-the-path* defined by the equilibrium strategy $\mathcal{E}^t$.
- $\mathcal{E}^t(i)$ denotes the action of player i in $\mathcal{E}^t$.

RSFP Algorithm

Initialization:

- Randomly sample an action $x_i \in A_i$ for each player i.
- Run SFP with the randomly sampled actions and obtain a NE strategy, denoted by $\mathcal{E}^0$.
- Set $t \leftarrow 1$

Iteration $t > 0$:

Step 1 Initialize $\mathcal{S}$ as follows:

For each player i,

- If $i \in p(\mathcal{E}^{t-1}),\ \mathcal{S}(i) = \mathcal{E}^{t-1}(i)$.
- If $i \notin p(\mathcal{E}^{t-1}),\ \mathcal{S}(i) = x_i$, where x_i is a randomly sampled action from A_i

Step 2 Run SFP with the initial strategy $\mathcal{S}$ to obtain a NE strategy $\mathcal{E}^t$.

Step 3 Set $t \leftarrow t + 1$, and go to *Step 1*.

Convergence of RSFP to an Optimal Solution

In this section, we prove that RSFP converges to an optimal solution when the history size L used in each SFP run is 1.

Lemma II.2. *When $L = 1$, $c(p(\mathcal{E}^t)) \geq c(p(\mathcal{E}^{t+1})),\ \forall t$.*

Proof. When $L = 1$, in each iteration of SFP, players calculate a best reply to the most recent actions played by all other players. Thus, in each SFP run of RSFP a new NE is obtained only when it has shorter path length than the previous one, or it has the same length. The latter may happen if one of the players deviates to an action with a lower index (due to lexicographic tie breaking rule) even though the path length is the same. □

Lemma II.2 shows that RSFP will either improve or stall in each SFP run. Next, we prove that if the current NE is not globally optimal, RSFP will eventually find a better NE.

Lemma II.3. *Let $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ be a set of actions and $p(\mathcal{S})$ be the list of* on-the-path *players defined by $\mathcal{S}$. Also let $\mathcal{S}_{p(\mathcal{S})}$ denote the actions from $\mathcal{S}$ corresponding*

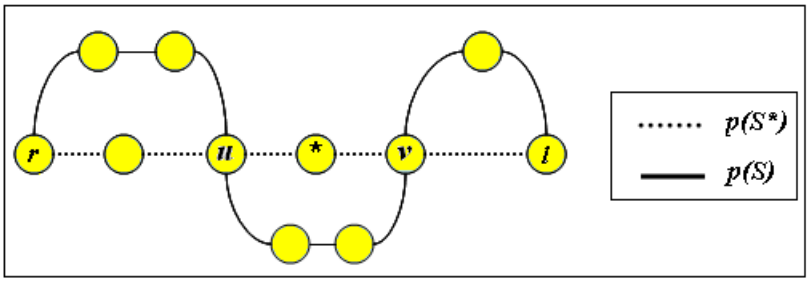


Figure 2.3: An illustration for the proof of Lemma II.3.

to on-the-path *players. If* $c(p(\mathcal{S}))$ *is not optimal, then there exists a list of actions for* off-the-path *players, denoted by* $\mathcal{S}'_{N\setminus p(\mathcal{S})} = \{s'_j | j \notin p(\mathcal{S})\}$*, such that the best reply of at least one of the* on-the-path *players to the strategy* $\mathcal{S}_{p(\mathcal{S})} \cup \mathcal{S}'_{N\setminus p(\mathcal{S})}$ *is to deviate to obtain a* ***strictly*** *better (shorter) path.*

Proof. Let $\mathcal{S}^* = \{a_1^*, a_2^*, \ldots, a_n^*\}$ be an n-tuple of strategies such that $c(p(\mathcal{S}^*)) < c(p(\mathcal{S}))$. Select two consecutive players u and v $(u < v)$ in $p(\mathcal{S}) \cap p(\mathcal{S}^*)$ such that $a_u \neq a_u^*$ and the length of the corresponding sub-path between u and v in $\mathcal{S}^*$ is strictly shorter than that of $\mathcal{S}$ (see Figure 2.3). Note that such two players exist since the two paths intersect at root and sink nodes and one of the paths is strictly shorter. Construct $\bar{\mathcal{S}} = (\bar{a}_1, \ldots, \bar{a}_n)$, where

$$\bar{a}_j = \begin{cases} a_j & \text{if } j \in p(\mathcal{S}), \\ a_j^* & \text{otherwise.} \end{cases}$$

By construction, when best replying to $\bar{\mathcal{S}}$, player u will deviate from his current action a_u to action a_u^* (or an even better one) to obtain a strictly shorter path. □

Theorem II.4. *With* $L = 1$ *in each SFP run, RSFP will converge to an optimal solution in finite time with probability one on finite horizon DP problems.*

Proof. By lemmas II.2 and II.3, there is a strictly positive probability that in each

of the SFP runs of RSFP *on-the-path* players will deviate to obtain a shorter path length unless the optimal solution is already found. In addition, the path lengths returned in each SFP run are monotone non-increasing, *i.e.*, either shorter than or the same as the path length from the previous iteration. Since there are finitely many paths in finite horizon problems, RSFP will converge to an optimal solution in finite time with probability one. □

To show that RSFP converges to the optimal solution in finite time we used the monotonicity of the path lengths returned by SFP when the history size $L = 1$. If the history size $L > 1$, the monotonicity property no longer applies. However, the numerical experiments suggest that even when $L > 1$, the RSFP converges to an optimal solution.

2.4.2 An SFP Based Local Search (SFPLS)

Like any other search algorithm, RSFP has to explore the entire feasible set in order to confirm that it has found an optimal solution. The randomization between SFP runs and sampling from the history in each SFP run provide the necessary exploration. On the other hand, RSFP also has to use exploitation to reach its goal rapidly, which is achieved through best reply calculations. Each SFP run uses exploitation until it finds a local optimal solution (a NE strategy), and it is beneficial to spend time and effort in this fashion to find the local optimal solutions if the problem in consideration only has a few of them. However, for problems with a very large number of local optimal solutions, it might be undesirable to spend resources on identifying the local optimal solutions that may not be close to the optimal solution. Therefore, we also develop a slightly different version of RSFP called the SFP Based Local Search Algorithm (SFPLS), which instead of running a full SFP in each run,

randomizes actions of *off-the-path* players immediately after best reply calculations are made. In other words, we combine best-reply calculations and randomization of actions in each iteration of SFP instead of running a full SFP and identifying local optimal solutions.

SFP Based Local Search Algorithm

Initialization:

- Randomly sample an action $x_i \in A_i$ for each player i.
- Set $\mathcal{S}^0 = (x_1^0, x_2^0, \ldots, x_n^0)$
- $k \leftarrow 1$

Iteration t:

Step 1 Best Reply: Calculate best reply of *on-the-path* players:

$$y_i^t = \arg\min_{x \in A_i} c((x_1^t, x_2^t, \ldots, x_{i-1}^t, x, x_{i+1}^t, \ldots, x_n^t)), \quad \forall i \in p(\mathcal{S}^{t-1}).$$

Step 2 Randomize: Sample a random action $y_i^t \in A_i$ for all *off-the-path* players.

Step 3 Set $\mathcal{S}^t = (y_i^t, \ldots, y_n^t)$.

Step 4 $t \leftarrow t + 1$, go to *Step 1*.

The convergence proof of SFPLS is identical to that of RSFP algorithm and therefore omitted.

As explained above, in each iteration of SFPLS, the *off-the-path* players randomly sample an action from their set of feasible actions in search for a shorter path than the current one. Since the samples are independent of the ones in previous iterations, the worst case bound on the number of iterations required to find a shorter path would be close to that of a pure random search algorithm. For example, consider the

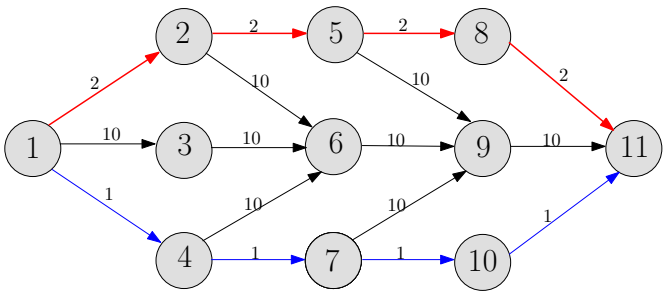


Figure 2.4: An example to illustrate the number of iterations required for SFPLS to find a shorter path than the current one.

network given in Figure 2.4. Suppose that the red arcs represent the path formed by the initial set of actions. The length of this path is 8 units. The blue arcs on the network represent the only path that is shorter than the initial (red) path. Note that, SFPLS can successfully identify this path only if the players 4, 7 and 10 sample the blue arcs. Since the samples are chosen from the entire set of feasible strategies, the number of iterations required to identify the blue path would be close to a pure random search. In addition, note that in RSFP *off-the-path* players sample random strategies once the algorithm is trapped at a local optimal solution (*i.e.*, NE). Therefore, the example in Figure 2.4 also illustrates the number of SFP runs RSFP can take to find a NE with shorter path length. Due to these observations we are not going to pursue worst-case convergence rate analysis, but rather compare the algorithms based on their empirical performance.

2.5 Numerical Results

We use the classical dynamic lot sizing problem (DLSP) to test the performance of the algorithms developed in previous sections. DLSP is the problem of finding a schedule of order quantities (or lot sizes) to satisfy a given demand in each period while minimizing the sum of fixed and variable production and inventory holding costs for a planning horizon of N periods [21]. Algorithms to solve this problem

has been studied under several different assumptions. Federgruen *et al.* developed an algorithm to solve the general model with n periods in $O(n \log n)$ time and also described an $O(n)$ time algorithm for models with nondecreasing setup costs and also for models where in each interval of time per unit order cost increases by less than the cost of carrying a unit in stock [21]. Ahuja *et al.* also studied capacitated dynamic lot sizing problems without setup costs. They used successive shortest-path algorithms for minimum-cost flow problems to solve dynamic lot sizing problem in $O(n \log n)$ time [2]. In these papers, the authors used the structure of the problem to quickly find the optimal solutions; however, our aim is to model the problem without exploiting the structural properties and apply our algorithms on the network obtained from this model. We formulate DLSP within a dynamic programming (DP) framework by defining the states to be the tuple (s, I_s), where s and I_s denote the period and the inventory level at the beginning of that period, respectively. Optimal value function $f(s, I_s)$ of the DP recursion can be written as:

$$f(s, I_s) = \min_{(d_s - I_s)^+ \le x_s \le \sum_{j=s}^{N} d_j} \{c_s \mathbf{1}_{\{x_s > 0\}} + h_s(I_s + x_s - d_s) + p_s x_s + f(s+1, I_s + x_s - d_s)\}$$

$$\text{for } s = 1, \ldots, N-1 \text{ and } I_s = 0, \ldots, \sum_{j=s}^{N} d_j$$

with terminal condition:

$$f(N, I_N) = c_N + p_N(d_N - I_N) \quad \text{for } I_N = 0, \ldots, \delta_N,$$

where

- d_s = demand in period s
- c_s = fixed cost of production in period s
- p_s = variable cost per unit of production in period s

- h_s = cost of carrying a unit of inventory during period s
- x_s = decision variable that denotes amount of production in period s.

Random data for the test problems are generated using the first order autoregressive equations proposed in [21] to generate random data. Specifically, the following equations are used:

$$d_1 = e_1^d,\ c_1 = e_1^c,\ p_1 = e_1^p,\ h_1 = e_1^h$$

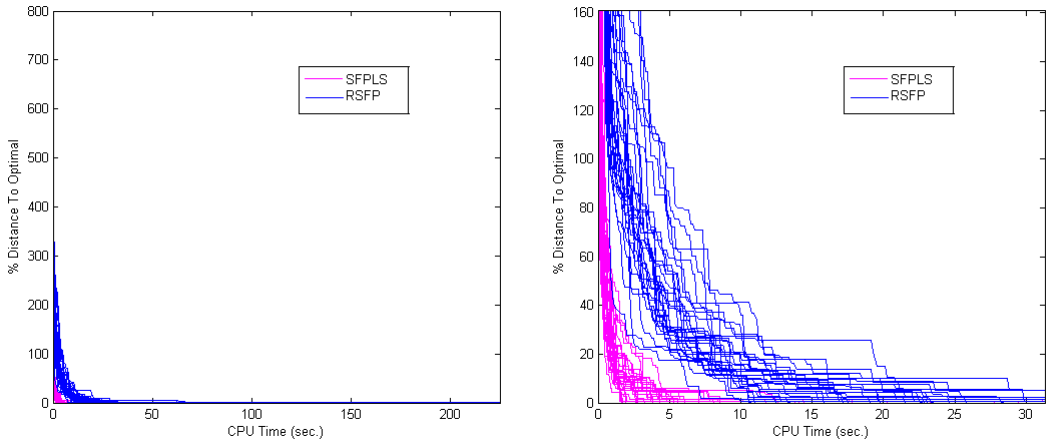
and for $s > 1$:

$$\begin{aligned} d_s &= \alpha d_{s-1} + (1-\alpha)e_s^d, \\ c_s &= \alpha c_{s-1} + (1-\alpha)e_s^c, \\ p_s &= \alpha p_{s-1} + (1-\alpha)e_s^p, \\ h_s &= \alpha h_{s-1} + (1-\alpha)e_s^h, \end{aligned}$$

where the random variables $e_s^{(.)}$, $s = 1 \ldots N$, are independent and uniformly distributed on the interval $[1, 5]$ and $\alpha \in [0, 1]$ represents the correlation between periods. Note that $\alpha = 0$ corresponds to i.i.d. data in each period and $\alpha = 1$ results in constant values over time.

Random data generated for a 20 period DLSP resulted in a DP network with approximately 400 nodes. We ran RSFP and SFPLS on this problem 50 times with random initial strategies. The results are shown in Figure 2.5. Figures 2.5(a) and 2.5(b) are obtained with the same data, where Figure 2.5(b) is the zoomed in version of Figure 2.5(a). The magenta and blue lines in Figure 2.5 show the change in percent distance between incumbent solution and the optimal solution for the SFPLS and RSFP, respectively. The percent distance is calculated by

$$\frac{\text{incumbent value - optimal value}}{\text{optimal value}}.$$



(a) (b)

Figure 2.5: Performance of RSFP and SFPLS on a 20 period DLSP. The algorithms are run on the same problem 50 times with random initial strategies.

As can be seen in Figure 2.5, while both algorithms take several minutes to find the optimal solution, they can identify good (but not necessarily optimal) solutions very quickly.

During the experiments, CPU times were recorded when a solution with objective value within 1% of the optimal value was identified and the results are presented in Figure 2.6. As can be seen in Figure 2.6, 90% of SFPLS runs identified a solution within 1% of the optimal in less than 10 seconds. On the other hand, only 30% of RSFP runs found a solution within 1% of the optimal in less than 10 seconds. Note that each run of RSFP and SFPLS is started with a randomly generated solution. Therefore, these results also show the benefit of using our algorithms compared to randomly searching the feasible region.

The numerical results obtained using DLSP suggest that SFPLS performs better than RSFP on the test problems. One possible explanation is that RSFP aims to identify the best Nash Equilibrium strategy by identifying a better NE than the current one in each iteration. However, the network in consideration has large

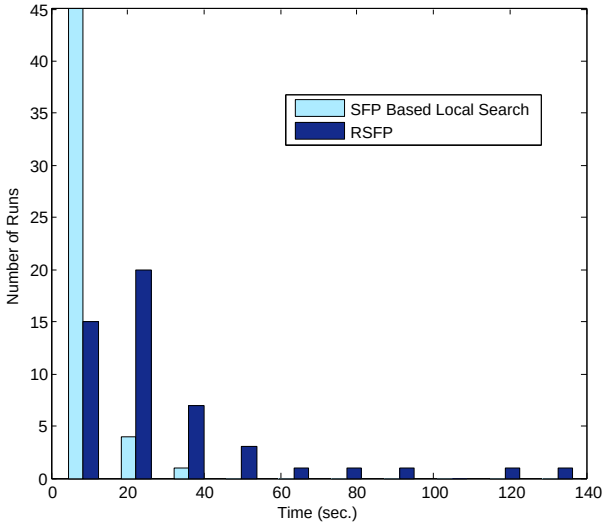


Figure 2.6: Histogram of solutions with objective values within 1% of the optimum returned by RSFP and SFPLS on DLSP. Both algorithms are run 50 times with random initial strategies.

number of NEs. (The histogram in Figure 2.2 is obtained using the same problem and it gives an idea of the number of NEs on the network.) On the other hand, SFPLS is searching for better solutions, not necessarily for equilibrium strategies, and achieves its goal faster. Therefore, on a network with large number of local optimal solutions, a more greedy approach such as SFPLS performs better.

To show that SFPLS does not always outperform RSFP, we generated DLSP network for a 10 period problem with random data. However, instead of using the generated cost data, we assigned a random number between 1 and 100 to each arc of the network. With this approach we were able to obtain a densely connected DP network with random arc costs which eliminated any structural properties the algorithms may have benefit from. The results of running both algorithms on this DP network are given in Figure 2.7. In this case, SFP based local search and RSFP gave comparable results.

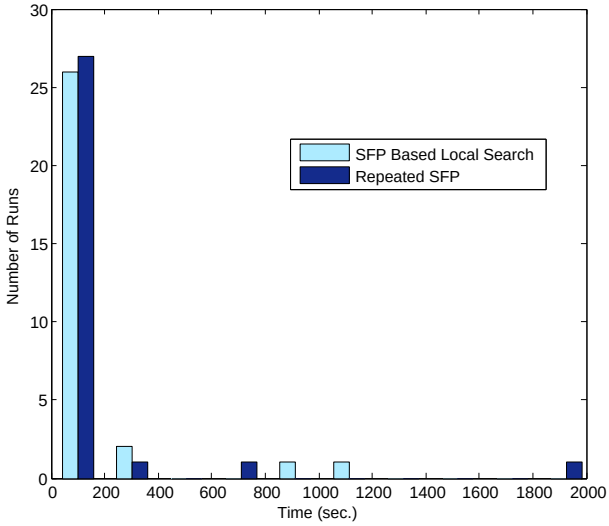


Figure 2.7: Histogram of solutions with objective values within 1% of the optimum returned by RSFP and SFPLS on DLSP network with random arc costs. Both algorithms are run 30 times with random initial strategies.

In conclusion, the performance of both algorithms is highly dependent on the network structure of the DP at hand. Therefore, depending on the type of the problem, one algorithm may outperform the other.

2.6 Conclusions

RSFP and SFPLS are designed for deterministic large-scale DP problems, where a good solution is needed fairly quickly. Both algorithms are stochastic algorithms that learn from the past decisions and aim to improve solutions in each iteration. The classical DP algorithms (such as Dijkstra's Algorithm), on the other hand, are designed to find the optimal solutions by searching the feasible set with backward or forward recursion. As mentioned before, due to curse of dimensionality, these classical DP algorithms lose their effectiveness as the dimensionality of the problem increases. For large-scale problems, we propose using SFP based algorithms to find good (if not optimal) solutions quickly.

The numerical results presented in this chapter were obtained using a small test problem for which we were able to calculate the optimal solution. Note that our algorithms do not explicitly exploit any structural property of the test problems. Therefore, they can be easily applied to sequential decision making problems with more complex objective functions requiring intensive computations or to black-box optimization problems which do not have a closed form objective function.

The algorithms we designed are parallelizable as the best replies can be computed on parallel computers. Unfortunately, in our tests we were unable to use parallel computing techniques and thus could not present the full potential of the algorithms. However, clearly the performance of the algorithms will be much better when run on parallel computers.

In each iteration of the SFP based algorithms presented in this chapter, best reply of an *on-the-path* player is calculated by comparing the length of the path sampled at the beginning of the iteration with the possible path lengths that can be obtained if the *on-the-path* player deviates. To calculate a best reply, information on only a limited portion of the network is needed. In other words, with our algorithms it is possible to obtain good solutions by searching only a portion of the DP network. In addition, this provides us the flexibility to design these algorithms in an online fashion to simultaneously learn and optimize stochastic DP problems, which is a future research direction we will pursue.

CHAPTER III

Application of RSFP and SFPLS to Traveling Salesman Problem

3.1 Introduction

In Chapter II, we described the Sampled Fictitious Play (SFP) algorithm and how it can be used to find a globally optimal solution of a Dynamic Programming (DP) problem. We specified two algorithms based on SFP: Repeated SFP (RSFP) and SFP Based Local Search (SFPLS). We showed that both algorithms converge to the optimal solution on deterministic and finite horizon DP problems. We also tested the performance of both algorithms on small test problems.

In this chapter, we will present the empirical results on the performance of these algorithms applied to the DP formulation of the Traveling Salesman Problem (TSP) to show their potential for solving larger-scale problems. We will also discuss possible modifications that can improve the performance of the algorithms.

3.2 Traveling Salesman Problem

Given a set of cities and the distances between them, TSP aims at finding the shortest tour that visits each city exactly once [14]. The problem was first formulated in the 1930s [3] and it has been extensively studied since then. The popularity of the problem stems from the fact that it is an NP-complete problem [37] and has var-

ious practical application areas such as machine scheduling, cellular manufacturing and frequency assignment problems [28]. Possible approaches to solving the problem include using branch-and-bound and branch-and-cut methods based on integer programming formulations or designing heuristics such as 2-opt search, k-opt search and Lin-Kernighan heuristics that exploit the structure of TSP. There is also a vast literature on the applications of well known heuristics to TSP such as simulated annealing, genetic algorithms and tabu search [41]. In addition, approximation algorithms for finding upper and lower bounds on the optimal solution of TSP have also been extensively studied. While the list provided here represents a small subset of the literature on TSP, it is representative of the importance of TSP's.

Since RSFP and SFPLS are designed specifically for DP problems, we only consider the DP formulation of TSP in this chapter . Held et al. [29] and Bellman [10] provided the DP formulation of TSP along with the computational requirements of the method and some numerical results. As Held et al. has argued, DP is not an efficient method to solve TSP; however, it offers several advantages when used in an approximation framework [29]. Therefore, in this chapter we propose using RSFP and SFPL as the means to obtain close to optimal solutions for TSP quickly and we aim to show the potential of these algorithms on a large-scale DP network that represents TSP.

To formulate TSP as a DP, we consider a directed graph denoted by $G = (V, A)$ where V and A represent the sets of cities and the links connecting cities, respectively. Let c_{ij} denote the length of link (i, j) and assume $c_{ij} = c_{ji}$ *i.e.*, the problem is symmetric. We can consider TSP as a time staged decision making problem, where given the current city k we are in and the list of cities that haven't been visited so far, we need to decide the next city to visit. Assuming that we start in city 1, define

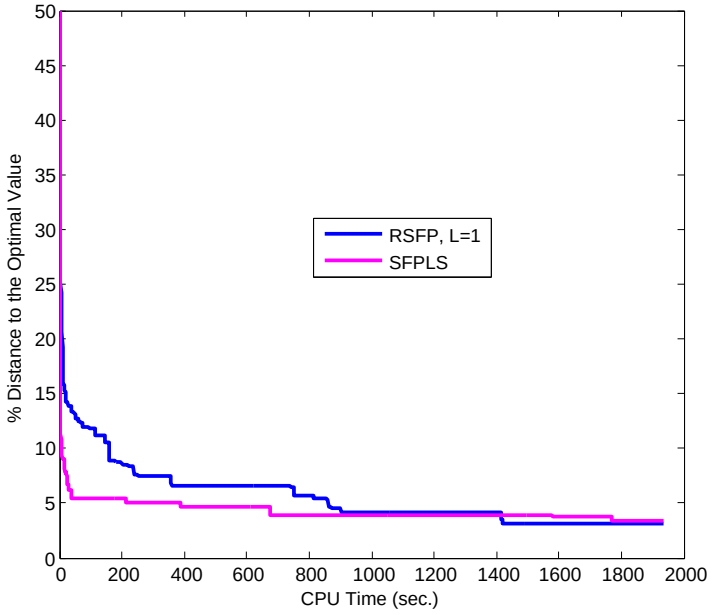


Figure 3.1: Average performance of RSFP and SFPLS on a 15-city TSP. The algorithms are run 15 times with random initial strategies.

the states to be (S, k) where S is the set of cities that have not been visited and k is the current city. The DP recursion is:

$$f(S, k) = \min_{m \in S}(c_{km} + f(S \setminus m, m))$$

with the boundary condition:

$$f(\emptyset, 1) = 0.$$

The number of nodes on the DP network defined by the above recursion for a TSP with n cities is in $O(n2^n)$ [14].

3.3 Application of RSFP and SFPLS to TSP

We used a problem instance with 15 cities and ran RSFP and SFPLS 15 times with different starting points. We recorded the value of the incumbent solution as a function of CPU time (in seconds) during each run. The DP network generated for

this problem has 114,700 nodes including the root and the sink nodes. The problem data are generated by using the coordinates given in [1] as city locations and the Euclidean distance between the cities as the link lengths.

Figure 3.1 shows how the average percent distance of the incumbent solution to optimum changes as time progresses based on the 15 runs. As it can be seen from Figure 3.1, the values obtained with both RSFP and SFPLS decrease rapidly; however, they both stall at a local optimal solution for a long time before they find a new improving solution as they get closer to the optimum. It is important to note that these algorithms do not take advantage of the symmetry or the Euclidean distances, which algorithms in the literature often rely on. In addition, the performance of the SFP based algorithms can be increased significantly with parallel computing techniques. In the remainder of this section, we will present a simple modification that further improves the performance of RSFP and also discuss the effect of history size on the performance of RSFP and modified RSFP.

One straightforward way to improve the performance of RSFP is to help the algorithm search the feasible region more efficiently as the incumbent solution gets closer to the optimum. In the original algorithm, when a player is *off-the-path*, the action it plays is not considered in the payoff function (recall that the identical payoff is defined to be the path length). Therefore, given the actions of all players, which also define the path, the best reply of *off-the-path* players can be any feasible strategy. However, in Chapter II the best reply action of *off-the-path* players was specified to be the last entry in their history of best replies. The motivation behind this was to have *off-the-path* players' histories fixed as soon as possible in order to speed up the convergence of an SFP run to a NE. This approach, however, limits the capability of the algorithm to explore and is one of the reasons that the algorithm can not

easily find an improving solution once a "close" to optimum solution is reached. To address this issue, we have slightly modified RSFP by leaving the histories of *off-the-path* players unchanged. The details of an SFP run in the modified RSFP are given below.

Modified SFP Algorithm

Initialization

- Set $k \leftarrow 0$
- Randomly sample an action $x_i^0 \in A_i$ for each player i.
- Set $y_i^0 = x_i^0 \ \forall i \in \mathcal{N}$ and add it to the history, *i.e.*, $\mathcal{H}_i^1 = [y_i^0]$ (*i.e.*, the histories of all players are initialized with a random action).
- Set $k \leftarrow k + 1$.

Iteration $k \geq 1$

Step 1 Sample:

Step 1a. For each player i, sample an action $x_i^k \in A_i$ from the history $\mathcal{H}_i^k$ uniformly at random.

Step 1b. Set $\mathcal{S}^k = (x_1^k, x_2^k, \ldots, x_n^k)$.

Step 1c. Identify the unique path, $p(\mathcal{S}^k) = (i_1^k, i_2^k, \ldots, i_m^k)$, defined by $\mathcal{S}^k$.

Step 2 Play:

For each player i,

- If $i \in p(\mathcal{S}^k)$, player i plays the best reply strategy calculated by

$$y_i^k = \arg\min_{x \in A_i} c(x_1^k, x_2^k, \ldots, x_{i-1}^k, x, x_{i+1}^k, \ldots, x_n^k).$$

If there are multiple actions that minimize the function above, choose one of them according to a lexicographic tie breaking rule (see the original algorithm description in Section 2.3.1 for details).

Step 3 Update History:

- Add the played action to the history: $\mathcal{H}_i^{k+1} = [\mathcal{H}_i^k; y_i^k]$.[1]

Step 4 Termination Criteria:

- If $1 \leq k \leq L$ and $y_i^0 = y_i^1 = y_i^2 = \ldots = y_i^k$ for all $i \in \mathcal{N}$, then terminate.
- If $k \geq L+1$ and $y_i^{k-L} = y_i^{k-L+1} = y_i^{k-L+2} = \ldots = y_i^k$ for all $i \in \mathcal{N}$, then terminate.
- Otherwise
  - If the size of $\mathcal{H}_i^{k+1} > L$, then delete the earliest entry of $\mathcal{H}_i^{k+1}$.
  - Set $k \leftarrow k+1$
  - Go to *Step 1*.

We compared the performance of RSFP and modified RSFP on a problem instance with 12 cities by running both algorithms 30 times with history size $L = 1$. In this experiment we used the data of the first 12 cities of the previous 15-city example is used. We recorded the value of the incumbent solution as a function of CPU time (in seconds). As can be seen from Figure 3.2, even though SFPLS still outperforms the modified RSFP, the modification has significantly improved the performance of RSFP. The empirical results suggest that modified-RSFP converges to the optimal solution. However, the convergence proof of modified RSFP is omitted here and will be a future research topic.

[1] $[\mathcal{H}_i^k; \cdot]$ denotes the concatenation function.

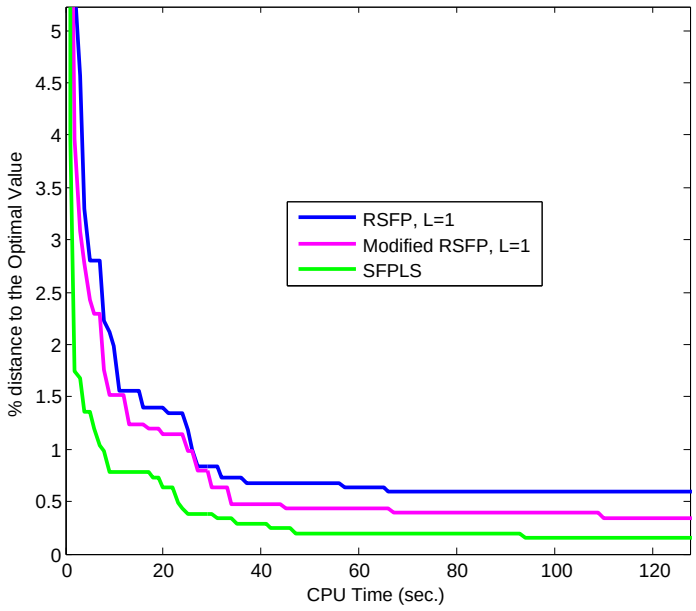


Figure 3.2: Average performance of RSFP, modified RSFP with $L = 1$ and SFPLS on a 12-city TSP. The algorithms are run 30 times with random initial strategies.

In Chapter II, we presented the convergence results for RSFP with history size of 1 (*i.e.*, $L = 1$) and had a conjecture based on empirical results that the algorithm converges even with $L > 1$. Figure 3.3 shows the performance of RSFP on a 12-city TSP with $L = 1$ and $L = 5$. The results suggest that while the algorithm still converges with $L = 5$, the rate of convergence is much slower.

The effect of changing the history sizes on the performance of modified RSFP is presented in Figure 3.4. Similar to previous results history size of 1 performs better compared to history size of 5.

All the experiments for 12-city TSP were run until the optimal solution was found. During each run, we recorded the time at which the algorithms found a solution within 1% of the optimum. Figure 3.5 presents a histogram of this time data for 30 runs. As can be seen from the figure, the algorithms with $L = 5$ get stuck at a local optimal solution more often.

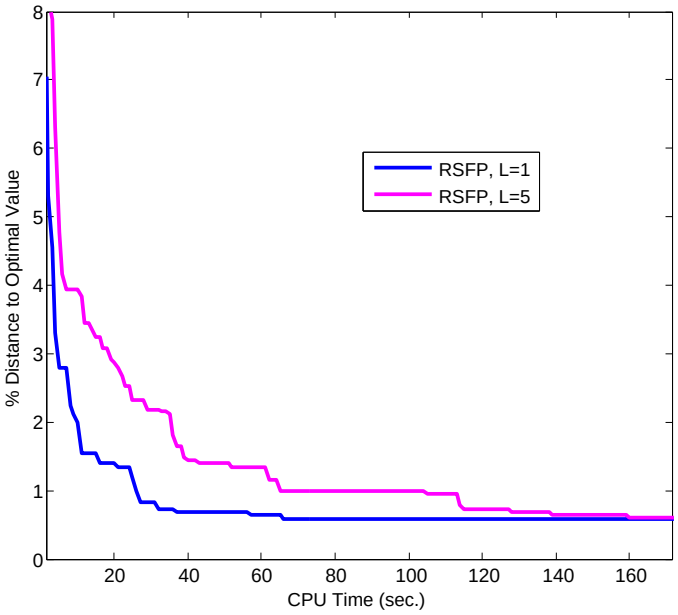


Figure 3.3: Average performance of RSFP with $L = 1$ and $L = 5$ on a 12-city TSP. The algorithms are run 30 times with random initial strategies.

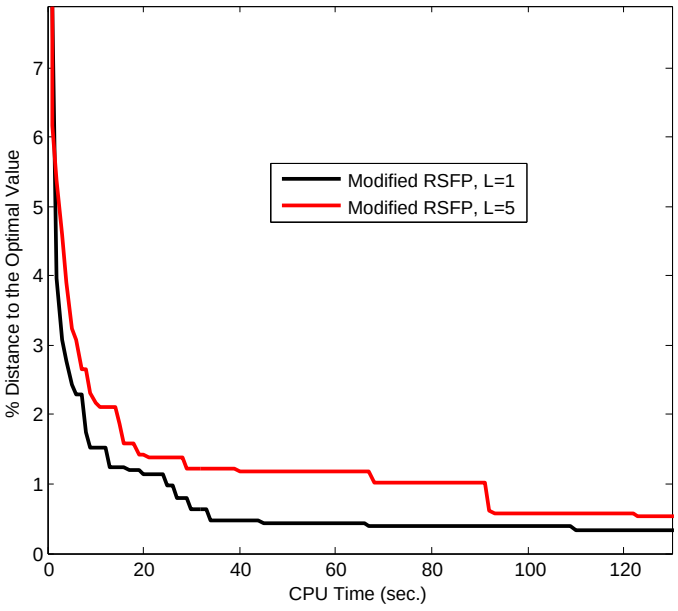


Figure 3.4: Average performance of modified RSFP with $L = 1$ and $L = 5$ on a 12-city TSP. The algorithms are run 30 times with random initial strategies.

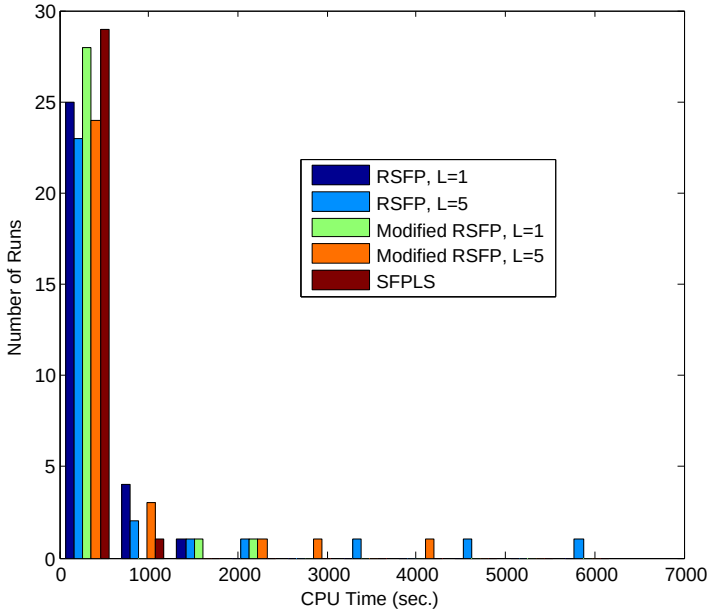


Figure 3.5: Histogram of solutions that are within 1% neighborhood of optimum returned by RSFP, modified RSFP an SFPLS on DP network of 12-city TSP. All algorithms are run 30 times with random initial strategies.

3.4 Conclusions

In this chapter, we studied the potential of the algorithms described in Chapter II for solving large-scale DP problems and investigated possible ways to improve their performance. It should be noted that the performance of these algorithms can be significantly improved by utilizing parallel computing. As was discussed previously, SFP can be easily implemented using parallel computing and therefore the running times of RSFP and modified RSFP can be significantly reduced.

CHAPTER IV

Sampled Fictitious Play Based Online Learning Algorithms for Markov Decision Processes

4.1 Introduction

In this chapter, we propose an online learning algorithm [15] based on Fictitious Play concepts for homogeneous Markov Decision Problems (MDPs) where the transition probabilities are unknown. MDPs are characterized as sequential decision making problems that involve choosing actions, observing the information provided within the system, and then modifying these actions based on the new information [39]. The nature of these problems makes it possible to propose solution approaches even when the dynamics of the system (*e.g.*, cost of a decision, the probability of random disturbances in the system, etc.) are unknown. More specifically, by designing a learning algorithm that simultaneously observes the "unknowns" in the system and optimizes the decisions using information obtained via these observations, the optimal solution of the problem can be approximated. Ideally such an algorithm uses simple calculations to generate approximate solutions in each iteration. Therefore, Fictitious Play (FP) concepts can be very useful in this framework. FP is an iterative learning algorithm designed to imitate behavior of players engaged in a repeated game and for a subclass of games, to find a Nash Equilibrium[1] solution of a game

[1] A Nash Equilibrium is a collection of probability distributions over each player's strategies with the property that no player can unilaterally improve its utility in expectation by changing its own distribution [22].

[16]. Therefore, our approach is to model an MDP problem as a game (by defining players, strategies, and the payoff function) such that the optimal solution of the problem is a Nash Equilibrium of this game. This way, FP can be effectively used to find the optimal solution of the MDP.

Learning algorithms for MDPs have been extensively studied in the literature. Q-Learning, SARSA, actor-critic methods [5], [13], Approximate Dynamic Programming [39] are well known examples of such algorithms. Q-Learning and SARSA (a variant of Q-Learning) have been very popular in recent years due to the fact that they do not require the knowledge of the system model [5]. Therefore these algorithms can be applied to a large variety of problems without making any structural assumptions about the problem at hand. Both of these algorithms estimate the expected value of each state-action pair and update these estimates as more information becomes available. Q-Learning is an *off-policy* learning algorithm, *i.e.*, the update of the value of a state-action pair is independent of the policy that is being followed, whereas SARSA is an *on-policy* learning algorithm, in which the policy that is being followed is directly used in the value updates [5]. Actor-critic method is also a learning algorithm, in which the beliefs on the optimal actions in each state are recorded in addition to the value estimates.

In contrast to the above algorithms, in this chapter we design an SFP based learning algorithm that not only keeps an estimate of the optimal value and the optimal action for each state but also estimates the system parameters (the transition probability distribution). *We assume that the randomness in the system depends on the state but is independent of the action taken.* This assumption holds for many practical problems. For example in inventory control problems [40], the demand is usually independent of the production amount. In dynamic location problems [43], in

which an equipment trailer has to be placed in one of the predefined locations so that the expected cost of completing a work order from a random location is minimized, the probability that a work order comes from a particular location is independent of where the equipment trailer is placed. Note that, if this assumption is not satisfied (*i.e.*, the random disturbance depends on not only the state but also the action), the algorithmic approach we propose in this chapter can still be used with slight modifications and the convergence analysis can easily be adapted for the modified approach.

The target classes of problems for our algorithm are the ones with finite state space and discrete probability transition functions with finite sample space. However, the results of our work can be generalized for problems in which the state transitions are governed by a parameterized continuous distribution by estimating the parameters of this distribution. We empirically show that using the estimate of the system dynamics can considerably speed up the convergence of learning algorithms by comparing the performance of our algorithm to those of SARSA and Q-Learning. We chose SARSA for comparison since our algorithm, like SARSA, can be classified as an on-policy method. We also compare our algorithm with Q-Learning since both algorithms are "learning" via observing the system in an online fashion.

The algorithms that estimate the model parameters such as ours are also classified as *Indirect Algorithms* [27]. Various indirect learning algorithms are studied in [4], [31], [32] and [27]. In [4], Barto et al. discuss connections between control theory, online learning algorithms and indirect algorithms. They present simulation results showing the difference between online and offline algorithms as well as direct and indirect algorithms by comparing several algorithms such as Real Time Dynamic Programming (RTDP), Adaptive-RTDP, Q-Learning and conventional Dy-

namic Programming. In [31] and [32], Kearns et al. study the convergence rate of Q-Learning and analyze an alternative algorithm that simulates the system until enough information is collected on a set of states and then uses optimization routines to update the value of these "known" states. The difference in our algorithm from that of Kearns et al. is that we perform the simulation and the optimization operations simultaneously. Our numerical experiments, presented in Section 4.5, show promising results.

In learning algorithms, such as the ones described above, the performance strongly depends on the sampling rules employed to explore the action space. The most straightforward approach is to randomly sample actions and update values based on the information obtained from these sampled actions. However, biasing the sampling procedure to choose "better" actions can significantly improve the performance of these algorithms [18]. To achieve this, one can go to the other extreme and use a greedy approach that always samples the best action based on the current beliefs. However, such an approach would introduce too much bias and possibly force the algorithm to get "stuck" at a local optimal solution. To deal with this, one of the most common approaches is to use ϵ-greedy sampling. In this sampling method, the best action based on the current information is sampled with high probability while with ϵ probability a random action from the feasible set is sampled to facilitate more exploration. As an alternative to these sampling approaches, Chang et al. [17] designed an adaptive sampling algorithm for finite horizon stochastic dynamic programming problems in which they incorporate the regret analysis developed for the multiarmed bandit problems into the action sampling process. With this adaptive sampling approach, they save the effort of sampling "poor actions" (and this would significantly improve the algorithm performance if the sampling is relatively time

consuming) without sacrificing the convergence properties of the algorithm. The notion of adaptive sampling (and therefore its possible advantages) is naturally inherent in our algorithm through the utilization of "best action" histories, which are part of the FP framework. The details on how exactly we use these histories will be explained further after the details of the algorithm are given.

As mentioned earlier, we use FP concepts in designing our algorithm. FP is an iterative algorithm used to imitate the behavior of players engaged in a repeated game [16], [42] and it has been shown to converge to a Nash Equilibrium for identical interest games [35]. At every iteration of FP, each player (or decision maker) selects the best pure action (*i.e.*, a best reply) given all other players play an action according to the empirical distribution of their own best replies from the previous iterations [16]. Finding a best reply to the actions of all other players, which follow the probability distribution defined by the history of past best replies, requires making best reply calculations for every possible combination of actions in the histories of players. This approach is computationally time consuming since the history size grows in each iteration. Lambert et al. proposed a Sampled Fictitious Play (SFP) algorithm, in which best replies to a sample of actions drawn from the history of best replies are calculated, and showed that it converges to a Nash Equilibrium [33]. Computational requirements of the SFP algorithm are dramatically less compared to that of FP, since the best reply calculation of each player is made only for a given sample of actions.

To facilitate application of these concepts in our algorithm, similar to our approach from Chapter II, we define a game on a homogeneous infinite horizon MDP by treating the states and feasible decisions in each state as the players and actions available to each player, respectively. In this framework, a collection of actions

played by all players defines a policy for the MDP problem. The identical payoff each player receives is then naturally defined to be the expected total discounted reward (or cost) of this policy. The value of a policy can be found either using the knowledge of system dynamics, or by simulating the policy and observing the system. Since here we consider problems for which the probability distribution of the random disturbance in the system is not readily available, we are bound to simulate and estimate the unknowns in the system. A direct application of SFP in this setting would be to have each player randomly sample an action from the history of best replies and then calculate a best reply for each player given the sampled actions of all other players. However, a best reply calculation will require the simulation of the system several times to be able to compare the payoff of every feasible action of all players. In this approach, these simulations can only be run offline since it is computationally infeasible to perform them in an online fashion. In contrast, in the proposed algorithm we simultaneously estimate the probability distribution, the optimal value of each state, and the optimal action of each state in an online fashion during a single simulation of the system by using the value estimates of the states to calculate the best reply. Specifically, in each iteration we record an estimate of the probability that a given random event occurs in each state, an estimate of the optimal value, and history of best reply actions of each state. Note that the empirical probability distribution on the best reply actions from the history can be viewed as the probabilistic estimate of the optimal action in that state. Given the most up-to-date values of each state, we randomly draw an action from the history of best replies for the current active state, and then observe the random disturbance and the state to which the system transitions. Using these observations, we update the estimate of the random disturbance probability and the estimate of the optimal value. We

calculate the new best reply and add it to the history. Note that adding the best reply action to the history can be viewed as updating the estimate of the optimal action (*i.e.*, history of best replies) of the current state as it increases the probability that this action is sampled next time the system enters the same state. Therefore, as the algorithms proceeds, the values of states will get closer to the optimal values, and thus the optimal actions will be identified via best reply calculations. As a result, the probability of sampling the "right" action will increase in an adaptive fashion. We show that the estimates of the values converge to the optimal values with this approach. We present the empirical results on the dynamic location and windy gridworld problems.

This chapter is organized as follows. In Section 4.2, notation and assumptions are given. In Section 4.3, the algorithm is described. Section 4.4 presents the convergence proofs. The algorithm's performance is tested on the dynamic location and windy gridworld problems in Section 4.5. The chapter is concluded in Section 4.6.

4.2 Notation and Assumptions

In this chapter we consider an infinite horizon, discrete time, homogeneous Markov Decision Process (MDP) defined by the collection of objects

$$(S; A_s, p(\cdot|s,a), c(s,a),\ \forall s \in S)$$

where

- S is the set of states ($|S| < \infty$, where $|S|$ is the cardinality of set S).
- A_s is the set of feasible actions in state $s \in S$ ($|A_s| < \infty$).
- $p(s'|s,a) = \text{Prob}(s_{t+1} = s'|s_t = s, a_t = a)$ is the probability the system transitions into state s' at time $t+1$ given the system is in state s and the action

$a \in A_s$ is chosen at time t.

- $c(s, a) \in \mathbb{R}$ is the immediate cost of choosing action a in state s.

Once an action a is chosen in state s, an exogenous random disturbance is realized and the system transitions into state s'. We assume that the random disturbance depends on the current state s but is independent of the action a chosen. As mentioned earlier, inventory control problem [40] and dynamic location problems [43] are examples of problems that satisfy this assumption. We use the following notation to represent the random disturbance and the transitions in the system:

- $\omega(s) \in \Omega(s)$ denotes the random disturbance, which is assumed to be independent of action taken in state s. Sample space $|\Omega(s)|$ is assumed to be finite for every state s: $|\Omega(s)| < \infty, \ \forall s \in S$.

- $p(\omega(s) = \hat{\omega}(s)|s)$ is the probability that the random disturbance $\hat{\omega}(s)$ occurs when the system is in state s. In the reminder of the chapter we will use the shorthand notation $p(\hat{\omega}(s))$ to represent this probability.

- $f(s, a, \hat{\omega}(s))$ is the system evolution equation function. That is, given a realization $\hat{\omega}(s)$ of $\omega(s)$ and action $a \in A_s$, the system will transition from state s to the (deterministic) state $f(s, a, \hat{\omega}(s))$.

Using the notation defined above, the transition probabilities can be written as follows:

$$p(s'|s, a) = \sum_{\omega(s) \in \Omega(s)} p(\omega(s)) \mathbf{1}_{\{f(s,a,\omega(s))=s'\}}, \tag{4.1}$$

where $\mathbf{1}_{\{.\}}$ is the indicator function.

In this chapter, we specifically focus on the problem of minimizing discounted expected cost with $\gamma \in (0, 1)$ denoting the discount factor. We assume that feasible states, actions in each state of the problem and the costs of these actions are

known, but no or little information is available about the probability distribution of the random disturbance within the system. That is, $p(\omega(s)),\ \forall s \in S$, are not known. But a realization of $\omega(s)$ can be observed every time the system visits state s. One approach for solving the problem in this setting is to observe and learn about the system dynamics under an implemented strategy by updating this strategy in an online fashion as more information about the system becomes available. In an online updating, the updates are done simultaneously while observing the system [5]. Therefore, the iteration count of the algorithm coincides with the real time during which the system evolves. In this chapter, we use SFP concepts to develop an online algorithm, therefore, we use the subscript t to refer to both the iteration of the algorithm and the real time.

4.3 SFP Based Learning Algorithm (SFPL)

The following notation is used to describe the SFP Based Learning Algorithm (SFPL):

- MDP denotes the infinite horizon homogeneous Markov Decision Process under consideration.
- $J_t(\cdot) : S \to \mathbb{R}$ is the *estimate* of the optimal cost function for MDP returned by the algorithm at iteration/time t.
- $p_t(\cdot|s,a)$ and $p_t(\hat{\omega}(s))$ denote estimates of, respectively, transition and random disturbance probabilities at iteration/time t.
- MDP_t is the Markov Decision Process defined on the same state and action space as MDP with the difference that the transition probabilities are given by $p_t(\cdot|s,a),\ \forall s \in S,\ a \in A_s$. In other words, MDP_t is the homogeneous MDP

defined by the collection of objects $(S; A_s, p_t(\cdot|s,a), c(s,a),\ \forall s \in S)$.

- $H(s)$ is the history of actions taken in state (or played by player) s.
- $I_t(s)$ is the number of times the process has visited state s up to (and including) iteration/time t.
- $W(s)$ is the vector of realizations of $\omega(s)$ observed up to (and including) the current iteration. By definition, the size of vector $W(s)$ at iteration t is equal to $I_t(s)$ for all s except possibly the current state of the system (for which the associated objects are being updated during the current iteration).

SFPL is initialized at a random state (unless the starting state is already identified) and all histories are set to be empty. In each iteration $t \geq 1$, a random strategy from the history of the current state is selected (if the history is empty a feasible strategy is sampled) and the random disturbance in the system is observed. Based on this observation, the beliefs (or estimates) on the random disturbance probabilities are updated and a best reply is calculated with the updated beliefs. The best reply strategy is then added to the history of the current state and the system transitions into the next state according to the played action and the realized random disturbance. It is worth pointing out again that the action that is played (and thus affects the transition) is randomly sampled from the history in the beginning of the iteration, whereas the action added to the history is the one obtained through the best reply calculation.

SFPL is designed for problems which satisfy the assumption that the probability distribution of the random disturbance is independent of the action. As mentioned before, the algorithm can easily be modified if this assumption is not satisfied. More specifically, if the random disturbance depends on both the state and the action

then the count of random disturbance realizations needs to be recorded for each state-action pair In this case, $W(s,a)$ and $p_t(\omega(s,a))$ would denote the probability estimate that the random disturbance $\omega(s,a)$ occurs in state s under action a. In addition, the value estimates and the count of visits also need to be recorded for each state-action pair (*i.e.*, $J_t(s,a)$ and $I_t(s,a)$). With these modifications, the algorithm will proceed as explained above with the difference that the observations will be recorded and the value updates will be calculated for each state-action pair. It is important to note that these modifications increase the memory requirements of the algorithm since data need to be kept for each state-action pair. The computational requirements of the algorithm, however, would be similar since the expected value of the current state-action pair has to be calculated to update its value.

4.3.1 Outline of Algorithm SFPL

Below, the notation $X \leftarrow [X, x]$ represents the concatenation function.

- Initialization:

  1. Assign initial values to $J_0(s)$ for all $s \in S$.
  2. Set $H(s) = [\,]$, $I_0(s) = 0$, $W(s) = [\,]$ for all $s \in S$.
  3. Set $t = 1$ and observe $s_1 \in S$. (This is the state where the process starts.)

- Iteration t (below, $s_t \in S$ represents the state at the beginning of iteration):

  State :

  1. $I_t(s_t) = \begin{cases} I_{t-1}(s_t) + 1 & \text{if } s = s_t, \\ I_{t-1}(s_t) & \text{if } s \neq s_t. \end{cases}$

  Action :

1. Randomly select an action $a_t \in H(s_t)$ (if $H(s_t)$ is empty, randomly select an action a_t from A_{s_t}).
2. Observe the realization $\hat{\omega}(s_t)$ of the random disturbance $\omega(s_t)$; update $W(s_t) \leftarrow [W(s_t),\ \hat{\omega}(s_t)]$.

Computation :

1. Calculate the "best reply":

$$a^* = \underset{a \in A_{s_t}}{\operatorname{argmin}} \left\{ c(s_t, a) + \gamma \frac{1}{I_t(s_t)} \sum_{i=1}^{I_t(s_t)} J_{t-1}(f(s_t, a, W_i(s_t))) \right\}, \qquad (4.2)$$

   where $W_i(s_t)$ is the ith component of vector $W(s_t)$.

2. Update the history $H(s_t) \leftarrow [H(s_t),\ a^*]$.

3. Set

$$J_t(s) = \begin{cases} c(s, a^*) + \gamma \frac{1}{I_t(s)} \sum_{i=1}^{I_t(s)} J_{t-1}(f(s, a^*, W_i(s))) & \text{if } s = s_t, \\ J_{t-1}(s) & \text{otherwise,} \end{cases}$$

   or, equivalently,

$$J_t(s) = \begin{cases} c(s, a^*) + \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} p_t(\hat{\omega}(s))\ J_{t-1}(f(s, a^*, \hat{\omega}(s))) & \text{if } s = s_t, \\ J_{t-1}(s) & \text{otherwise,} \end{cases} \qquad (4.3)$$

   where

$$\begin{aligned} p_t(\hat{\omega}(s)) &= \frac{\#\text{ of times } \hat{\omega}(s) \text{ is observed in state } s \text{ up to and including iteration } t}{I_t(s)} \\ &= \frac{\sum_{i=1}^{I_t(s)} 1_{\{W_i(s) = \hat{\omega}(s)\}}}{I_t(s)} \end{aligned}$$

   is the estimate of $p(\hat{\omega}(s))$ at iteration t, that is, the probability that the random disturbance $\hat{\omega}(s) \in \Omega(s)$ occurs when the system is in state s.

Transition :

1. Set $s_{t+1} = f(s_t, \bar{a}, \hat{\omega}(s_t))$.
2. Update $t \leftarrow t + 1$.

4.4 Convergence of SFPL

In this section, we first prove that SFPL converges to the optimal solution with probability one under the assumption that the inherent randomness in the system ensures that every state is visited infinitely often regardless of the policy employed. Later, we discuss possible approaches when the assumption does not hold.

Assumption IV.1. *Assume that every state is visited infinitely often, regardless of the policy employed, due to the inherent randomness in the system.*

Although this assumption may seem very restrictive, there are various problem classes that satisfy this assumption. For example, for the windy gridworld problem described in the numerical results section, it is possible to have a wind that blows in a random direction with a random strength so that any state can be visited in the next iteration with a positive probability. In a capacitated inventory control problem, if the demand is allowed to be negative to represent returns and can be at any feasible level with positive probability, this assumption would be valid. However, even for the problems that do not satisfy this assumption, we can always incorporate the well known ϵ-greedy approach in the sampling phase of our algorithm to make sure that all states are visited infinitely often.

The following condition can be used to check if this assumption is valid for a given problem:

Claim IV.2. *If the induced Markov Chain under any stationary deterministic policy of MDP is ergodic, then Assumption IV.1 is satisfied.*

Proof. Follows from the definition of ergodicity. □

Lemma IV.3. $\lim_{t\to\infty} p_t(\hat{\omega}(s)) \to p(\hat{\omega}(s))$ *for all* $\hat{\omega}(s) \in \Omega(s)$ *with probability 1.*

Proof. The lemma follows directly from Assumption IV.1 and Strong Law of Large Numbers [44]. □

In the remainder of the chapter, $J^*(s)$ is used to represent the optimal value of state s in MDP.

Let $F_t : \mathbf{R}^{|S|} \times S \to \mathbf{R}$ denote a function that takes a value vector $J \in \mathbf{R}^{|S|}$ and a state $s \in S$ as inputs and returns a new value for s using the probability distribution $p_t(\omega(s))$ as follows:

$$F_t(J, s) = \min_{a \in A_s} \left\{ c(s, a) + \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} p_t(\hat{\omega}(s)) J(f(s, a, \hat{\omega}(s))) \right\}. \tag{4.4}$$

Note that if J_{t-1} and s_t are input to F_t then the value returned by the function will be the value of state s_t at the end of iteration t of SFPL. Therefore, SFPL generates a sequence of value vectors J_t such that:

$$J_t(s) = \begin{cases} F_t(J_{t-1}, s_t) & \text{if } s = s_t, \\ J_{t-1}(s) & \text{if } s \neq s_t. \end{cases}$$

The progress of SFPL and the values obtained in each iteration depend on the choice of J_0. However, since

$$\bar{c} \equiv \max_{s \in S} \max_{a \in A_s} |c(s, a)| \tag{4.5}$$

is finite and the values are discounted in every iteration, all values obtained in subsequent iterations will be finite as long as $\|J_0\|_\infty < \infty$. This is formally proven below.

Lemma IV.4. *Suppose SFPL is started with an arbitrary finite initial value vector J_0. Let $\bar{c}$ be as in Equation (4.5) and let $\tilde{c} = \max\{\bar{c}, (1-\gamma)\|J_0\|_\infty\}$. Then*

$$|J_t(s)| \leq \frac{\tilde{c}}{1-\gamma} < \infty \ \forall t \in \{1, 2, \ldots\} \text{ and } \forall s \in S.$$

Proof. The lemma holds for $t = 0$ by definition of $\tilde{c}$. Assume it holds for $\|J_{t-1}\|_\infty$ and consider the value update at iteration t:

- For $s = s_t$,

$$\begin{aligned} |J_t(s_t)| &= \left| \min_{a \in A_{s_t}} \left\{ c(s_t, a) + \gamma \sum_{\hat{\omega}(s_t) \in \Omega(s_t)} \bar{p}_t(\hat{\omega}(s_t)) J_{t-1}(f(s_t, a, \hat{\omega}(s_t))) \right\} \right| \\ &\leq \bar{c} + \gamma \max_{s \in S} |J_{t-1}(s)| \\ &\leq \tilde{c} + \gamma \frac{\tilde{c}}{1-\gamma} = \frac{\tilde{c}}{1-\gamma}, \end{aligned}$$

  where the last inequality follows from the inductive hypothesis.

- For $s \neq s_t$,

$$|J_t(s)| = |J_{t-1}(s)| \leq \frac{\tilde{c}}{1-\gamma}, \ \forall s \neq s_t,$$

  which follows from the inductive hypothesis.

□

Let $\bar{F}(J, s) : \mathbf{R}^{|S|} \times S \to \mathbf{R}$ be another update function defined as follows:

$$\bar{F}(J, s) \equiv \min_{a \in A_s} \left\{ c(s, a) + \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} p(\hat{\omega}(s)) J(f(s, a, \hat{\omega}(s))) \right\}, \tag{4.6}$$

i.e., $\bar{F}(J, s)$ can be viewed as the classic asynchronous value iteration update using the actual probability distribution, $p(\omega(s))$, applied to s. The proof of the following lemma closely resembles the proof of the contractive property of value iteration.

Lemma IV.5. *Let J' and J'' be two value vectors in $\mathbf{R}^{|S|}$ and let $s \in S$. Then,*

$$|\bar{F}(J', s) - \bar{F}(J'', s)| \leq \gamma \|J' - J''\|_\infty.$$

Proof.

$$\begin{aligned}
|\bar{F}(J', s) - \bar{F}(J'', s)| &= \left| \min_{a \in A_s} \left[c(s,a) + \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} p(\hat{\omega}(s)) J'(f(s,a,\hat{\omega}(s))) \right] - \right. \\
&\qquad \left. \min_{a \in A_s} \left[c(s,a) + \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} p(\hat{\omega}(s)) J''(f(s,a,\hat{\omega}(s))) \right] \right| \\
&\leq \max_{a \in A_s} \left| \left[c(s,a) + \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} p(\hat{\omega}(s)) J'(f(s,a,\hat{\omega}(s))) \right] - \right. \\
&\qquad \left. \left[c(s,a) + \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} p(\hat{\omega}(s)) J''(f(s,a,\hat{\omega}(s))) \right] \right| \\
&= \max_{a \in A_s} \left| \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} p(\hat{\omega}(s)) \left(J'(f(s,a,\hat{\omega}(s))) - J''(f(s,a,\hat{\omega}(s))) \right) \right| \\
&\leq \gamma \|J' - J''\|_\infty.
\end{aligned}$$

□

Lemma IV.6. *For any $\epsilon > 0$ and $J \in \mathbf{R}^{|S|}$ with $\|J\|_\infty \leq \frac{\tilde{c}}{1-\gamma}$ for constant $\tilde{c}$ defined as in Lemma IV.4, there exists $T < \infty$ such that:*

$$|F_t(J, s) - \bar{F}(J, s)| < \epsilon, \ \forall t \geq T, \ \forall s \in S \text{ with probability 1.} \tag{4.7}$$

Proof. Let s be an arbitrary state in S. Then,

$$\begin{aligned}
|F_t(J, s) - \bar{F}(J, s)| &= \left| \min_{a \in A_s} \left[c(s,a) + \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} p_t(\hat{\omega}(s)) J(f(s,a,\hat{\omega}(s))) \right] - \right. \\
&\qquad \left. \min_{a \in A_s} \left[c(s,a) + \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} p(\hat{\omega}(s)) J(f(s,a,\hat{\omega}(s))) \right] \right| \\
&\leq \max_{a \in A_s} \left| \left[c(s,a) + \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} p_t(\hat{\omega}(s)) J(f(s,a,\hat{\omega}(s))) \right] - \right. \\
&\qquad \left. \left[c(s,a) + \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} p(\hat{\omega}(s)) J(f(s,a,\hat{\omega}(s))) \right] \right| \\
&= \max_{a \in A_s} \left| \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} J(f(s,a,\hat{\omega}(s)))(p_t(\hat{\omega}(s)) - p(\hat{\omega}(s))) \right| \\
&\leq \gamma \frac{\tilde{c}}{1-\gamma} \sum_{\hat{\omega}(s) \in \Omega(s)} \left(p_t(\hat{\omega}(s)) - p(\hat{\omega}(s)) \right).
\end{aligned}$$

By assumption, $|\Omega(s)| < \infty, \forall s \in S$. Moreover, by Lemma IV.3 $\lim_{t\to\infty} p_t(\hat{\omega}(s)) \to p(\hat{\omega}(s))$ for all $s \in S$ with probability 1. Thus, the term on the right hand side of the above inequality converges to 0 with probability 1. Therefore, for any J with $\|J\|_\infty \leq \frac{\tilde{c}}{1-\gamma}$,

$$\lim_{t\to\infty} |F_t(J,s) - \bar{F}(J,s)| = 0, \ \forall s \in S.$$

This implies, for any $\epsilon > 0$ and any J with $\|J\|_\infty \leq \frac{\tilde{c}}{1-\gamma}$, $\exists\ T < \infty$ such that:

$$|F_t(J,s) - \bar{F}(J,s)| < \epsilon, \ \forall t \geq T \text{ with probability 1.}$$

□

Lemma IV.6 states that the functions $F_t(\cdot)$ pointwise converges to the function $\bar{F}(\cdot)$. Next, we show that the sequence J_t obtained by applying F_t starting from an arbitrary initial value vector J_0 converges to J^*.

Lemma IV.7. *Let J_0 be a finite initial value vector and let $\tilde{c}$ be the finite constant defined in Lemma IV.4:*

$$\tilde{c} = \max\{\bar{c}, (1-\gamma)\|J_0\|_\infty\}.$$

Fix $\epsilon > 0$ and choose $T < \infty$ such that,

$$|F_t(J,s) - \bar{F}(J,s)| \leq \epsilon, \ \textit{with probability 1} \tag{4.8}$$

$\forall t \geq T, \ \forall s \in S$ and $\forall J$ with $\|J\|_\infty \leq \frac{\tilde{c}}{1-\gamma}$.

Define a sequence J_t^ϵ as follows:

$$J_t^\epsilon(s) = \begin{cases} J_t(s) & \textit{if } t \leq T, \ \forall s \ \in S, \\ J_{t-1}^\epsilon(s) & \textit{if } t > T \textit{ and } s \neq s_t, \\ \bar{F}(J_{t-1}^\epsilon, s_t) & \textit{if } t > T \textit{ and } s = s_t. \end{cases}$$

Then,

$$\|J_t^\epsilon - J_t\|_\infty \leq \frac{\epsilon}{1-\gamma},\ \forall t \geq 0 \textit{ with probability 1.}$$

Proof. By lemma IV.4, $|J_t(s)| \leq \frac{\tilde{c}}{1-\gamma}$, $\forall t \in \{1, 2, \ldots\}$ and $\forall s \in S$. Also by lemma IV.6, a finite T specified as above always exists.

By definition, $|J_t^\epsilon(s) - J_t(s)| = 0$, for $t \leq T$, $\forall s \in S$. Therefore, we only need to consider $t > T$. The proof is by induction. Let $t = T+1$ and consider the two cases, $s \neq s_{T+1}$ and $s = s_{T+1}$, where s_t represents the state of SFPL at iteration t:

Case 1: $s \neq s_{T+1}$:

$$|J_{T+1}^\epsilon(s) - J_{T+1}(s)| \ = \ |J_T^\epsilon(s) - J_T(s)| = |J_T(s) - J_T(s)| = 0,$$

which follows directly from the definitions of J_t^ϵ and J_t.

Case 2: $s = s_{T+1}$:

$$\begin{aligned} |J_{T+1}^\epsilon(s) - J_{T+1}(s)| &= |\bar{F}(J_T^\epsilon, s_{T+1}) - F_{T+1}(J_T, s_{T+1})| && (4.9)\\ &= |\bar{F}(J_T, s_{T+1}) - F_{T+1}(J_T, s_{T+1})| && (4.10)\\ &\leq \epsilon \leq \frac{\epsilon}{1-\gamma} && (4.11) \end{aligned}$$

where equations (4.9) and (4.10) directly follow from the definitions of J_t^ϵ and J_t and inequality (4.11) follows from Equation (4.8).

Now assume $\|J_t^\epsilon - J_t\|_\infty \leq \frac{\epsilon}{1-\gamma}$, $\forall t \leq T+k$ and $\forall s \in S$ for some $k \geq 1$, and consider the following cases for $t = T+k+1$:

Case 1: $s \neq s_{T+k+1}$:

$$|\bar{J}_{T+k+1}^\epsilon(s) - J_{T+k+1}(s)| = |\bar{J}_{T+k}^\epsilon(s) - J_{T+k}(s)| \leq \frac{\epsilon}{1-\gamma},$$

by the inductive hypothesis.

Case 2: $s = s_{T+k+1}$:

$$
\begin{aligned}
&|J^{\epsilon}_{T+k+1}(s) - J_{T+k+1}(s)| = && (4.12)\\
&\quad = |J^{\epsilon}_{T+k+1}(s_{T+k+1}) - J_{T+k+1}(s_{T+k+1})| && (4.13)\\
&\quad = |\bar{F}(J^{\epsilon}_{T+k}, s_{T+k+1}) - F_{T+k+1}(J_{T+k}, s_{T+k+1})| && (4.14)\\
&\quad = |\bar{F}(J^{\epsilon}_{T+k}, s_{T+k+1}) - \bar{F}(J_{T+k}, s_{T+k+1}) + \bar{F}(J_{T+k}, s_{T+k+1}) - F_{T+k+1}(J_{T+k}, s_{T+k+1})| && (4.15)\\
&\quad \leq |\bar{F}(J^{\epsilon}_{T+k}, s_{T+k+1}) - \bar{F}(J_{T+k}, s_{T+k+1})| + |\bar{F}(J_{T+k}, s_{T+k+1}) - F_{T+k+1}(J_{T+k}, s_{T+k+1})| && (4.16)\\
&\quad \leq \gamma \|J^{\epsilon}_{T+k} - J_{T+k}\|_{\infty} + \epsilon && (4.17)\\
&\quad \leq \frac{\gamma\epsilon}{1-\gamma} + \epsilon && (4.18)\\
&\quad = \frac{1}{1-\gamma}\epsilon && (4.19)
\end{aligned}
$$

where inequality (4.16) is by triangle inequality. Inequality (4.17) follows from Lemmas IV.5 and IV.6. The inequality (4.18) follows from the inductive hypothesis.

□

Note that, J^{ϵ}_t is obtained by applying asynchronous value iteration after an iteration $T < \infty$. Since all states are visited infinitely often by Assumption IV.1, J^{ϵ}_t converges to the optimal solution of MDP [13].

Theorem IV.8.

$$\lim_{t\to\infty} J_t(s) = J^*(s), \ \textit{with probability 1}, \forall s \in S.$$

Proof. By Lemma IV.7, for any sequence J_t, we can find a sequence $J^{\epsilon}_t(s)$ that is arbitrarily close to $J_t(s)$, $\forall s \in S$. Since J^{ϵ}_t converges to J^*, so does J_t:

$$\lim_{t\to\infty} J_t = J^* \text{ with probability 1.}$$

□

In the proof of Theorem IV.8, it is assumed that the algorithm visits every state infinitely often due to the inherent random disturbance in the system. It has been shown that our estimate of the values for each state converges to the optimal values. However, we are not always required to find the optimal values of all states. For

example, in the windy gridworld problem [5], which involves finding the fastest route from a starting point to a destination point on a grid subject to wind with a random strength, we only need to find the optimal values (and actions) of the states that are visited with positive probability under the optimal policy (see Section 4.5 for the details of the windy gridworld problem). In other words, if the system is simulated under the optimal policy infinitely long[2], some states will be visited infinitely often (*i.e.*, will form a recurrent class of the induced Markov Chain under the optimal policy). In problems like windy gridworld, we are only interested in the values of the states in this recurrent class. Below, we argue that SFPL is able to find the optimal actions of the states that are visited infinitely often under the optimal policy by setting $J_0(s) < J^*(s)$, $\forall s \in S$, even if the Assumption IV.1 does not hold.

First, recall that the history of a state is populated by the actions obtained via best reply calculations in that state and the next state the system transitions into is defined by the action sampled from the history of the current state and the random disturbance observed during the current iteration. Setting $J_0(s) < J^*(s)$, $\forall s \in S$, implies that our initial estimate of the value of any given state is optimistic (*i.e.*, lower than the optimal value) and its low value gives incentive to choose actions that lead to that state in the best reply phase. Therefore, during the initial iterations of the SFPL, most of the neighboring states of a state will be visited due to their low value estimates. In other words, each state will try to explore and choose diverse actions initially. However, gradually, they will start choosing the "right" actions and therefore their histories will begin to be dominated by these actions. The numerical results on an instance of gridworld problem given in Section 4.5 also shows that SFPL visits the states in the "recurrent class" of the induced Markov Chain under

[2]Note that if it is an optimal stopping problem or has an absorbing state like windy gridworld, we would continue the simulation by jumping into the initial state

the optimal policy more often than other states and discovers the optimal actions and therefore the optimal values of these states.

4.5 Numerical Results

We use the dynamic location problem adapted from [43] and windy gridworld problem described in [5] to test the performance of the SFPL algorithm compared to Q-Learning and SARSA[3] algorithms. In this section, we first give detailed descriptions of Q-Learning and SARSA and then we present our numerical results on the two example problems.

4.5.1 Q-Learning and SARSA

Q-Learning is a learning algorithm designed for MDPs where either the transition probability distribution is unknown or calculating the expected value of future decisions is computationally intractable [39]. Q-Learning is an off-policy algorithm that estimates the value of each state-action pair using the following update formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, s_t) + \alpha[c(s_t, a_t) + \gamma \min_{a \in A_{s_{t+1}}} Q(s_{t+1}, a) - Q(s_t, a_t)]$$

where (s_t, a_t) is the state-action pair that is being updated at iteration t, s_{t+1} is the next state the system transitions to from state s_t under action a_t, and α is the step size parameter.

The formal description of the Q-Learning algorithm is as follows [5]:

- Initialization:
  - Assign initial values to $Q(s, a)$ for all $s \in S$, $a \in A_s$.
  - Set $t = 1$ and observe $s_1 \in S$ (*i.e.*, s_1 is the state in which the process begins).

[3]The name SARSA stands for "State-Action-Reward-State-Action".

- Iteration t:

  1. Find a decision $a_t \in A_{s_t}$ using a policy derived from Q (*e.g.*, ϵ-greedy).
  2. Execute decision a_t, observe the immediate cost $c(s_t, a_t)$ and the next state s_{t+1}.
  3. Update $Q(s_t, a_t)$ using:

$$Q(s_t, a_t) \leftarrow Q(s_t, s_t) + \alpha \left[c(s_t, a_t) + \gamma \min_{a \in A_{s_{t+1}}} Q(s_{t+1}, a) - Q(s_t, a_t) \right] \quad (4.20)$$

  4. Set $t \leftarrow t + 1$ and go to Step 1.

The formal description of SARSA algorithm, a variant of Q-Learning, is as follows [5]:

- Initialization:
  - Assign initial values to $Q(s, a)$ for all $s \in S$, $a \in A_s$.
  - Set $t = 1$ and observe $s_1 \in S$ (*i.e.*, s_1 is the state in which the process begins).
  - Find a decision $a_1 \in A_{s_1}$ using a policy derived from Q (*e.g.*, ϵ-greedy).
- Iteration t:
  1. Execute decision a_t, observe the immediate cost $c(s_t, a_t)$ and the next state s_{t+1}.
  2. Find a decision $a_{t+1} \in A_{s_{t+1}}$ using the policy derived from Q (ϵ-greedy).
  3. Update $Q(s_t, a_t)$ using:

$$Q(s_t, a_t) \leftarrow Q(s_t, s_t) + \alpha \left[c(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \quad (4.21)$$

  4. Set $t \leftarrow t + 1$ and go to Step 1.

As can be seen above, the difference between SARSA and Q-Learning is that the term $\min_{a \in A_{s_{t+1}}} Q(s_{t+1}, a)$ in the Q-Learning update formula (equation 4.20) is replaced with the term $Q(s_{t+1}, at + 1)$ in SARSA (equation 4.21) [13]. In other words, Q-Learning updates the value of a state-action pair independent of the decision chosen in the current iteration, whereas SARSA uses the decision it chooses directly to update the values.

Given the step size parameter α satisfies usual stochastic approximation conditions and every state-action pair is visited infinitely often, the values returned by both Q-Learning and SARSA algorithms are proven to converge to the optimal values with probability one [5].

4.5.2 Dynamic Location Problem

The dynamic location problem analyzed in this section is adapted from the work of Rosenthal et al. in [43]. In this problem a workforce that serves Q facilities moves between these facilities according to a stationary probability distribution denoted by probability transition matrix P, where P_{ij} is the probability that the workforce moves from facility i to facility j. An equipment trailer can be located at any of the facilities in Q. The cost of moving the trailer from facility i to a facility j is denoted by R_{ij}. If the workforce is located at facility i and the trailer is at facility j, then U_{ij} denotes the cost of obtaining the equipment from the trailer. The problem is modeled as an infinite horizon MDP by defining the states to be the tuple (w, r), where w and r are the location of the workforce and the trailer, respectively. We assume that this system does not have an absorbing state and thus runs indefinitely.

We used a problem with 4 facilities and therefore 16 states. We used the following data for the cost of trailer relocation, the cost of using the equipment when the trailer

and the workforce are at different locations:

$$R = \begin{pmatrix} 0 & 100 & 200 & 300 \\ 100 & 0 & 100 & 200 \\ 200 & 100 & 0 & 100 \\ 300 & 200 & 100 & 0 \end{pmatrix}, \; U = \begin{pmatrix} 0 & 999 & 999 & 999 \\ 200 & 0 & 100 & 100 \\ 200 & 100 & 0 & 100 \\ 200 & 100 & 100 & 0 \end{pmatrix}.$$

We also used the following probability transition matrix for the movement of the work crew:

$$P = \begin{pmatrix} 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0.6 & 0.4 \\ 0.1 & 0.4 & 0.1 & 0.4 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix}.$$

It is important to note that the knowledge of the probability distribution of the workforce location (*i.e.*, the random disturbance) given above is only used to simulate the process and to calculate the optimal solution of the problem. SFPL and Q-Learning do not use this information but estimate the probability distribution through sampling.

In this section we compare the performances of SFPL and Q-Learning on the Dynamic Location problem. For the Q-Learning algorithm we set the ϵ-greedy sampling parameter $\epsilon = 0.1$ and the step size $\alpha = 1/I(s,a)$ where $I(s,a)$ is the number of times the value of state-action pair (s,a) is updated. The convergence properties of stochastic search algorithms (including SARSA and Q-Learning) strongly depend on the parameters ϵ and α. George et al. lists various step size parameters and discusses the effect of using different step sizes in [25]. The selection of such parameters is an active research area and beyond the scope of our experiments. The results presented below are obtained for fixed parameters.

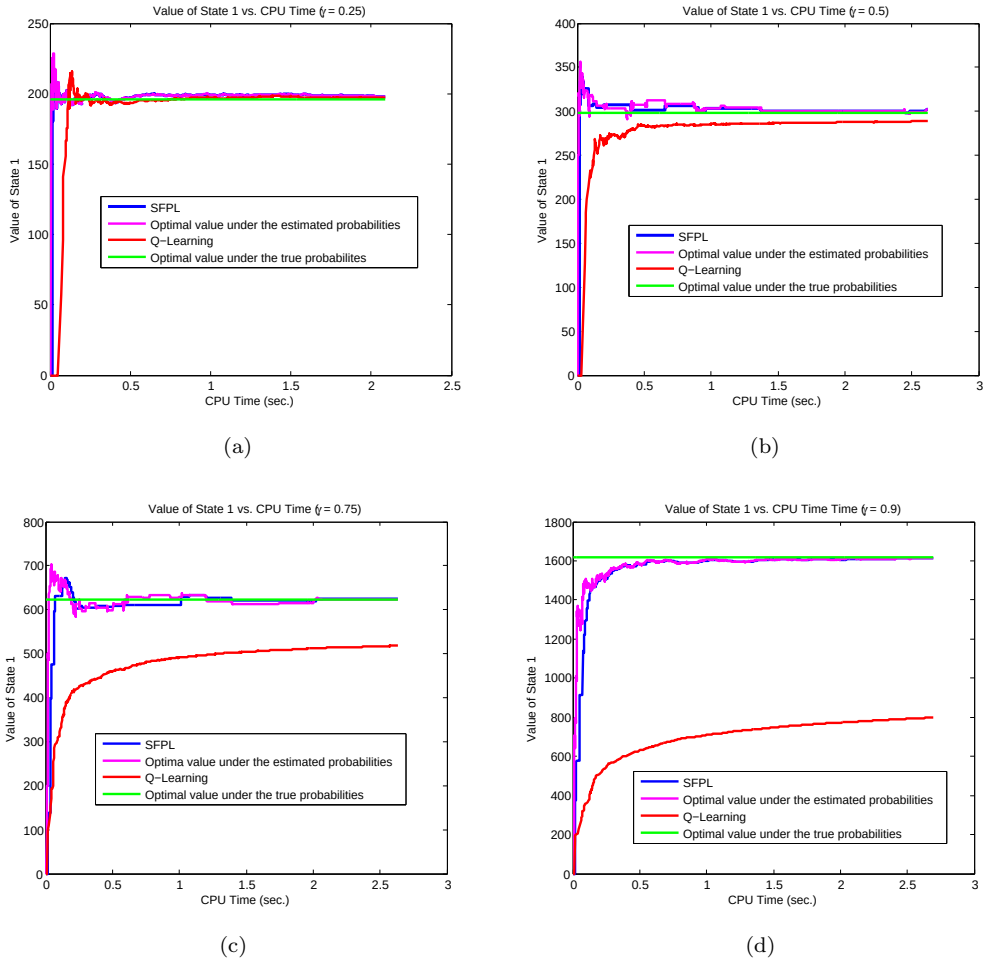
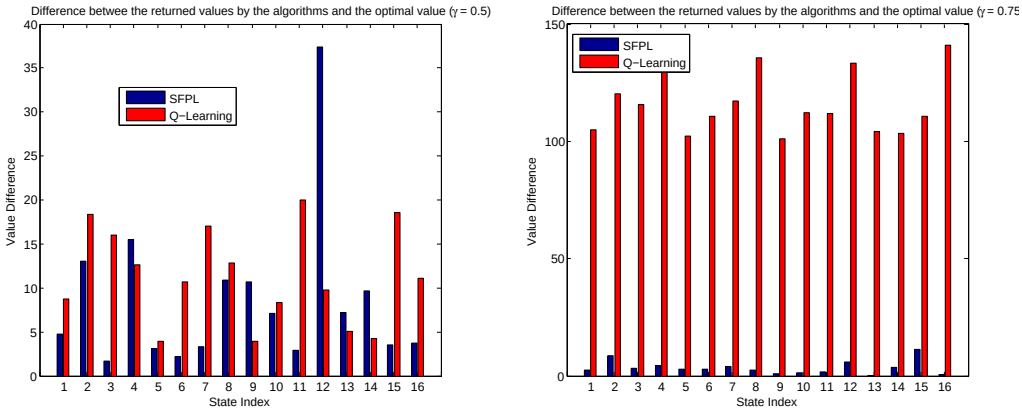


Figure 4.1: Comparison of the values returned by SFPL, Q-Learning and the optimal values of MDP and MDP_t (a) $\gamma = 0.25$, (b) $\gamma = 0.50$, (c) $\gamma = 0.75$, (d) $\gamma = 0.9$.



(a) (b)

Figure 4.2: Comparison of difference of the values returned by SFPL, Q-Learning and the optimal values of each state for (a) $\gamma = 0.5$, (b) $\gamma = 0.75$.

Figure 4.1 shows the comparison of the values returned by SFPL and Q-Learning for state $(1, 1)$ when both algorithms are run for the same amount of CPU time. On this figure the optimal value of state $(1, 1)$ in MDP (the system with the true probability distribution $p(\omega(s))$) and in MDP_t (the estimated system at time t with the probability distributions $p_t(\omega(s))$) are also plotted for comparison. Note that calculating the optimal solution of MDP_t is not part of SFPL. However, this comparison provides valuable insights about the performance of SFPL. For example, as shown in Figure 4.1 values returned by SFPL are very close to the optimal solution of the estimated system MDP_t after a certain time. This implies that SFPL starts choosing the best actions that can possibly be obtained with the current knowledge of the system.

The results presented in Figure 4.1 are obtained using various discount rates ($\gamma = 0.25$, $\gamma = 0.50$, $\gamma = 0.75$ and $\gamma = 0.9$). As can be seen from the figures, the performances of Q-Learning and SFPL are comparable for lower discount rates ($\gamma = 0.25$ and $\gamma = 0.50$). However, SFPL outperforms Q-Learning for higher discount

rates ($\gamma = 0.75$ and $\gamma = 0.9$).

Figure 4.2 shows the absolute value difference between the final values returned by SFPL and Q-Learning and the optimal values $J^*(s)$ of all states for $\gamma = 0.5$ and $\gamma = 0.75$. As before, the performance of the two algorithms is comparable for lower discount rates; however, SFPL performs better for higher discount rates.

4.5.3 Windy Gridworld

The windy gridworld problem involves going from one point to another on a grid each column of which is subject to winds with various strengths. We use the grid given in [5], also shown in Figure 4.3. In the figure, the numbers below each column represents the mean strength of the wind blowing towards north (upward) affecting only that column. We assume that in each column the wind strength can be one unit above, one unit below, or at its mean value equally likely at any given time[4]. Therefore, if the mean strength of the wind is 0 units, then with 1/3 probability the wind blows in the south direction with unit strength, with 1/3 probability the wind blows in north direction with unit strength and with 1/3 probability there is no wind in effect. In each state the feasible actions are to move in one of the eight possible directions, namely East, Southeast, South, Southwest, West, Northwest, North and Northeast. Once a decision is made in a given state, at a given time, the next state is determined by the current state (*i.e.*, the current location), the decision at the current state (*i.e.*, the movement direction) and the realization of the wind strength at that time. If the process transitions into an infeasible state (*i.e.*, out of the boundaries of the grid), then it is forced to jump back to the closest state within the boundaries. The strength of the wind affecting a column is random and possibly changes every time a state in that column is visited. As shown in Figure 4.3 the

[4]Note that the probability distribution of the random disturbance is only used to simulate the process and to calculate the optimal solution of the problem

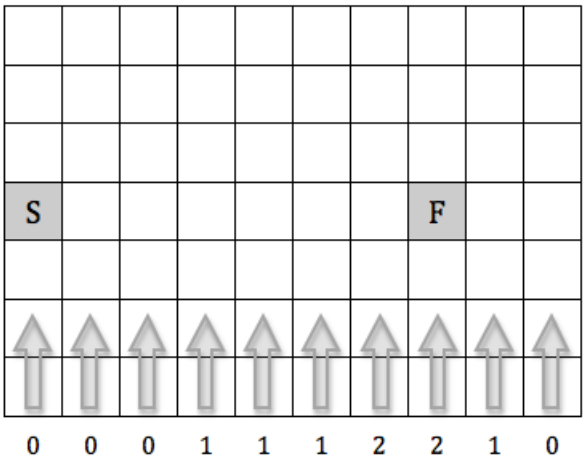


Figure 4.3: Windy GridWorld Problem.

problem is to go from point S to point F as fast as possible. The immediate reward of each decision is defined to be -1 units and the objective is to maximize the total reward (*i.e.*, minimize the total travel distance).

Note that windy gridworld problem does not satisfy Assumption IV.1. In this problem, the process ends once the destination state is reached. In the context of infinite horizon problems, we can view the destination state as an absorbing state *i.e.*, the system is stuck in the destination state, independent of the policy employed, once the process enters this state. Therefore, some states may not be visited infinitely often under some stationary deterministic policies. However, with small modifications, details of which are given below, it is possible to apply SFPL to this problem. As we will discuss further, the numerical results obtained from applying SFPL to the windy gridworld supports our claim from Section 4.4 that the values of the states in the "recurrent class" of the induced Markov Chain under the optimal policy converge to the optimal values. In what follows, the details of the SFPL implementation on the windy gridworld and the discussion of the numerical results are given.

In the implementation of SFPL on the windy gridworld problem, to facilitate

learning of the algorithm, the system is restarted at the initial state (point S) once the destination (point F) is reached. One complete run, starting from the initial state S to the final state F, is called an "episode".

We compared the performance of SFPL with that of SARSA [5]. We used the implementation code for SARSA developed by John Weatherwax available online at [46]. For the SARSA algorithm we set the ϵ-greedy sampling parameter $\epsilon = 0.1$ and the step size $\alpha = \frac{1}{I(s,a)}$ where $I(s,a)$ is the number of times the value of state-action pair (s,a) is updated.

In SFPL, the values of all states are initialized at value 0. We ran SFPL and SARSA for one minute of CPU time. During this time, SFPL completed around 2000 episodes, whereas SARSA completed around 15000 episodes (*i.e.*, the system starts at the initial state and reaches the final state 15000 times). While running SFPL, we also calculated the optimal solution of MDP_t by running the classical synchronous value iteration algorithm with the most up-to-date estimate of the probability distribution on the wind strength every time the system entered the initial state[5].

Figure 4.4 shows the value estimates of the initial state S returned by SFPL and SARSA every time these algorithms hit this state. The optimal value of S under the true probability distribution and also under the estimated probability distributions (*i.e.*, the optimal value of S in MDP_t) are also plotted for comparison. Note that SFPL finds values closer to the optimum faster than SARSA.

Running SFPL for 2000 episodes took around 1 minute of CPU time; whereas running value iteration every time the system entered the initial state took several hours to complete. As can be seen in Figure 4.5, the values returned by SFPL and the optimal values of MDP_t obtained via value iteration are not significantly different.

[5]We stopped the clock during the value iteration algorithm so that we were able to record the time requirement of SFPL exclusively.

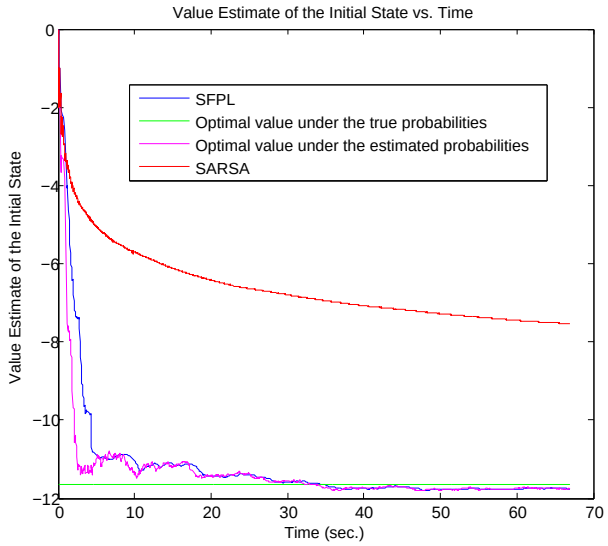


Figure 4.4: The value estimates of the initial state S returned by SFPL, SARSA and by running value iteration on the intermediate estimated system MDP_t. The optimal value under the true probabilities is also included for comparison.

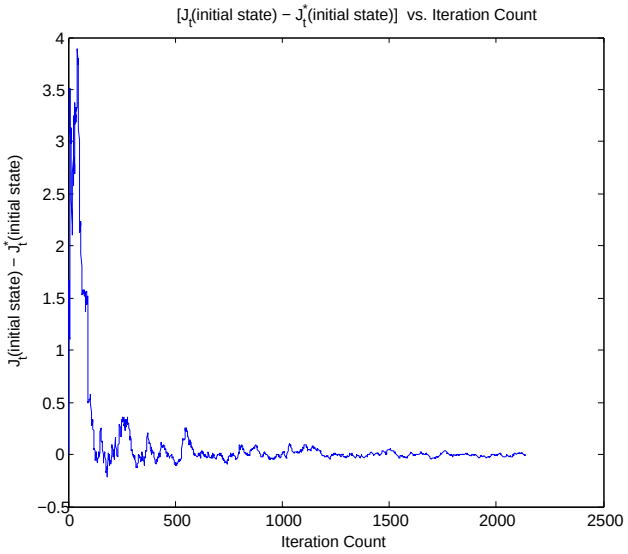
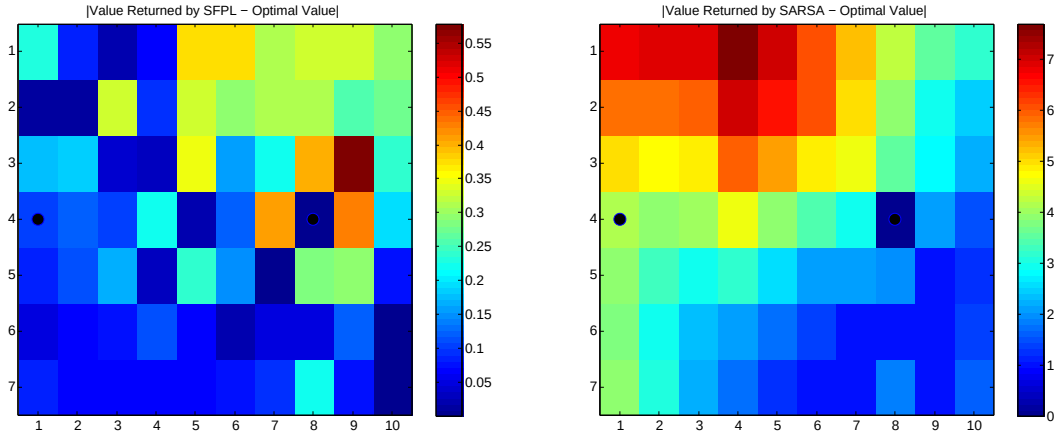


Figure 4.5: Difference between the value estimate of the initial state S by SFPL and the optimal values of the estimated system MDP_t calculated using value iteration.

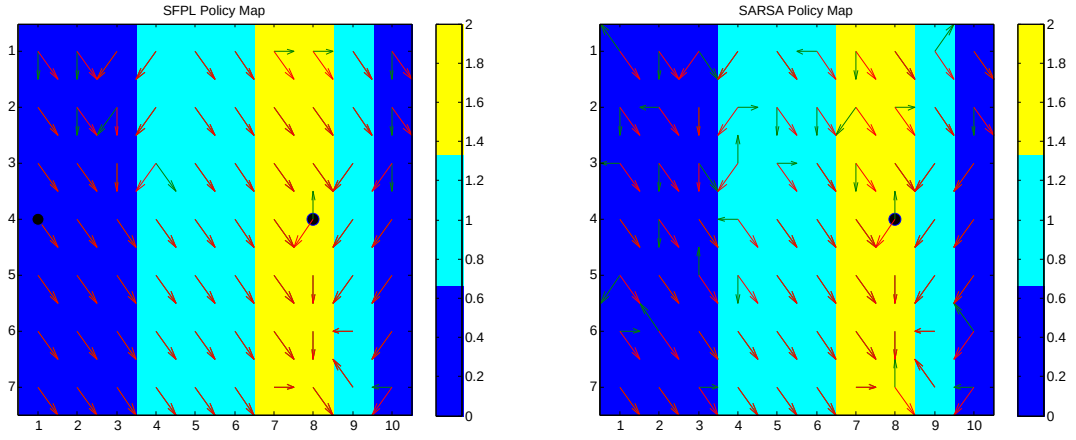


(a) (b)

Figure 4.6: Absolute value difference of the final estimated values returned by SFPL (a) and SARSA (b) and the optimal values under the true probability distribution. The graphs are adapted from the code developed by John Weatherwax available online at [46].

Figure 4.6 shows the absolute value difference between the final estimated values returned by SFPL and SARSA and the optimal values under the true probability distribution. As can be seen in the figure, the blue color is dominant on the states that are to the south of the initial and final states, denoted by large black points, in both figures. This is because both algorithms sample the optimal strategy, which is to move in the southeast direction (also shown in Figure 4.7), more often than other available strategies and therefore the value estimates of the states to the south of the grid get closer to the optimal values. These figures also show that error of the value estimates obtained by SFPL is smaller than that of SARSA on all states.

Another way to compare algorithms is to check if the algorithms successfully identify the optimal actions in each state. The information on the policy returned by SFPL and SARSA is presented in Figure 4.7. In this figure, different colors on the grid represent the nominal value of the wind on each column. For example, on the yellow states the mean wind strength is 2 units towards north, whereas on light blue



(a) (b)

Figure 4.7: The final policy returned by SFPL (a) and SARSA (b) (represented with black arcs) vs. the optimal policy obtained using the values returned by value iteration (represented with red arcs).

and dark blue states the mean strength of the wind is 1 and 0 units towards north, respectively. On these figures the red arrows on each state represent the optimal action whereas the black arrows denote the action returned by the algorithms. If the actions from the algorithms coincide with that of the optimal action for a given state, than only the red arc is shown. As presented in figure, SFPL found the correct actions for more states than SARSA.

In each iteration of SARSA and Q-Learning, the value assigned to feasible actions of the current state is compared and the one with the lowest value (or the highest, if it is a maximization problem) is selected. On the other hand, in SFPL, the best action in the current state is obtained by calculating and comparing the expected value of each feasible action of the current state (see Equation (4.2)). Note that while the computational requirement of SARSA and Q-Learning per iteration is significantly less than that of SFPL, the magnitude of improvement obtained in each iteration of SFPL is higher. Therefore, to make the claims more meaningful, the numerical

results presented in this section are obtained by running the algorithms for the same amount of CPU time.

Another important property that affects an algorithm's performance is its memory requirement. Q-Learning and SARSA update the values of each state-action pair. Therefore, these algorithms need to keep a data structure of size $O(|S| \times \max_s |A_s|)$. In SFPL, the history of best replies and the history of random disturbances are recorded. However, this data is only used to sample actions (from the history of best replies) and to estimate the probability distribution of the random disturbances. Therefore, in these algorithms, it is enough to keep the count of each action in the history of best replies and the count of each random event in the history of random disturbances. Therefore, the memory requirement of SFPL is $O(|S| \times \max_s(\max\{|A_s|, |\Omega(s)|\}))$. In conclusion, the memory requirement of SFPL may be more than those of Q-Learning and SARSA depending on the problem structure.

4.6 Conclusions

SFPL is an asynchronous, online learning algorithm designed using SFP concepts for discounted, homogenous, infinite horizon MDPs. This algorithm is shown to converge to the optimal solution by updating beliefs on the probability distribution of the random disturbance in the system and the optimal value of each state, at the same time adaptively sampling actions from the history of best replies.

Comparing SFPL with SARSA and Q-Learning provides insights in many respects. First of all, one significant difference between the two classes of algorithms is that SFPL uses the information on the probability distribution of the random disturbance in the system whereas SARSA and Q-Learning do not. The difference between

numerical results of SFPL and SARSA (and Q-Learning) suggest that incorporating the extra knowledge about the system into the approximation procedures, when possible, can improve the performance of the algorithms. Secondly, using the history of best replies to sample actions helps to discover the states in the "recurrent class" of the induced Markov Chain under the optimal policy and therefore find better values (*i.e.*, values close to the optimal) for these states. The theoretical analysis of the effect of using history of best replies is still an open research question, however, the empirical experiments show promising results. The results obtained with using various discount rates also suggest that SARSA and Q-Learning algorithms are more influenced by discount rate than SFPL.

CHAPTER V

Conclusions and Future Research Directions

5.1 Summary and Conclusions

In this dissertation, we considered two classes of sequential decision making problems: (a) discrete, deterministic and finite horizon DP problems and (b) stochastic, discounted, infinite horizon MDPs. Chapters II and IV are dedicated to the analysis of Sampled Fictitious Play (SFP) based algorithms designed for the two classes of problems listed above. In these chapters, the details of the developed algorithms are given first, which are followed by the convergence results and the numerical experiments on their performance. Chapter III presents numerical results obtained by applying the algorithms developed in Chapter II to the Traveling Salesman Problem and discusses some modifications that might potentially improve their performance.

The algorithms studied in Chapter II, namely RSFP and SFPLS, are stochastic search algorithms for discrete, deterministic and finite horizon DP problems. As noted earlier, stochastic algorithms cannot outperform the label correcting algorithms such as Dijsktra's Algorithm in theory if the requirement is to find an optimal solution. However, on large-scale problems, Dijsktra's algorithm loses its effectiveness, and it usually is unable to find the optimal solution within a reasonable time limit as required by most real-world applications. For such problems, we propose

using SFP methods to find close to optimal solutions quickly. As presented in Chapter II, SFP based search algorithms not only converge to the optimal solution but also find close to optimal solutions quickly. The potential of these algorithms is also supported by numerical experiments presented in Chapter III highlighting the performance improvement after small modifications.

In Chapter IV, we propose using SFP concepts to design an online learning algorithm for infinite horizon, discounted MDPs, where the probability distribution of the exogenous random disturbance is unknown. The learning algorithm, called Sampled Fictitious Play based Learning Algorithm (SFPL), uses the information (*i.e.*, frequency of the random events) obtained by observing the system to find the best possible decision in each iteration. This algorithm is proven to converge to the optimal solution, and the numerical results based on the dynamic location and windy gridworld problems show the algorithm converges quickly. The numerical experiments also suggest that learning the system dynamics and simultaneously optimizing the system variables is more efficient than running these two routines separately (*i.e.*, first observing the system and collecting data on unknown parameters and then calculating the optimal solution with the estimated parameters). We also present comparisons of SFPL with two algorithms from the Reinforcement Learning literature, Q-Learning and SARSA (slightly modified version of Q-Learning). These comparisons show that the performance of Q-Learning strongly depends on the step size parameter while SFPL does not use a step size parameter and that Q-Learning algorithm is more susceptible to discount rate of the problem than SFPL.

The advantages of using SFP based algorithms for sequential decision making problems are as follows:

- The SFP approach does not require a closed form representation of an optimiza-

tion model. The algorithms presented can even be applied to complex black-box optimization problems.

- The SFP based algorithms developed in this dissertation do not rely on structures of costs, transition functions and objective function. Therefore, these algorithms are applicable to a wide range of problems.

- The SFP based algorithms do not need the knowledge of the entire state space (*e.g.*, network topology). They discover the state space of a given problem gradually. It is a well known fact that the entire feasible region has to be searched in order to confirm that an optimal solution has been found. However, with our approach, it is possible to find good solutions by only partially discovering the state space.

- To model a DP problem as a game, we define states to be the players and the objective function to be the payoff function for each player. Therefore, the best reply calculations of players naturally divides the large-scale problem into manageable subproblems which can be solved in parallel.

- The parallel processing possibility, in some cases, is one of the considerations for applying these algorithms to large-scale problems, since a parallel implementation can increase the performance of the algorithms substantially.

5.2 Future Research Directions

5.2.1 Future Research Directions for the work in Chapter II

- *Analysis of equilibrium solutions in the game defined on a DP problem:*

  The number of NEs and their distance to the optimal solution depend on the problem type, data of the problem, and the topology of the network representation (*i.e.*, the DP network) of the problem. If the number of NEs is large

or if they are densely located on certain parts of the network, identifying the optimal solution among these NEs will be more time consuming compared to problems with fewer NEs. Therefore, the ability to estimate the number of equilibrium strategies in a problem or identify the parts of the network where the NEs are densely located can help us predict the performance of the SFP based algorithms on a given problem instance before implementing it.

- *Relationship between a NE returned by SFP and the initial point:*

  For a given initial point, there is only a subset of NEs that the SFP might converge to. For example, SFP will not return a NE having a worse objective value. Numerical experiments suggests that this subset is actually smaller compared to the entire set of NEs. Therefore, analyzing the nature of the relationship between the NEs returned by SFP and the initial point might inspire new ideas to improve performance of SFP.

- The proof of convergence results for the modified algorithms developed in Chapter III are omitted and are among the future research directions we will pursue.

5.2.2 Future Research Directions for the work in Chapter IV

- *Analysis of the effect of using history of best replies:*

  In SFPL, every player samples an action from the history of best replies. Numerical results suggest that this approach helps the algorithm visit "good" states (states that are in the "recurrent class" of induced Markov Chain under the optimal policy) more often and therefore learn the system dynamics related to these states quickly. Our convergence analysis, however, does not capture this property of the algorithm. Extending the convergence analysis to show that the SFPL algorithm finds the optimal solution of the states, which are in

the recurrent class of the induced Markov Chain, without visiting all states infinitely often (possibly on a certain subclass of problems) would be a significant contribution.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Test Data for Traveling Salesman Problem. `http://http://www.tsp.gatech.edu/world/countries.html`.

[2] R. Ahuja and D. Hochbaum. Solving linear cost dynamic lot-sizing problems in $o(n \log n)$ time. *Operations Research*, 56(1):255–261, 2008.

[3] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2007.

[4] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[5] A. Barto and R. Sutton. *Reinforcement Learning: An Introduction*. MIT Press Cambridge, MA, 1998.

[6] S. Baumert, S. Cheng, A. Ghate, D. Reaume, D. Sharma, and R. Smith. Joint optimization of capital investment, revenue management, and production planning in complex manufacturing systems. Technical report, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, 2005.

[7] J. C. Bean, J. R. Lohmann, and R. L. Smith. Equipment replacement under technological change. *Naval Research Logistics*, 41:117–128, 1994.

[8] R. Bellman. On some applications of the theory of dynamic programming to logistics. *Naval Research Logistics Quarterly*, 1, 1954.

[9] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[10] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, 1962.

[11] R. E. Bellman and S. E. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, 1962.

[12] D. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, 1976.

[13] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[14] D. Bertsimas and R. Weismantel. *Optimization Over Integers*. Dynamic Ideas, 2005.

[15] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press New York, NY, USA, 1998.

[16] G. Brown. Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13:374–376, 1951.

[17] H. Chang, M. Fu, J. Hu, and S. Marcus. An adaptive sampling algorithm for solving Markov decision processes. *Operations Research*, 53:126–139, 2005.

[18] H. Chang, M. Fu, J. Hu, and S. Marcus. *Simulation-based Algorithms for Markov Decision Processes*. Springer Berlin, 2007.

[19] S. Cheng, M. Epelman, and R. Smith. CoSIGN: A Parallel Algorithm for Coordinated Traffic Signal Control. *IEEE Transactions on Intelligent Transportation Systems*, 7:551–564, 2006.

[20] E. Denardo. *Dynamic Programming: Models and Applications*. Dover, 2003.

[21] A. Federgruen and M. Tzur. A simple forward algorithm to solve general dynamic lot sizing models with n periods in $0(n \log n)$ or $0(n)$ time. *Management Science*, pages 909–925, 1991.

[22] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

[23] A. Garcia, S. Patek, and K. Sinha. A decentralized approach to discrete optimization via simulation: Application to network flow. *Operations Research*, 55:717, 2007.

[24] A. Garcia, D. Reaume, and R. Smith. Fictitious play for finding system optimal routings in dynamic traffic networks. *Transportation Research Part B*, 34:147–156, 2000.

[25] A. George and W. Powell. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine Learning*, 65:167–198, 2006.

[26] A. Ghate, R. L. Smith, and M. Epelman. Sampled sampled fictitious play for complex systems optimization. Technical report, Technical Report, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, 2005.

[27] V. Gullapalli and A. Barto. Convergence of indirect adaptive asynchronous value iteration algorithms. *Advances in neural information processing systems*, pages 695–695, 1994.

[28] G. Gutin and A. P. Punnen. *The Traveling Salesman Problem and Its Variations*. Springer, 2002.

[29] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[30] R. A. Howard. Dynamic programming. *Management Science*, pages 317–348, 1966.

[31] M. Kearns and S. Singh. Finite-sample convergence rates for q-learning and indirect algorithms. *Advances in Neural Information Processing Systems*, pages 996–1002, 1999.

[32] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49:209–232, 2002.

[33] T. Lambert III, M. Epelman, and R. Smith. A fictitious play approach to large-scale optimization. *Operations Research*, 53:477–489, 2005.

[34] C. Lee, M. Epelman, C. White, and Y. Bozer. A shortest path approach to the multiple-vehicle routing problem with split pick-ups. *Transportation Research Part B*, 40:265–284, 2006.

[35] D. Monderer and L. Shapley. Fictitious play property for games with identical interests. *Journal of Economic Theory*, 68:258–265, 1996.

[36] J. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, pages 48–49, 1950.

[37] C. H. Papadimitirou and K. Steiglitz. *Combinatorial optimization: Algorithms and complexity*. Dover Publications, Inc, 1998.

[38] S. M. Pollock and R. Smith. A Formalism for Dynamic Programming. Technical report, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, 1985.

[39] W. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality.* Wiley-Interscience, 2007.

[40] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc. New York, NY, USA, 1994.

[41] G. Reinelt. *The Traveling Salesman Problem: Computational Solutions for TSP Applications.* Springer-Verlag, 1994.

[42] J. Robinson. An iterative method of solving a game. *Annals of Mathematics*, pages 296–301, 1951.

[43] R. Rosenthal, J. White, and D. Young. Stochastic Dynamic Location Analysis. *Management Science*, 24:645, 1978.

[44] S. Ross. *Introduction to Probability Models.* Academic Press, 2006.

[45] J. N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

[46] J. Weatherwax. SARSA Implementation in MATLAB. `http://waxworksmath.com/Authors/N_Z/Sutton/sutton.html`.