APPROXIMATION TO THE OPTIMAL STRATEGY IN THE MOZART CAFÉ PROBLEM BY SIMULTANEOUS PERTURBATION STOCHASTIC APPROXIMATION

by
Cheng Peng

A thesis submitted to The Johns Hopkins University in conformity
with the requirements for the degree of Master of Science in Engineering

Baltimore, Maryland
May, 2022

Abstract

The rendezvous search problem is an old and classic problem in operations research. In this problem, two agents with unit speed are placed in some common region and they try to find each other in the least expected time with the assumption that they neither have devices for communication nor necessarily share the same coordinates/directions. In this thesis, we focus on one specific problem in this field, called the "Mozart Café problem," in which two agents search for each other among n discrete locations (cafés). They can go to any café each day and will stay there the whole day waiting for the other to come, and they wish to minimize the expected time to rendezvous. Previous researchers have found and shown the optimal strategies for $n = 2, 3$ cases. In this study, we first present some preliminary work on a variant of the general Mozart Café problem on the $n = 4$ case in which each agent can leave a token in the initial café he visits saying that he would never come back. The optimal strategy on this variant provides a lower bound for the optimal expected rendezvous time in the general $n = 4$ case. Then we propose a novel modelling technique named k-Markovian modelling where the model parameters can be optimized by stochastic optimization algorithms. This also parameterizes this problem. The aims of this work are to provide a parameterization method and demonstrate the potential to approximate the optimal rendezvous search strategy.

Thesis Reader

Professor John Charles Wierman
Professor
Department of Applied Mathematics and Statistics
Johns Hopkins University

Dedicated to myself, the young man who has been learning mathematics with both pleasure and pain.

Acknowledgments

Throughout the writing of this master's thesis, I have received a great deal of support and assistance.

I would first like to thank my thesis advisor, Professor John C. Wierman. I was assigned to Prof. Wierman as his advisee when I came to Johns Hopkins University (JHU), and the topic of this thesis stemmed from the group project in his course EN.553.601 "Introduction to Research" offered in Spring 2021. I have learned a lot of research techniques from Prof. Wierman, which has made me qualified to continue for a doctoral study after graduation from my master's program in JHU. During the process of thesis writing, Prof. Wierman has also provided substantial advice to me.

I would also like to thank my another research advisor in JHU, Prof. James C. Spall. The Simultaneous Perturbation Stochastic Approximation method was invented by Prof. Spall in the early 1990s, and it is the core in the parameter optimization process for our *k*-Markovian model. Meanwhile, the knowledge about Monte Carlo simulation and stochastic search and optimization that was taught in Prof. Spall's courses played a very important role in this thesis.

I would like to acknowledge my two group mates: Leonard Pick and Marcos Valdes who collaborated with me in the course EN.553.601. They contributed significantly for the preliminary work in Chapter 2.

When I wrote this thesis, there was not much trouble in terms of academic writing, and my proficiency in writing has saved a lot of time when making modifications after

Prof. Wierman edits the draft. Actually, my writing ability was trained by several previous research advisors. They are: Prof. Sheung Chi Phillip Yam from The Chinese University of Hong Kong, Prof. Ka Chun Cheung from The University of Hong Kong, and Dr. Antwan Clark from JHU. They used to provide substantial suggestions to me when I wrote manuscripts for research projects. I wish to thank them for their patience and guidance.

In addition, I would like to thank my parents for supporting me for my study in JHU. Without them, I would have no chance to come to this world-class university to study what I love the most. I would also like to thank my partner Mary Zhao for her love and help when I had trouble in doing this thesis project.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Search Theory

The theory of optimal search, a.k.a. search theory, is one of the oldest topics in operations research. During World War II, Bernard Koopman and his colleagues from US Navy initially developed search theory to provide efficient ways to detect submarines [1]. Later, in 1966, the US Navy used search theory to efficiently search for an H-bomb that was lost in the ocean near Palomares, Spain. Apart from military use, the US Coast Guard also used search theory in rescue efforts. In labor economics, Mortensen proposed a dynamic model of job search that based on optimal stopping and studied the problem about which job an unemployed person should accept, and which he/she should reject given that the value of money is constant, and the distribution of alternatives is known and constant [2].

The problem of optimal search consists of a searcher and a target, which can either be an object (immobile) or an agent (mobile). The search theory aims to find an optimal strategy that completes the search process in the least expected time among all possible strategies. Historically, there have been several stages for the formulation of the optimization problem in optimal search [3]. In the early stages, all of the optimization tasks were on the searcher's side while the hidden agent had its own aims.

But later on, after [4], games in which the searcher who tries to minimize the search time and the hider who tries to maximize the search time received much attention.

In 1976, Steve Alpern introduced the term “rendezvous search game” as a special class of optimal search games for the first time in his talk at Vienna. Unlike previously studied optimal search problems, the rendezvous search problem focused on how two or more agents with unit speed can optimize the process by which they meet when they get lost in some common region without the knowledge of each other’s location [5]. The agents may either minimize the expected time to meet or maximize the probability of meeting within a given time span. In this thesis, we define optimality by the minimum expected time.

Generally, two types of strategies are considered for rendezvous search problems: asymmetric and symmetric. In an asymmetric strategy, the agents do not necessarily follow the same strategy while in a symmetric strategy, the agents must independently adopt an identical (randomized) strategy. A common asymmetric approach is the “wait-for-mommy” strategy in which one agent stays still and the other traverses the search space until they meet. In this case, the problem is reduced to the problem in which the searcher looks for an immobile target. However, in reality, since the agents may not be able to communicate and determine who is to stay still, the class of symmetric strategies is more practical. In this thesis, we assume the rendezvous strategy is symmetric.

Alpern listed ten open problems in rendezvous search [5], which are summarized below.

1. **The astronaut problem**. The agents are uniformly and independently placed

on a sphere without a common spherical coordinate system. They move at unit speed and the process terminates if their distance is smaller than some constant.

2. **Rendezvous on a circle**. Both agents are uniformly and independently placed on a circle without any common sense of direction. The search terminates if they meet each other at some point on the circle.

3. **Rendezvous on the infinite line (agent-symmetric form)**. Both agents are randomly placed on a common line and they try to find each other in the shortest time.

4. **Rendezvous on the infinite line (agent-asymmetric form)**. This problem is similar to the previous one, but now the cumulative distribution function $F(d)$ of the initial distance d between the agents is given.

5. **Mozart Café problem**. Two agents want to meet each other among n different Mozart Cafés. On day 1 the n cafes are indistinguishable, so each can do no better than picking one at random. If they don't meet on day 1, then they can choose to return on day 2 to the same cafe, or to go to a random new one. And so on. How can they meet each other in the shortest time?

6. **Mozart Café problem with river**. Suppose there are $2n$ Mozart cafés, with n of them on each side of a river. But the problem has changed, because after not meeting on day 1 for example, an agent has three choices for the next day: the same café, a different one on the same side of the river, a random one on the other side of the river.

7. **Multiple agent rendezvous**. A two-agent rendezvous problem may be modified so that the objective is for n agents to meet at the same location.

8. **Asynchronous rendezvous**. Unlike the previous problems that assume the

agents enter the search region and begin their search simultaneously, this problem allows some coordination.

9. **Rendezvous without proximity**. Unlike the classical form of rendezvous search problem in which the search process terminates once the agents arrive at the same location or come within a specified distance, this type of problem posit a subset R of $Q \times Q$, where Q is the common region, and the process terminates when the locations of the agents form a pair in R.

10. **When to give up: rendezvous with failure**. In real world scenarios, it is common that when the search space is way too large, one or more agent may eventually give up. This type of rendezvous search problem determines the optimal stopping time.

In this thesis, we shall focus on the Mozart Café problem.

1.2 The Mozart Café Problem

We now give a full description of the Mozart Café problem. Two agents agree to meet each other at Mozart Café in Vienna. However, when the day comes, they realize that there are n different Mozart Cafés in Vienna ($n \geq 2$). Suppose they neither have a common labelling/indexing of these n Cafés nor have devices to contact each other. In each unit time step (e.g., one hour, two hours, or one day), each agent will stay in a café and wait. Between adjacent time steps, they can either stay at the same café or move to a different one. How can they meet in the shortest expected time? Since the agents do not share the same labelling of the cafés, this problem is also called "rendezvous search on n discrete locations", and can be simplified as rendezvous search on K_n where K_n is the complete graph with n nodes. There is another version of this problem called "telephone problem", in which two rooms are connected by n pairs of telephones (the telephones have a one-to-one mapping) and there is one person in each room. But they do not know the exact mapping of the telephones. At every time step, each person pick up a phone and say "hello". They wish to minimize the time when they first pick up a pair of phones and hear each other.

As we have introduced in Section 1.1, the strategies of the Mozart Café problem can be roughly divided into two categories: symmetric and asymmetric. For symmetric strategies, the agents should follow the same mixed strategy while for asymmetric strategies, the agents may not follow the same strategy. It has been shown that if the agents use asymmetric strategies, the minimum expected rendezvous time is $\frac{n+1}{2}$ and it is stochastically minimized by a strategy in which one agent stays still and the other traverses all cafés in random order [6].

When it comes to symmetric strategies, things become much harder. In 1990, Anderson and Weber [6] proposed proposed a strategy class called AW(n) strategy where "AW" stands for the authors' last names, and n stands for the number of cafés. The strategy works as follows. In the first time step, because each agent knows nothing about the cafés, they can do no better than randomly picking a café. If they do not meet in this step, then the remaining strategy is defined in terms of blocks with $n-1$ successive steps. For $j = 0, 1, 2, \ldots$, in the block $j(n-1)+2, \ldots, j(n-1)+n$, each agent either stays in the café that he visited at time $j(n-1)+1$ consecutively throughout the whole block with probability p_0 or traverse the remaining $n-1$ cafés that differ from the one he visited in step $j(n-1)+1$ in any order with probability $1-p_0$.

As calculated by Anderson and Weber [6], when $n = 2$, the optimal p_0 is 0.5 and the minimum expected meeting time is 2, which indicates that uniformly random search is indeed the optimal strategy. When $n = 3$, the optimal p_0 is $\frac{1}{3}$, and the optimal expected rendezvous time is $2\frac{2}{3}$. When $n = 4$, the best p_0 is 0.3220 and the expected rendezvous time is 3.5685. The optimality of AW(2) and AW(3) were validated in [7]. However, AW(4) is not optimal since a better strategy which has an expected rendezvous time 0.00015 smaller than that of AW(4) was proposed in [8]. In general, previous works on the Mozart Café problem have only found and proved optimal strategies for $n = 2$ and 3 cases. But for $n > 3$ cases, no optimal solutions have been found. In the following chapters, a new approach will be proposed and discussed, and we shall regard the AW(n) strategy as a benchmark for comparison when applying our method to the Mozart Café problem.

1.3 Preliminary Work and Research Motivation

In this thesis, we refer to the Mozart Café problem described by Steve Alpern as the general Mozart Café problem. Finding the optimal strategy for the general Mozart Café problem with $n \geq 4$ remains an open problem. However, we may obtain the lower bound for the expected rendezvous time if the agents are allowed to do more than simply determining which café to go in the next time step. We regard this type of problem as a sub-problem of the general Mozart Café problem. One interesting sub-problem is the token problem in which each agent can leave a token at the café that he visits saying that he will not come back in the future. We have done some preliminary study on the token approach, which provides a lower bound to the optimal solution of the general Mozart Café problem. More details will be introduced in Chapter 2.

As an extension to our preliminary work, we focus on the general Mozart Café problem in this thesis. There are some reasons why the general problem is difficult to solve. On the one hand, the set of all possible strategies is infinite. On the other hand, there is no previous work that parameterized the rendezvous strategy, and thus there does not exist a generic representation for the set of strategies. Meanwhile, the ways that Weber showed the optimality of AW(2) and AW(3) were not scalable enough to be extended to $n \geq 4$ cases. To solve larger scale Mozart Café problems, one approach is to formulate it as a classical optimization problem which contains a parametric model that specifies a strategy by a finite number of parameters and a cost function that evaluates how good a strategy is. Thus, our first step is to find a parametric model that serves as a generic representation for the strategies. In this way, for any strategy, we are able to estimate its expected rendezvous time, which can be regarded as the cost function. Then, finding the optimal strategy is transformed into an optimization problem where we iteratively update the model parameters so as to converge to an optimal solution.

Before we present our approach, we want to clarify that our approach serves as a way to approximate the optimal rendezvous strategy in the general Mozart Café problem, but we do not prove global optimality. Since in many real-world optimization problems, such as optimizing a deep artificial neural network on a very large image dataset, the strong nonconvexity of the cost function and the huge parameter space make it intractable to find a global optimal solution. Nevertheless, local optimal solutions may already be reasonably good enough. Our ideology for this thesis work is similar, and our aim is to provide a systematic way to find "good enough" rendezvous strategy for large problems. Previous works on the Mozart Café problem were rigorous, but they lack scalability, which results in the infeasibility to be applied to real-world scenarios.

In the following chapters of this thesis, we shall first introduce the preliminary work on the token approach in Chapter 2. Then, we introduce the core of this thesis, the k-Markovian modelling, in Chapter 3. Later, in Chapter 4, we introduce the algorithm simultaneous perturbation stochastic approximation (SPSA), which is used for parameter optimization. Simulation results are provided in Chapter 5. Based on the simulation results, discussions and reflections are provided in Chapter 6. Finally, we state the conclusions in Chapter 6. The code for the simulation study is shown in Appendix I, and rendezvous strategies (i.e., transition probabilities) can be found at the author's GitHub repository [1].

[1]Link to the GitHub repository: `https://github.com/jamespengcheng/k-Markovian`

Chapter 2

Preliminary Work on Token Approach

In this chapter, we introduce our preliminary work on the token approach in the $n = 4$ case, which was completed in the Spring semester of 2021. Although the k-Markovian approach in Chapter 3 almost has nothing to do with the token approach, it was the way of modelling the system by a Markov Chain that inspired us to design the k-Markovian approach. Meanwhile, it was the success in the token approach that stimulated us to continue working on the Mozart Café problem and complete this thesis.

Let us denote the cafés by A, B, C, D. Each agent has one token to leave at the first café that he visits saying that he will not return in the future. We only allow one token for each agent because if each of them can leave more than one, they might not be able to meet in some cases. For example, suppose agent I is at C and he has left tokens at A and B, while agent II is at A and he has left tokens at C and D. Then, they will never meet. Thus, each agent has only one token in hand.

We allow the agents to leave tokens at the starting position because once the agents are aware of each other's initial position, they can do rendezvous search only among the cafés where there are no tokens, which reduces the problem to a general

Mozart Café problem in a smaller number of cafés, and we may apply the results of Anderson and Weber directly. Hence, based on an agent's awareness of the other agent's initial position, he has the following three states.

- 0: The agent has only visited his initial position, knowing nothing about other cafés.
- 1: The agent has visited his initial position plus another café that is NOT the initial position of the other agent.
- 2: The agent has visited his initial position and known the other agent's initial position.

State 1 can be treated as an intermediate state between states 0 and 2 where the agent has excluded one café from being the initial café of the other agent. Without loss of generality, we suppose agent I starts from café A while agent II starts from café B. Moreover, we assume that both agents know that the other will also leave a token in the initial position. Then, an agent can get to know the other's initial position either by visiting it directly or visiting all other cafés. The corresponding visiting history of the three states are listed below.

- 0: The agent has only visited his initial position.
- 1: The agent has visited his initial position and exactly one of C & D.
- 2: The agent has visited his initial position and either B or both C & D.

The state of the system that consists of both agents and four cafés can be defined by an ordered pair giving each agent's state. Since the agents are symmetric, we disregard the order of the numbers in the set. Additionally, we name the state representing rendezvous by $(3, 3)$. The set of system states is shown below.

$$\mathcal{S} = \{\{0,0\}, \{0,1\}, \{0,2\}, \{1,1\}, \{1,2\}, \{2,2\}, \{3,3\}\}.$$

In each time step, the state of the system is updated from the state in the previous time step. Thus, it is reasonable to model the system by a Markov Chain. A Markov Chain is a type of stochastic process in which the system transits from one state to another with respect to certain probability distributions. Meanwhile, the probability for the system to be in each state depends only on its previous state [9]. Depending on the time, Markov Chains can be roughly divided into two categories: discrete-time Markov Chains and continuous-time Markov Chains. Our solution utilizes the discrete case. In a discrete-time Markov Chain, suppose the time steps are $0, 1, 2, \ldots, n$, for any $n \in \mathbb{Z}^+$, and the possible states of the system are $i_0, i_1, \ldots, i_n$. We then have Equation 2.1.

$$\mathbb{P}(X_n = i_n | X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-1}) = \mathbb{P}(X_n = i_n | X_{n-1} = i_{n-1}). \tag{2.1}$$

At any time t, the transition probability matrix P_t for the Markov Chain is an $|\mathcal{S}| \times |\mathcal{S}|$ matrix whose entries satisfy

$$(P_t)_{i,j} = \mathbb{P}(X_{t+1} = j | X_t = i) \quad \forall (i, j) \in \mathcal{S} \times \mathcal{S}.$$

In other words, each row of P_t is the probability distribution over all states in the next time step given that the current state is the one represented by this row. Our solutions incorporate this one-step-look-ahead property. We derive systems of linear equations about the expected rendezvous time on each state of the system with respect to the rows of the transition probability matrix. In the following sections in this chapter, we assume agent I starts from A and agent II starts from B. Suppose the expected rendezvous time given that they start from A and B respectively is E. Then we can calculate the overall expected rendezvous time by Equation 2.2, where the $\frac{1}{4}$ represents the case that they meet on the first time step.

$$E_{\text{overall}} = \frac{1}{4} + \frac{3}{4}E. \tag{2.2}$$

In the following sections, we shall make theoretical analysis first. Then we present the results from numerical simulation.

2.1 Theoretical Analysis

As is described by Steve Alpern, in every time step, an agent either stays still or goes to another café. Furthermore, in the token problem, an agent will neither go back to his initial position (if he has already left) nor go to the other agent's initial position (if he knows it). We refer to these cafés as "feasible cafés". For example, if agent I has left café A and is now at café C without knowing agent II's initial position, the feasible cafés are B, C, and D. Next, if he goes to D, then he realizes that agent II started from B (since there is no token in C or D), his feasible cafés become C and D. We assume that if an agent decides to go to a feasible café that is different from the current one, he will uniformly randomly choose one to visit. This assumption makes sense because there is no difference among those feasible cafés that are different from the current one. Therefore, we set the waiting probabilities in each state of an agent as a parameter, and the goal is to find the set of parameters that minimizes the expected rendezvous time. The definition of parameters is shown below.

- θ_0: $\mathbb{P}$(Staying Still | Visiting History at state 0).
- θ_1: $\mathbb{P}$(Staying Still | Visiting History at state 1).
- θ_2: $\mathbb{P}$(Staying Still | Visiting History at state 2).

One notable thing is that we allow the agents to stay still consecutively. If we do not allow this, the state definition using the visiting history no longer possesses the Markov property, and we would need to further consider whether an agent has stayed still in the previous time step or not, which largely increases the complexity of the formulation. We have the following equation.

$$
\begin{aligned}
E_{0,0} =& 2\left(\frac{1-\theta_0}{3}\right)^2 + 2\theta_0\frac{1-\theta_0}{3} + (1+E_{0,0})\theta_0^2 + (1+E_{1,1})\cdot 2\left(\frac{1-\theta_0}{3}\right)^2 + (1+E_{1,0}) \\
&\cdot 2\theta_0\frac{2(1-\theta_0)}{3} + (1+E_{1,2})\cdot 2\frac{1-\theta_0}{3}\cdot\frac{2(1-\theta_0)}{3} + (1+E_{2,2})\left(\frac{1-\theta_0}{3}\right)^2.
\end{aligned} \tag{2.3}
$$

In this equation, $2\left(\frac{1-\theta_0}{3}\right)^2$ represents meeting at C or D. The term $2\theta_0\frac{1-\theta_0}{3}$ represents meeting at A or B. The term $(1+E_{0,0})\theta_0^2$ represents both of them waiting and the state remaining $0,0$. The term $(1+E_{1,1})\cdot 2\left(\frac{1-\theta_0}{3}\right)^2$ represents one going to C, the other going to D, and the state becoming $\{1,1\}$. The term $(1+E_{1,0})\cdot 2\theta_0\frac{2(1-\theta_0)}{3}$ represents one staying still while the other goes to C or D and the state becoming $1,0$. The term $(1+E_{1,2})\cdot 2\frac{1-\theta_0}{3}\cdot\frac{2(1-\theta_0)}{3}$ represents one going to C or D while the other visits his friend's initial position, and the state becomes $\{1,2\}$. $(1+E_{2,2})(\frac{1-\theta_0}{3})^2$ represents both of them visit each other's initial position and the state becoming $\{2,2\}$.

$$
\begin{aligned}
E_{1,0} =& \theta_1\frac{1-\theta_0}{3} + \frac{1-\theta_0}{3}\cdot\frac{1-\theta_1}{2} + \theta_0\frac{1-\theta_1}{2} + \theta_0\theta_1(1+E_{1,0}) + \theta_1\frac{1-\theta_0}{3}(1+E_{1,1}) \\
&+ \left(\theta_1\frac{1-\theta_0}{3} + \frac{2(1-\theta_0)}{3}\cdot\frac{1-\theta_1}{2} + \frac{1-\theta_0}{3}\cdot\frac{1-\theta_1}{2}\right)(1+E_{1,2}) \\
&+ \frac{1-\theta_0}{3}\cdot(1-\theta_1)(1+E_{2,2}) + \theta_0\frac{1-\theta_1}{2}(1+E_{0,2}).
\end{aligned} \tag{2.4}
$$

Assume agent I stays still while agent II visits C. The term $\theta_1\frac{1-\theta_0}{3}$ represents meeting at C. The term $\frac{1-\theta_0}{3}\cdot\frac{1-\theta_1}{2}$ represents meeting at D. The term $\theta_0\frac{1-\theta_1}{2}$ represents meeting at A. The term $\theta_0\theta_1(1+E_{1,0})$ represents both staying still and the state remaining as $\{1,0\}$. The term $\theta_1\frac{1-\theta_0}{3}(1+E_{1,1})$ represents II staying still while I goes to D and the state becoming $\{1,1\}$. The term $\left(\theta_1\frac{1-\theta_0}{3} + \frac{2(1-\theta_0)}{3}\cdot\frac{1-\theta_1}{2} + \frac{1-\theta_0}{3}\cdot\frac{1-\theta_1}{2}\right)(1+E_{1,2})$ includes these cases:

- I goes to B, II stays still, with probability $\theta_1\frac{1-\theta_0}{3}$.
- I goes to either C or D, II goes to A, with probability $\frac{2(1-\theta_0)}{3}\cdot\frac{1-\theta_1}{2}$.

- I goes to C, II goes to D, with probability $\frac{1-\theta_0}{3} \cdot \frac{1-\theta_1}{2}$.

In all cases, the state becomes $\{1, 2\}$. The term $\frac{1-\theta_0}{3} \cdot (1-\theta_1)(1+E_{2,2})$ represents I going to B and II going to A or D. The term $\theta_0 \frac{1-\theta_1}{2}(1+E_{0,2})$ represents I staying still, II going to D, and the state becoming $\{0, 2\}$.

$$E_{1,1} = \theta_1(1-\theta_1) + (1+E_{1,1})\theta_1^2 + (1+E_{1,2})\theta_1(1-\theta_1) + (1+E_{2,2})(1+\theta_1^2 - 2\theta_1). \quad (2.5)$$

Assume agent I visits C and agent II visits D. The term $2\theta_1 \frac{1-\theta_1}{2} = \theta_1(1-\theta_1)$ represents meeting at C or D. The term $(1+E_{1,1})\theta_1^2$ represents both staying still and the state remaining as $\{1, 1\}$. The term $(1+E_{1,2})\theta_1(1-\theta_1)$ represents I staying still while II visits A or I visits B while II stays still. The term $(1+E_{2,2})(1+\theta_1^2-2\theta_1)$ includes these cases:

- I goes to D, II goes to C, with probability $\left(\frac{1-\theta_1}{2}\right)^2$.
- I goes to B, II goes to A, with probability $\left(\frac{1-\theta_1}{2}\right)^2$.
- I goes to D, II goes to A, with probability $\left(\frac{1-\theta_1}{2}\right)^2$.
- I goes to B, II goes to C, with probability $\left(\frac{1-\theta_1}{2}\right)^2$.

In these cases, the state becomes $\{2, 2\}$.

$$\begin{aligned} E_{0,2} = {} & \theta_2 \frac{1-\theta_0}{3} + (1-\theta_2)\frac{1-\theta_0}{3} + \theta_0(1+E_{0,2}) \\ & + \left(\theta_2 \frac{1-\theta_0}{3} + (1-\theta_2)\frac{1-\theta_0}{3}\right)(1+E_{1,2}) + \frac{1-\theta_0}{3}(1+E_{2,2}). \end{aligned} \quad (2.6)$$

Assume agent I stays still, while agent II has visited both C and D, and he is now at C. The term $\theta_2 \frac{1-\theta_0}{3}$ represents meeting at C. The term $(1-\theta_2)\frac{1-\theta_0}{3}$ represents meeting at D. The term $\theta_0(1+E_{0,2})$ represents I staying still, while II visits any one of C or D. The term $\left(\theta_2 \frac{1-\theta_0}{3} + (1-\theta_2)\frac{1-\theta_0}{3}\right)(1+E_{1,2})$ includes these cases:

- I visits D, II stays still, with probability $\theta_2 \frac{1-\theta_0}{3}$.
- I visits C, II visits D, with probability $(1-\theta_2)\frac{1-\theta_0}{3}$.

In these cases, the states becomes $\{1, 2\}$.

$$\begin{aligned} E_{1,2} = {} & \theta_1(1-\theta_2) + \frac{1-\theta_1}{2}\theta_2 + \theta_1\theta_2(1+E_{1,2}) \\ & + \left(\frac{1-\theta_1}{2} + \frac{1-\theta_1}{2}(1-\theta_2)\right)(1+E_{2,2}). \end{aligned} \tag{2.7}$$

Assume agent I has only visited A and C, and he is at C, agent II has visited A or both C and D, and he is at D. The term $\theta_1(1-\theta_2)$ represents meeting at C. The term $\frac{1-\theta_1}{2}\theta_2$ represents meeting at D. The term $\theta_1\theta_2(1+E_{1,2})$ represents both staying still, and the states remaining as $1, 2$. The term $\left(\frac{1-\theta_1}{2} + \frac{1-\theta_1}{2}(1-\theta_2)\right)(1+E_{2,2})$ include the following cases:

- I goes to B, II visits any of C or D, with probability $\frac{1-\theta_1}{2}$.
- I goes to D, II goes to C, with probability $\frac{1-\theta_1}{2}(1-\theta_2)$.

In these cases, the state becomes $\{2, 2\}$.

$$E_{2,2} = 2\theta_2(1-\theta_2) + (\theta_2^2 + (1-\theta_2)^2)(1+E_{2,2}). \tag{2.8}$$

The term $2\theta_2(1-\theta_2)$ represents meeting at C or D. The term $(\theta_2^2 + (1-\theta_2)^2)(1+E_{2,2})$ represents not meeting. Note that $E_{2,2}$ is just the 2-café case whose optimal meeting time is 2 with the optimal strategy being random strategy. We solve Equation 2.8 by setting $E_{2,2} = 2$, and find that $\theta_2 = \frac{1}{2}$, which is in accordance with the random strategy.

Once an agent reaches state 2, the rendezvous problem for him has been reduced to the general Mozart Café problem with two cafés. According to AW(2), the optimal strategy is just random search. So, we replace θ_2 by $\frac{1}{2}$, and from Equations 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, we get the following solutions.

- $E_{2,2} = 2$.
- $E_{1,2} = \frac{5-3\theta_1}{2-\theta_1}$.
- $E_{0,2} = \frac{5-2\theta_0}{3(1-\theta_0)} + \frac{5-3\theta_1}{3(2-\theta_1)}$.
- $E_{1,1} = \frac{\theta_1^3-6\theta_1+6}{(2-\theta_1)(1-\theta_1^2)}$.
- $E_{1,0} = \frac{2\theta_0\theta_1-2\theta_1-2\theta_0+5}{3} + \frac{\theta_1(1-\theta_0)}{3}E_{1,1} + \frac{\theta_1\theta_0-\theta_1-3\theta_0+3}{6}E_{1,2} + \frac{\theta_0(1-\theta_1)}{2}E_{0,2}$.
- $E_{0,0} = \frac{2\theta_0^2-4\theta_0+11}{9(1-\theta_0)^2} + \frac{2}{9}E_{1,1} + \frac{4\theta_0}{3(1+\theta_0)}E_{1,0} + \frac{4}{9}E_{1,2}$.

Unfortunately, obtaining the analytical solutions to this system of equations is intractable. We proceed in two ways. For the first, we turn to numerical optimizers. For the second, we design a Markov Chain to conduct a simulation. Since there are two parameters now, we use grid search to check the convergence of the Markov Chain when adjusting the value of θ_0 and θ_1.

2.2 Numerical Simulation

In this section, we conduct numerical simulations for the system modeled in the previous section. We design a Markov Chain in which the state representing rendezvous is an absorbing state. By the initial distribution and transition probability, we can compute the time required for convergence to the absorbing state. Meanwhile, we also provide a result given by an encapsulated optimizer in Python.

2.2.1 Markov Chain simulation

There are 7 states in the Markov Chain.

- $\{0, 0\}$: Given that they have not met, their visiting histories are both at state 0.
- $\{0, 1\}$: Given that they have not met, their visiting histories are at 0 and 1, respectively.
- $\{1, 1\}$: Given that they have not met, their visiting histories are both at 1.
- $\{0, 2\}$: Given that they have not met, their visiting histories are at 0 and 2, respectively.
- $\{1, 2\}$: Given that they have not met, their visiting histories are at 1 and 2, respectively.
- $\{2, 2\}$: Given that they have not met, their visiting histories are both at 2.
- $\{3, 3\}$: They meet each other.

Obviously, at the end of step 1, they must be in state $\{0, 0\}$. After step 1, the transition probability matrix is shown below.

$$P = \begin{bmatrix} \theta_0^2 & \frac{4\theta_0(1-\theta_0)}{3} & 2(\frac{1-\theta_0}{3})^2 & 0 & \frac{4(1-\theta_0)^2}{9} & (\frac{1-\theta_0}{3})^2 & \frac{-4\theta_0^2+2\theta_0+2}{9} \\ 0 & \theta_0\theta_1 & \theta_1\frac{1-\theta_0}{3} & \theta_0\frac{1-\theta_1}{2} & \frac{\theta_0\theta_1-\theta_1-3\theta_0+3}{6} & \frac{1-\theta_0}{3}\cdot(1-\theta_1) & \frac{-4\theta_1\theta_0+2\theta_0+\theta_1+1}{6} \\ 0 & 0 & \theta_1^2 & 0 & \theta_1(1-\theta_1) & 1+\theta_1^2-2\theta_1 & \theta_1(1-\theta_1) \\ 0 & 0 & 0 & \theta_0 & \frac{1-\theta_0}{3} & \frac{1-\theta_0}{3} & \frac{1-\theta_0}{3} \\ 0 & 0 & 0 & 0 & \frac{\theta_1}{2} & \frac{3(1-\theta_1)}{4} & \frac{1+\theta_1}{4} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

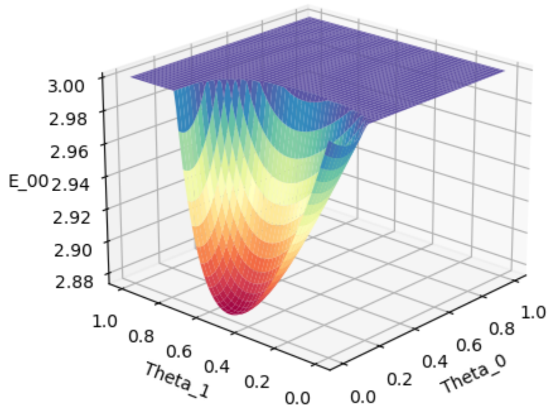


Figure 2-1. Markov Chain simulation

When running the simulation program, we use initial distribution $[1, 0, 0, 0, 0, 0, 0]$, and iteratively multiply matrix P. Since the state $\{3, 3\}$ is the absorbing state, the value in the last entry of the probability distribution vector converges to 1. As the numerical error can never be fully eliminated, we terminate the iteration when the value in the last entry of the probability distribution vector is no less than 0.999. To search for the optimal parameter $(\theta_0^\star, \theta_1^\star)$, we use grid search. Since neither θ_0 nor θ_1 can be 0 or 1 (otherwise, there will be an issue of zero-division), we divide the interval $[0.001, 0.999]$ into 100 points with common difference, and we take all combinations of θ_0 and θ_1 in the grid. In order to make the simulation more efficient, we terminate the iteration when the expected rendezvous time is larger than 3. We plot the relationship between $E_{0,0}$ and (θ_0, θ_1), which is shown in Figure 2-1.

Since we have replaced the estimated rendezvous time by 3 if it takes more than 3 steps to rendezvous, Figure 2-1 illustrates that there exists a neighborhood around the optimal solution (i.e., the bowl-shaped area in the figure) that is convex, while for regions outside of the neighborhood, the expected rendezvous time is always no less than 3. The smallest $E_{0,0}$ that we have obtained is 2.8759 when $\theta_0 = 0.0212$ and

Min $E_{0,0}$	θ_0	θ_1
2.87598723	0.02020691	0.4891782
2.87760838	0.05824739	0.5026782
2.8827747	0.08903752	0.47023937
2.88501544	0.10113257	0.47474662
2.8851602	0.10290303	0.51635975

Table 2-I. Results from COBYLA

$\theta_1 = 0.5050$.

2.2.2 Numerical Optimization

Apart from the Markov Chain simulation, we also use a numerical optimization algorithm named Constrained Optimization by Linear Approximation (COBYLA) to find the optimal value [10]. We repeated the experiment 50 times and recorded the optimal $E_{0,0}$, θ_0 and θ_1. The top 5 results are shown in Table 2-I. The optimal $E_{0,0}$ occurs at 2.87 when $\theta_0 \approx 0.02$ and $\theta_1 \approx 0.5$. This is in accordance with what we got in Section 2.2.1.

Thus, the minimum value of the expected overall meeting time can be calculated using Equation 2.9.

$$\min\left(E_{\text{overall}}\right) = \frac{1}{4} + \frac{3}{4} \cdot 2.87584 \approx 2.407. \tag{2.9}$$

As the result is much smaller than the expected time in AW(4), which is 3.5685 when the waiting probability is 0.3220, the result given by the token approach serves as a lower bound for the general Mozart Café problem on 4 cafés.

Chapter 3

k-Markovian Modelling

In this chapter, we shall introduce our k-Markovian modelling technique for the general Mozart Café problem. Some assumptions about the model and the agents will be introduced in Section 3.1. Then, in Section 3.2, we shall first introduce the k-Markovian modelling technique, and then introduce the way we parameterize the model. Important notations have been summarized in Table 3-I. We used Python to implement the model in this thesis project, but it can also be implemented in other scientific programming languages as well. Details about the implementation will also be discussed.

3.1 Assumptions

A search strategy for an agent in a rendezvous search problem can be treated as a "black box" that takes all the information in hand as the input, and then outputs a probability distribution over all the places that are one-step-accessible to the agent (i.e., can be accessed in the next time step). In the general Mozart Café problem, as the agents can neither contact each other nor leave any token in a café that he visits, all the information in hand is their own visiting history. Thus, our "black box" should be able to process an agent's visiting history and output a probability distribution over all the n cafés.

Since the “black box” processes the visiting history which consists of a sequence of cafés that the agent has visited in previous steps, the way of labelling the cafés is nontrivial. As is illustrated in the problem statement, the agents did not know that there are n distinct Mozart cafés in Vienna before their arrival. We can assume that they do not necessarily have a common labelling to the cafés. Further, as an agent only has knowledge of the cafés that he has visited, we assume that he labels a café if and only if he has visited it. In this way, a reasonable assumption to an agent's labelling is that he labels a café as number i if this is the i-th distinct café that he has visited where $i \in \{1, 2, \ldots, n\}$. Then, if an agent has visited m distinct cafés ($m \leq n$), his visiting history is a sequence in which each element is an integer between 1 and m.

Next, as an agent knows nothing about the unvisited cafés, he should be indifferent between them in choosing which one to visit. In every step, the probabilities of going to unvisited cafés should be equal. With this assumption, our modelling in this chapter is consistent with the preliminary work in Chapter 2 because we also evenly divided the probability to unvisited cafés in that chapter.

We have made assumptions for

- the input-output format of the model,
- the agents' labelling for the cafés,
- the probability of going to unvisited cafés.

Next, let us introduce the details about the k-Markovian modelling.

Table 3-I. Nomenclature

n	The number of cafés.
k	Maximum length of visiting history to be passed into the model.
$\mathbf{\Theta}$	The whole set of model parameters.
Ω	The parameter space.
$\boldsymbol{\theta}^{m,t}$	Subset of parameters with m visited cafés and t steps of visiting history.
$\mathbf{h}_{m,t}$	A visiting history with m visited cafés and length t ($t \leq k$).
$\boldsymbol{\theta}^{m,t}_{\mathbf{h}_{m,t}}$	Parameters extracted from $\boldsymbol{\theta}^{m,t}$ by $\mathbf{h}_{m,t}$.
x	The function that describes the position of an agent ($x : \mathbb{N} \to \{1, 2, \ldots, n\}$).
$T_{\mathbf{\Theta}}$	The rendezvous time when the model parameter is $\mathbf{\Theta}$.

3.2 k-Markovian modelling

3.2.1 Motivation

As we have stated before, our “black box”, or in other words, the model that represents a rendezvous search strategy, should be able to process the visiting history of an agent and output a probability distribution for the next step. However, as the length of a visiting history is not bounded from above, we set a limit $k \in \mathbb{N}$ as the maximum length of visiting history that we pass into the model so that the number of model parameters is finite. That is to say, when the length of visiting history of an agent is beyond k, we only pass the most recent k steps into the model. This is the reason why there is a k in the name of the modelling technique. Furthermore, if we regard the cafés that an agent visits in k consecutive steps as his “state”, then the next state, which consists of the most recent $k-1$ steps and the next step, only depends on the most recent k steps, i.e., the previous state. Hence, an agent’s state transition is modelled by a Markov Chain, which is the reason why we name the technique k-Markovian modelling.

We aim to devise a parametric model that is able to compute the probability distribution for the next step based on at most k previous steps. In this way, for any set of values of model parameters, we can conduct Monte Carlo simulation to estimate the expected rendezvous time, which makes it possible for us to leverage optimization algorithms to search for the model parameter that has the least expected rendezvous time.

3.2.2 Parameterization

In our modelling, k will be set as a hyper-parameter. At any time t, if $t \leq k$, the transition probability depends on the entire visiting history. Otherwise, the transition probability depends on the most recent k steps. We expect that when k is sufficiently large, i.e., much larger than the optimal expected rendezvous time, we should be able to approximate the optimal strategy by the k-Markovian model. The reason why we set k as a finite hyper-parameter is for the purpose of parameterization.

Our design of the model was inspired by the binary tree model in option pricing that was first introduced by William F. Sharpe in 1978 [11]. Sharpe used a binary tree to model the price change of the underlying asset in discrete time. In sufficiently small time periods, the price of the underlying asset may go up or down with certain probabilities. Then in the whole time horizon, the binary tree is able to capture any possible "path" of the asset's price. Inspired by Sharpe's way of capturing all possible paths, we want our model to capture all possible visiting histories. As we have introduced in Section 3.1, an agent labels a café by $i \in \mathbb{N}$ if this is the i-th distinct café that he has visited. The labelling for cafés is monotonically increasing. If at time step t, he has visited m distinct cafés ($m \leq n$), then there can be m^t possibilities in terms of his visiting history, and the k-Markovian model needs to be able to cover $m^{\min(t,k)}$ possibilities. Furthermore, for the next step, we need m probability values that describe the probabilities of going to visited cafés and 1 additional probability value that describes the probability of going to unvisited cafés (when $m < n$). (Since we have assumed that the probabilities of going to unvisited cafés are equal.) Thus, we need $m^{\min(t,k)} \times \min(m+1,n)$ parameters to represent all possibilities of visiting history and the next step. In this way, given a visiting history of length t, we extract $\min(m+1,n)$ parameters from the parameter set using the $\min(t,k)$ cafés from the visiting history as indices. The $\min(m+1,n)$ parameters that we have extracted will be used to calculate the transition probability. We shall discuss the probability calculation in Section 3.2.2.1. Moreover, for different values of $t \in \{1,2,\ldots,k\}$, we need separate subsets of model parameters to compute the transition probability. We design separate subsets of model parameters because we want each parameter in the whole parameter set to be only used to calculate the transition probability for one possible visiting history. Although this exhaustive designing method makes the number of parameters increase exponentially, due to the exclusiveness of the usage of each parameter, whenever we change the value of a parameter, it only affects the transition probability of one visiting history, which increases the k-Markovian model's power of representation. We shall elaborate on the model's power of representation in Section 3.2.2.1. The number of parameters corresponding to the number of visited cafés and the length of visiting history are summarized in Table 3-II. The first column stands for the number of cafés that an agent has already explored. From the second to the last column, each column represents a length of the visiting history ranging from 1 to k. Each cell in the table represents the number of parameters in this subset (when fixing the number of cafés explored and the length of visiting history). Please note that when m cafés have been visited, the length of visiting history must be at least m. So, the table is upper triangular.

Next, we shall elaborate on the way we calculate the transition probability based on the set of parameters described in Table 3-II.

Table 3-II. The number of parameters for k-Markovian modelling

Cafés visited	1-step	2-step	3-step	$\dots$	k-step
1	$1^1 \times 2$	$1^2 \times 2$	$1^3 \times 2$	$\dots$	$1^k \times 2$
2	–	$2^2 \times 3$	$2^3 \times 3$	$\dots$	$2^k \times 3$
3	–	–	$3^3 \times 4$	$\dots$	$2^k \times 3$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
n	–	–	–	$\dots$	$n^k \times n$

3.2.2.1 Probability Calculation

Let us denote the whole set of model parameters by $\mathbf{\Theta}$. Further, as we have partitioned the set of parameters into disjoint subsets by the number of visited cafés and the length of visiting history, we denote the subset of parameters corresponding to m ($m \leq n$) visited cafés and t ($t \leq k$) steps of visiting history by $\boldsymbol{\theta}^{m,t} \in \mathbb{R}^{m^k \times \min(m+1,n)}$. Later, given the visiting history $\mathbf{h}_{m,t}$ with m visited cafés and length t (for simplicity of notation, we assume $t \leq k$), we extract parameters from $\boldsymbol{\theta}^{m,t}$ by the k indices of cafés in the visiting history, which results in a vector $\boldsymbol{\theta}^{m,t}_{\mathbf{h}_{m,t}} \in \mathbb{R}^{\min(m+1,n)}$. Now, we want to store and retrieve the probability distribution for the next step in this vector $\boldsymbol{\theta}^{m,t}_{\mathbf{h}_{m,t}}$. One way to do this is to store the probability values directly. However, the major issue lies in the constraints for the parameters. Since a probability can only be a real number in $[0,1]$, and the summation of the probability values should be 1, this design of parameters makes the parameter optimization become a constrained optimization problem, which is not easy to solve. Thus, we need a function that maps any $\boldsymbol{\theta}^{m,t}_{\mathbf{h}_{m,t}} \in \mathbb{R}^{\min(m+1,n)}$ to a discrete probability distribution with $\min(m+1,n)$ outcomes. One famous function with such property, which is also widely used in artificial neural networks, is the softmax function

$$\sigma(\boldsymbol{\theta}^{m,t}_{\mathbf{h}_{m,t},i}) = \frac{\exp(\boldsymbol{\theta}^{m,t}_{\mathbf{h}_{m,t},i})}{\sum_{j=1}^{\min(m+1,n)} \exp(\boldsymbol{\theta}^{m,t}_{\mathbf{h}_{m,t},j})} \text{ for } i = 1, \dots, \min(m+1,n),$$

where $\boldsymbol{\theta}^{m,t}_{\mathbf{h}_{m,t},i} \in \mathbb{R}$ is the i-th element of $\boldsymbol{\theta}^{m,t}_{\mathbf{h}_{m,t}}$.

With the help of the softmax function, the parameters' values in $\boldsymbol{\Theta}$ are not necessarily the actual transition probability. However, as the exponential function grows sharply, even a small difference (say, 1 or 2) between the parameters results in a huge difference in the resulting probability mass. To tackle this issue, we preprocess the parameters by the hyperbolic tangent function before passing them into the softmax function when we compute the probability distribution. The hyperbolic tangent function is

$$\tanh(z) \equiv \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \; \forall\, z \in \mathbb{R}.$$

We choose the hyperbolic tangent function to preprocess the parameter because it is a smooth and nondecreasing function that scales any real number to a value in $(0, 1)$. By preprocessing the parameters with the hyperbolic tangent function and then passing them into the softmax function, when the value of one parameter is changed, the change of the resulting probability mass is much milder than directly using the softmax function. In other words, the transition probabilities are much less sensitive to the model parameters after preprocessing. Let us denote the position of an agent by a function $x : \mathbb{N} \to \{1, 2, \ldots, n\}$. Given the visiting history $\mathbf{h}_{m,t}$ and the corresponding parameters $\boldsymbol{\theta}^{m,t}_{\mathbf{h}_{m,t}}$, the transition probabilities satisfy Equation 3.1.

$$\mathbb{P}(x(t+1) = i | \mathbf{h}_{m,t}) = \frac{\exp(\tanh(\boldsymbol{\theta}^{m,t}_{\mathbf{h}_{m,t},i}))}{\sum_{j=1}^{n} \exp(\tanh(\boldsymbol{\theta}^{m,t}_{\mathbf{h}_{m,t},j}))}. \tag{3.1}$$

If $m + 1 \geq n$, then i represents any café's index. Otherwise, i either represents the m cafés that have been visited or the $n - m$ unvisited cafés entirely. In this way, we have transformed a constrained optimization problem into an unconstrained optimization problem. Next, we introduce our implementation for the k-Markovian model.

3.2.3 Implementation of k-Markovian Model

We implemented the k-Markovian model in Python. The model parameters are stored in a nested list in which each sublist stores the subsets of parameters with the same number of visited cafés. So, there are n sublists in total. Since $\boldsymbol{\theta}^{m,t} \in \mathbb{R}^{m^t \times \min(m+1,n)}$ ($m \leq n$ and $t \leq k$), we use a Numpy array with shape $(m, m, \ldots, m, \min(m+1, n))$ (t m's before the last entry) to store the subset of parameters $\boldsymbol{\theta}^{m,t}$. Then, there should be $\max(k - m + 1, 1)$ Numpy arrays stored in the m-th sublist.

For the model parameter set $\boldsymbol{\Theta}$, we denote the rendezvous time by a random variable $T_{\boldsymbol{\Theta}}$, and we need to estimate its expectation $\mathbb{E}(T_{\boldsymbol{\Theta}})$. However, because we need to make an infinite summation and the complexity for estimating $\mathbb{P}(T_{\boldsymbol{\Theta}} = t)$ grows exponentially with t, it is intractable to estimate $\mathbb{E}(T_{\boldsymbol{\Theta}})$ by

$$\mathbb{E}(T_{\boldsymbol{\Theta}}) = \sum_{t=1}^{+\infty} \mathbb{P}(T_{\boldsymbol{\Theta}} = t) \cdot t. \tag{3.2}$$

Thus, we should turn to Monte Carlo simulation in practice. In each sample run, we randomly initialize the starting cafés of the two agents and keep records for their labellings by two Python dictionaries whose keys are the agent's own indices and the values are the cafés. In each time step, we extract parameters from the parameter set and compute transition probabilities by 3.1. Once they rendezvous, the sample run stops. To make an accurate estimation, between every 100 sample runs, we compute the 95% confidence interval of $\mathbb{E}(T_{\boldsymbol{\Theta}})$ by

$$\bar{T}_{\boldsymbol{\Theta}} \pm t_{0.025,N-1} \cdot \frac{s_N}{\sqrt{N}}, \tag{3.3}$$

where $\bar{T}_{\boldsymbol{\Theta}} = \sum_{i=1}^{N} T_{\boldsymbol{\Theta}^{(i)}}$ is the mean rendezvous time in N runs, s_N is the sample standard deviation of the N rendezvous times, and $t_{0.025,N-1}$ is the critical value of the two-tailed Student's t-distribution with $N - 1$ degrees of freedom at significance level 0.05. In our simulation, we stop the estimation once the width of confidence interval is below some prespecified threshold.

Now, we have introduced our parameterization and important details in the model implementation. In the next chapter, we introduce the optimization algorithm that we adopt.

Chapter 4

Optimization by SPSA

Since we estimate $\mathbb{E}(T_{\boldsymbol{\Theta}})$ by Monte Carlo simulation, the relationship between the real value and the estimated value of rendezvous time can be expressed by

$$\mathbb{E}(T_{\boldsymbol{\Theta}}) = \hat{\mathbb{E}}(T_{\boldsymbol{\Theta}}) + \epsilon_{\boldsymbol{\Theta}}, \tag{4.1}$$

where $\epsilon_{\boldsymbol{\Theta}}$ is interpreted as the noise term due to Monte Carlo error, and $\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}})$ is an estimate of the expected rendezvous time from Monte Carlo simulation. As we have introduced in 3.3, the estimated rendezvous time is given by $\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}}) = \bar{T}_{\boldsymbol{\Theta}}$. As we adjust the number of repetitions (i.e., N) to reduce the width of the 95% confidence interval for $\mathbb{E}(T_{\boldsymbol{\Theta}})$ to be less than 0.5, in 95% of cases, the true expected rendezvous time will fall into the interval constructed by 3.3. Let us denote the parameter space by Ω. As our objective is to approximate $\boldsymbol{\Theta}^\star$ that satisfies

$$\boldsymbol{\Theta}^\star = \arg\min_{\boldsymbol{\Theta}\in\Omega} \mathbb{E}(T_{\boldsymbol{\Theta}}),$$

the objective function $\rho : \mathbb{R} \to \mathbb{R}$ that is derived from $\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}})$ should satisfy

$$\rho(\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}_1})) \leq \rho(\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}_2})) \Longleftrightarrow \hat{\mathbb{E}}(T_{\boldsymbol{\Theta}_1}) \leq \hat{\mathbb{E}}(T_{\boldsymbol{\Theta}_2}) \text{ for any } \boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2 \in \Omega. \tag{4.2}$$

In other words, the larger the value of $\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}})$, the larger the value of $\rho(\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}}))$. For example, we can let $\rho(\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}})) = \hat{\mathbb{E}}(T_{\boldsymbol{\Theta}})$ directly, or let $\rho(\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}})) = \exp(\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}}))$ so that the loss value increases faster when $\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}})$ gets larger.

Next, we need an optimization algorithm to approximate $\mathbf{\Theta}^\star$. Since ρ is estimated from $\hat{\mathbb{E}}(T_{\mathbf{\Theta}})$, ρ cannot be written in any closed form. The optimization algorithm should not be based on the first-order gradient because $\nabla_{\mathbf{\Theta}}\rho$ is not easily computable. Meanwhile, as ρ contains noise (due to the term $\epsilon_{\mathbf{\Theta}}$), the optimization algorithm should be able to handle noisy measurements of the loss function. Based on these two reasons, we adopt the method developed by Spall [12] in 1992 named simultaneous perturbation stochastic approximation (SPSA). In the next section, we provide a brief introduction to SPSA.

4.1 Introduction to SPSA

Multivariate optimization plays a critical role in the analysis of many engineering systems. In real-world optimization scenarios, obtaining the analytical solution is almost impossible due to non-convexity (or even stochasticity) of the objective function and high dimensionality of the parameter space. Thus, it is necessary to turn to optimization algorithms that iteratively search for the optimal solution. Simultaneous perturbation stochastic approximation (SPSA) is a randomized algorithm that optimizes systems with multiple unknown parameters in an iterative manner. It has been widely used in statistical parameter estimation, feedback control, signal processing, and simulation-based optimization [13]. The ideology of SPSA stemmed from a classical optimization method, gradient descent. However, they are used in different scenarios.

4.1.1 From Gradient Descent to Stochastic Approximation

The mathematical representation of most optimization problems is to minimize some objective function $L : \Omega \to \mathbb{R}$ with respect to a vector of adjustable model parameters $\boldsymbol{\theta}$ where $\boldsymbol{\theta} \in \Omega$. The model parameters start from some initial guess, and after a number of iterations where in each iteration we change the value of the parameters by some rules, the parameters are expected to reach some value that offers an improvement to the objective function. When $L \in C^1$, gradient descent (also often called steepest descent) is always adopted to update the parameters in each iteration. For the multivariate function L that is defined and differentiable in a neighborhood of a point $\boldsymbol{\theta}_t$ ($t \in \mathbb{N}$), it decreases fastest if one goes from $\boldsymbol{\theta}_t$ in the direction of the negative gradient of L, i.e., $-\nabla L(\boldsymbol{\theta}_t)$. It follows that there exists $r > 0$ such that for any $0 < \gamma_t \leq r$, $L(\boldsymbol{\theta}_{t+1}) \leq L(\boldsymbol{\theta}_t)$ where

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \nabla L(\boldsymbol{\theta}_t). \tag{4.3}$$

For problems with objective function L such that the closed-form of ∇L can be calculated, we may use direct gradient measurements in the algorithm. For example, when training an artificial neural network, the first-order gradient for the weights in the network structure can be calculated by backpropagation [14], so the network can be optimized by gradient descent directly. (Although in practice, stochastic gradient descent or mini-batch methods are usually adopted for efficiency, we do not investigate them here.) However, there also exist cases in which the closed-form of ∇L cannot be obtained. For example, when applying the k-Markovian strategy to the Mozart Café problem, the objective function $\rho(\boldsymbol{\Theta})$ is derived from $\mathbb{E}(T_{\boldsymbol{\Theta}})$. Since the closed-form of $\mathbb{E}(T_{\boldsymbol{\Theta}})$ is not computable, the closed-form of $\nabla \rho(\boldsymbol{\Theta})$ is also not computable. In this case, we rewrite Equation 4.3 to be

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - a_t \hat{\mathbf{g}}_t(\boldsymbol{\theta}_t), \tag{4.4}$$

where $\hat{\mathbf{g}}_t(\boldsymbol{\theta}_t)$ is the noisy estimation of the gradient $\nabla L(\boldsymbol{\theta}_t)$, and a_t is the step size. The term $\hat{\mathbf{g}}_t(\boldsymbol{\theta}_t)$ is noisy because we obtain it by stochastic approximation due to the unavailability of $\nabla L(\boldsymbol{\theta}_t)$. Under appropriate conditions, the iteration of Equation 4.4 converges almost surely to $\boldsymbol{\theta}^\star$ [15].

The essential part of Equation 4.4 is $\hat{\mathbf{g}}_t(\boldsymbol{\theta}_t)$. Generally, there are two forms of interest here, namely, one-sided gradient approximation and two-sided gradient approximation. Let $y(\cdot)$ be the noisy measurement of $L(\cdot)$ (i.e., $y(\cdot) = L(\cdot)+\text{noise}$). One-sided gradient approximation involves the measurements of $y(\theta_t)$ and $y(\theta_t + \text{perturbation})$ while two-sided approximation involves measurements of $y(\theta_t \pm \text{perturbation})$ [13]. SPSA can be implemented based on either type of of gradient approximation.

4.1.2 Computation of SPSA

Let us denote the perturbation at the t-th iteration by $\boldsymbol{\Delta_t}$ ($\boldsymbol{\Delta_t}$ should be in the same dimension as $\boldsymbol{\theta}_t$). SPSA requires every element of $\boldsymbol{\theta}_t$ to be randomly perturbed together, and thus obtain two measurements of $y(\cdot)$. As introduced in [16], to guarantee the convergence of SPSA, each element of $\boldsymbol{\Delta_t}$ should be random variables that are independent, bounded, and symmetrically distributed around 0. In many cases, we may let each element of $\boldsymbol{\Delta_t}$ follow the Bernoulli distribution over $\{-1, 1\}$ with probability mass 0.5 on each. For one-sided simultaneous perturbation, each element of $\hat{\mathbf{g}}_t$ satisfies

$$\hat{\mathbf{g}}_{ti} = \frac{y(\boldsymbol{\theta}_t + c_t\boldsymbol{\Delta}_t) - y(\boldsymbol{\theta}_t)}{c_t\boldsymbol{\Delta}_{ti}}, \tag{4.5}$$

where c_t is a scaling factor for the perturbation vector which usually gets smaller as t gets larger. Similarly, for two-sided case, we have

$$\hat{\mathbf{g}}_{ti} = \frac{y(\boldsymbol{\theta}_t + c_t\boldsymbol{\Delta}_t) - y(\boldsymbol{\theta}_t - c_t\boldsymbol{\Delta}_t)}{2c_t\boldsymbol{\Delta}_{ti}}. \tag{4.6}$$

The rationale behind SPSA is to utilize two (noisy) measurements of the objective function around the current value of parameters to obtain a "pseudo-gradient" of the function. After figuring out the term $\hat{\mathbf{g}}_t(\cdot)$, we can update the parameters by Equation 4.4. In the next section, we introduce our way of estimating the rendezvous time given any model parameter $\boldsymbol{\Theta} \in \Omega$.

4.2 Estimation by Monte Carlo Simulation

As has been indicated in Equation 3.2, given that $\mathbf{\Theta} = \mathbf{\Theta}_0$, since we need to make an infinite summation and the complexity for estimating $\mathbb{P}(T_{\mathbf{\Theta}_0} = t)$ grows exponentially with t, it is impractical to calculate $\mathbb{E}(T_{\mathbf{\Theta}_0})$ by Equation 3.2. Thus, we estimate it by Monte Carlo simulation instead. For any $\mathbf{\Theta}_0 \in \Omega$, we repeat $n_{\mathbf{\Theta}_0} \in \mathbb{N}$ independent sample runs and obtain $n_{\mathbf{\Theta}_0}$ sample rendezvous times. As we have explained in Section 3.2.3, $n_{\mathbf{\Theta}_0}$ should make the 95% confidence interval of $\mathbb{E}(T_{\mathbf{\Theta}_0})$ have width less than some pre-specified threshold. We simulate one sample run in the following way.

Algorithm 1 Sample run for $\mathbf{\Theta}_0$

```
    Input : Model parameter $\mathbf{\Theta}_0$
    Output : Sample rendezvous time $\hat{T}_{\mathbf{\Theta}_0}$
 1: procedure SAMPLE RUN($\mathbf{\Theta}_0$)
 2:     label1 ← {$i$:None for $i = 1, \dots, n$}, label2 ← {$i$:None for $i = 1, \dots, n$}
 3:     init1 ← random integer($1, \dots, n$), init2 ← random integer ($1, \dots, n$)
 4:     label1[1]=init1 and label2[1]=init2
 5:     if init1 == init2 then
 6:         $\hat{T}_{\mathbf{\Theta}_0} = 1$, return $\hat{T}_{\mathbf{\Theta}_0}$
 7:     else
 8:         trajectory1 = [0], trajectory2 = [0]
 9:         timer = 1
10:         while label1[trajectory1[-1]] $\neq$ label2[trajectory2[-1]] do
11:             Compute probability distribution by Equation 3.1 for each agent
12:             Determine their next steps $(x_1, x_2)$ by the probability distribution
13:             if $x_1$ has not been visited by agent1 then
14:                 Assign the smallest feasible natural number to agent1[$x_1$]
15:             if $x_2$ has not been visited by agent1 then
16:                 Assign the smallest feasible natural number to agent2[$x_2$]
17:             Add agent1[$x_1$] to trajectory1, add agent2[$x_2$] to trajectory2
18:             timer = timer + 1
19:         $\hat{T}_{\mathbf{\Theta}_0}$ =timer, return $\hat{T}_{\mathbf{\Theta}_0}$
```

Note that label1 and label2 are two hash tables whose keys are the actual indices of cafés and the values are the agents' labellings, and trajectory1 and trajectory2 are two lists which are used to store the visiting history (in the agents' labellings) of the each agent. Based on Algorithm 1, $\hat{E}(T_{\mathbf{\Theta}_0})$ is estimated by taking the average of sample rendezvous time over $n_{\mathbf{\Theta}_0}$ replications. Now we have demonstrated the way to estimate the expected rendezvous time, we introduce the parameter optimization in the next section.

4.3 Search for $\mathbf{\Theta}^\star$

Given that SPSA does not depend on the exact form of the first-order gradient and it can take noisy measurements of the objective function, we apply SPSA to the parameter optimization of the k-Markovian model. We let $\rho(\cdot) : \mathbb{R} \to \mathbb{R}$ be the objective function that takes the input $\hat{\mathbb{E}}(T_{\mathbf{\Theta}})$ (so, $\rho(\mathbf{\Theta})$ contains noise) which satisfies condition 4.2. Denote the parameter in the j-th iteration by $\mathbf{\Theta}_j$. SPSA iteratively updates $\mathbf{\Theta}$ by

$$\mathbf{\Theta}_{j+1} = \mathbf{\Theta}_j - a_j \cdot \mathbf{g}_j(\mathbf{\Theta}_j), \tag{4.7}$$

where $\mathbf{g}_j(\mathbf{\Theta}_j) \in \mathbb{R}^{|\mathbf{\Theta}|}$ is the "pseudo-gradient" estimated by SPSA and a_j is the gain coefficient that scales $\mathbf{g}_j(\mathbf{\Theta}_j)$. Each entry l of $\mathbf{g}_j(\mathbf{\Theta}_j)$ satisfies

$$\mathbf{g}_{jl}(\mathbf{\Theta}_j) = \frac{\rho(\mathbf{\Theta}_j + c_j\mathbf{\Delta}_j) - \rho(\mathbf{\Theta}_j - c_j\mathbf{\Delta}_j)}{2c_j}\mathbf{\Delta}_{jl}^{-1} \text{ for } l = 1, \dots, |\mathbf{\Theta}|. \tag{4.8}$$

In our implementation, the entries of the perturbation $\mathbf{\Delta}_j$ independently and identically follow the Bernoulli distribution over $\{-1, 1\}$ with equal probability.

Other major hyper-parameters of SPSA include A, a, α, c, γ, among which A, a, α affect the gain sequence $\{a_j\}_{j=0}^{\infty}$ and c, γ affect the sequence $\{c_j\}_{j=0}^{\infty}$ that scales the magnitude of perturbation. The relationships are shown below.

$$a_j = \frac{a}{(A + j + 1)^\alpha}. \tag{4.9}$$

$$c_j = \frac{c}{(j + 1)^\gamma}. \tag{4.10}$$

The reason why we compute $\{a_j\}_{j=0}^{\infty}$ by 4.9 is because the gain sequence for a stochastic approximation algorithm should satisfy the following condition 4.11 [17, 18], to guarantee convergence.

$$a_j > 0, a_j \to 0, \sum_{j=0}^{\infty} a_j = \infty, \text{ and } \sum_{j=0}^{\infty} a_j^2 < \infty. \tag{4.11}$$

Thus, we must have $\alpha \in (0.5, 1]$. Meanwhile, as indicated in [19], although the constant A is not typically shown in stochastic approximation literature, engineers have found that if we include this "stability constant", we can set a more aggressive, or in other words, larger a in the numerator to avoid instabilities in early iterations. Practically, engineers may choose A such that it is below 10% of the maximum number of iterations. When it comes to a, let us first denote the magnitude of the elements in $\mathbf{g}_0(\boldsymbol{\Theta}_0)$ by $||\mathbf{g}_0(\boldsymbol{\Theta}_0)||_\infty$ (assume that the elements in $\mathbf{g}_0(\boldsymbol{\Theta}_0)$ are of the same magnitude). Further, we denote the smallest desired change of magnitudes among the elements in $\boldsymbol{\Theta}$ in early iterations by $\delta||\boldsymbol{\Theta}||_\infty$. Then, engineers may choose a that satisfies

$$\frac{a}{(1+A)^\alpha} \cdot ||\mathbf{g}_0(\boldsymbol{\Theta}_0)||_\infty \approx \delta||\boldsymbol{\Theta}||_\infty.$$

In other words, in early iterations, because we need the algorithm to search in the parameter space efficiently, we may have a "bottom line" for the change of model parameters. Selecting an a in this way guarantees that the change of model parameters is no less than the "bottom line" (note that the gain $a_j = \frac{a}{(1+A)^\alpha}$ when $j = 0$).

For the sequence $\{c_j\}_{j=0}^{\infty}$, we use γ and c to scale the magnitude of perturbation. The sequence $\{c_j\}_{j=0}^{\infty}$ should also satisfy

$$c_j > 0, c_j \to 0, \sum_{j=0}^{\infty} \left(\frac{a_j}{c_j}\right)^2 < \infty. \tag{4.12}$$

As for γ, as suggested by [15, 20], the asymptotically optimal value is $\frac{1}{6}$. When it comes to c, with the Bernoulli ± 1 distribution for the elements in $\boldsymbol{\Delta}$, we may set c at a level approximately equal to the standard deviation of the measurement noise in $\rho(\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}_1}))$ so that the elements of $\mathbf{g}_j(\boldsymbol{\Theta}_j)$ will not get excessively large in magnitude.

The standard deviation can be estimated from several sample $\rho(\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}_1}))$ values at the initial guess $\boldsymbol{\Theta}_0$.

With the help of SPSA, we expect to approximate the optimal values of model parameters. In the next chapter, we present the results of simulation studies. But before that, we need to claim that the choice of $\rho(\cdot)$ and the hyper-parameter tuning are determined by trial and error. Above all, we always need to make adjustments to the settings of an algorithm when applying it on different problems.

Chapter 5

Simulation Studies

In this chapter, we provide the simulation results on $n = 2, 3, 4$ cases[1]. For each case, we first introduce the settings of the simulation which mainly include the hyper-parameters for the k-Markovian model and SPSA, and the design of the objective function $\rho(\cdot)$. Recall that for any given set of model parameters $\mathbf{\Theta}$, in each sample run, the agents start randomly and follow the strategy "encrypted" in $\mathbf{\Theta}$ until they meet. The number of repetitions for the sample run is dynamically adjusted until the length of the 95% confidence interval for $\mathbb{E}(T_{\mathbf{\Theta}})$ is below some pre-specified threshold. We refer to this pre-specified threshold as the admissible length. Since the computing time has a positive relationship with the number of cafés and a negative relationship with the admissible length, we choose the admissible length as small as the computing power permits for each value of n. Apart from the number of cafés and the admissible length, another factor that affects the running time is the value of k in the k-Markovian model. Since random search has expected rendezvous time n, it is expected that an optimal set of parameters $\mathbf{\Theta}^\star$ found by SPSA will produce an estimated expected rendezvous time $\leq n$. Thus, the first n steps are the most important for a rendezvous search process. On the one hand, if k is too small, it cannot represent all possible strategies in the first n steps. On the other hand, if k is too large, then the number of parameters in $\mathbf{\Theta}$ is too large, which makes the computing time too long. As a rule

[1]The transition probabilities can be found at: https://github.com/jamespengcheng/k-Markovian

of thumb, we set $k = n + 1$ in each case to strike a balance between the power of representation and the computing time.

In Sections 5.1.1, 5.2.1, 5.3.1, we visualize the optimization process and provide the final results. Empirically, we find that using $\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}})$ directly as the objective function is good enough. Thus, later in this chapter, the term "objective function" means $\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}})$. Based on the optimal set of model parameters $\boldsymbol{\Theta}^{\star}$ found by SPSA, we compute the corresponding transition probabilities and interpret the underlying optimal strategy. Additionally, to verify that $\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}^{\star}})$ is estimated correctly, we conduct another Monte Carlo simulation for the optimal estimated expected rendezvous time based directly on the transition probabilities calculated from $\boldsymbol{\Theta}^{\star}$. If the result is consistent with the minimum estimated expected rendezvous time searched by SPSA, then our optimal strategy is valid.

In the following sections of this chapter, we present the results when $n = 2, 3, 4$. Meanwhile, the AW(n) strategy is regarded as a benchmark for comparison. All simulations were done on a MacBook Pro (13-inch, M1, 2020) with 8 GB memory.

5.1 $n = 2$ Case

5.1.1 Settings and Results

In the optimization process, it is the measurement of the objective function that takes the majority of computing time because we need to repeat a sufficiently large number of sample runs of rendezvous search to make the width of the confidence interval for $\mathbb{E}(T_{\boldsymbol{\Theta}})$ be less than the admissible length. Thus, the admissible length directly affects the running time of objective function measurements. When choosing

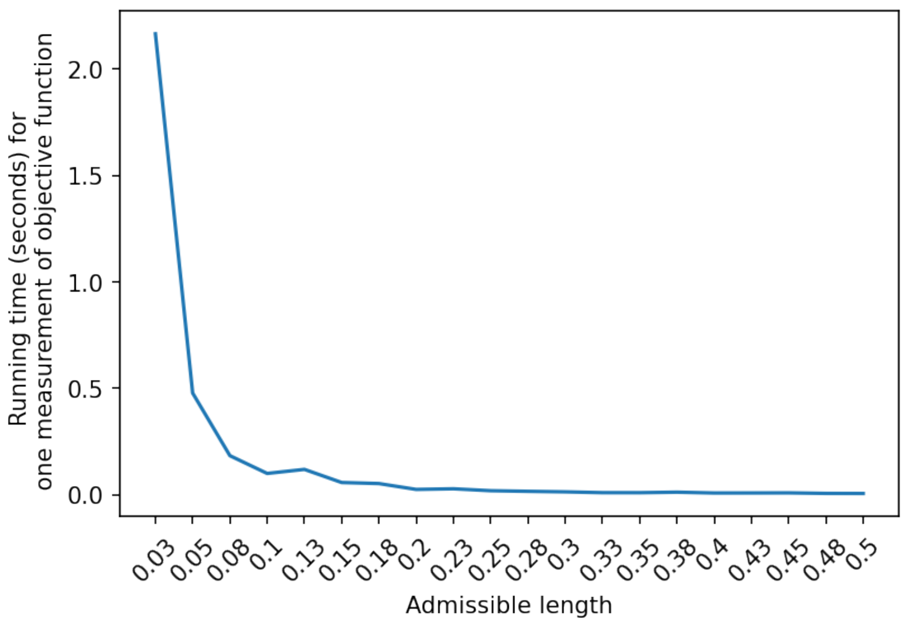


Figure 5-1. Effect of admissible length on running time (2-café)

the admissible length, we should balance the tradeoff between accuracy (the smaller admissible length, the better) and computational efficiency (the larger admissible length, the better). For the admissible length in the $n = 2$ case, we conduct a random experiment to measure the average running time of one measurement of the objective function. The admissible lengths range from 0.03 to 0.5 with common difference. For each admissible length, we collect 50 samples of running time where in each sample, the model parameters are initialized randomly. Then, we take the average of the 50 samples and regard it as the average running time for this admissible length. The relationship between the admissible length and the running time (in seconds) for one measurement of the objective function is shown in Figure 5-1.

Although there is a huge jump in terms of the running time when adjusting the admissible length from 0.05 to 0.03, one measurement of objective function in the 0.03 case only takes a little bit more than 2 seconds, which is not bad given that SPSA only measures the objective function twice in one iteration. We set the admissible length as 0.03 in order to obtain more accurate results.

In the $n = 2$ case, the AW(2) strategy illustrates that random search is optimal and the expected rendezvous time should be 2. When it comes to our k-Markovian model, it is equivalent to setting every parameter in $\mathbf{\Theta}$ to be the same. To verify this, we initialize a k-Markovian model such that every parameter is 1. Under the setting of admissible length being 0.03, we obtain average rendezvous time to be 1.999 with standard deviation to be 0.013. Thus, our simulation program is reliable in that the estimated expected rendezvous time is almost the same as the theoretical expectation.

For the hyper-parameters in k-Markovian model, we set $k = 3$. For the hyper-parameters in SPSA, we set $a = 0.2$, $c = 0.012$, and the maximum number of iterations to be 800. The optimization process is shown in Figure 5-2. We stop the optimization process at the 800th iteration because we find that there is not much change in the estimated expected rendezvous time after 600 iterations. The optimal solution searched by SPSA is $\hat{\mathbb{E}}(T_{\mathbf{\Theta}^\star}) = 2.0244$ with 95% confidence interval $[2.0094, 2.0394]$.

To verify that the optimal strategy and optimal estimated expected rendezvous time are correct, we extract the transition probabilities from $\mathbf{\Theta}^\star$. The strategy is presented in the table in Figure 5-3. Please note that we have kept three decimal places for the convenience of presentation.

For the table in Figure 5-3, the explanations of columns are listed below.

- **n_explored**: The number of cafés that the agent has already visited.
- **partial**: True if this is the entire visiting history of the agent, and False if this is just the history in the last k steps (because the entire visiting history is longer than k).

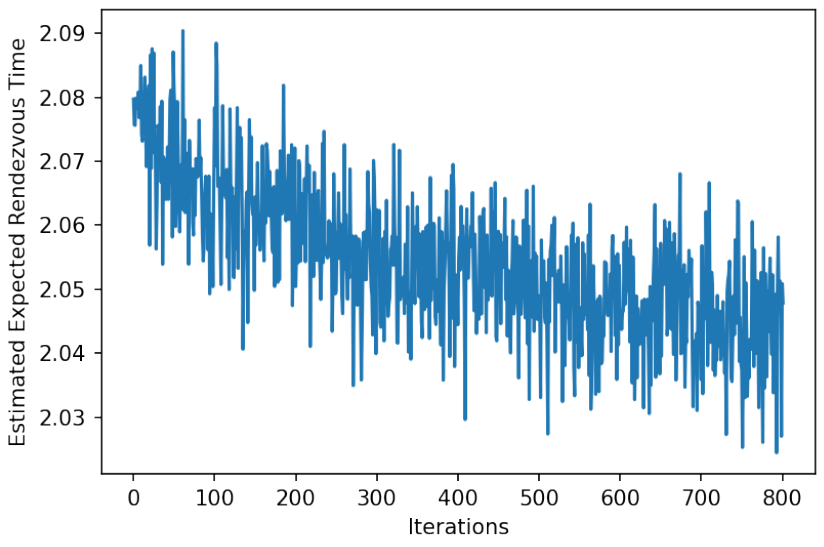
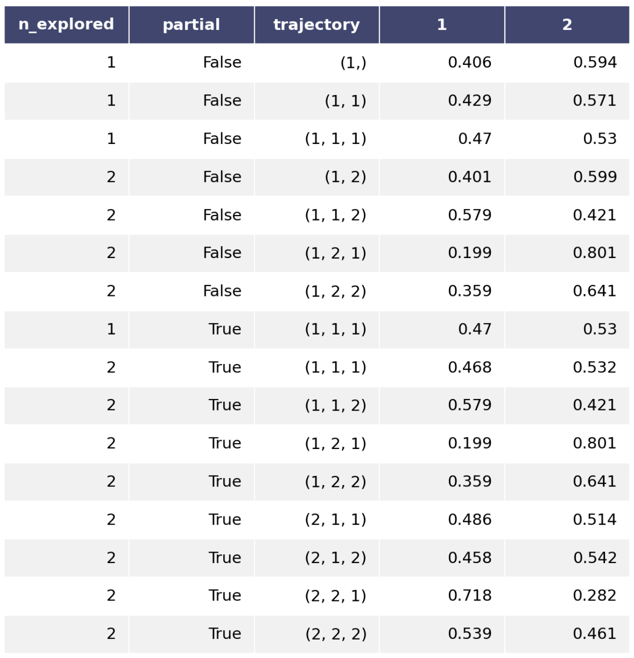


Figure 5-2. Optimization process (2-café)



n_explored	partial	trajectory	1	2
1	False	(1,)	0.406	0.594
1	False	(1, 1)	0.429	0.571
1	False	(1, 1, 1)	0.47	0.53
2	False	(1, 2)	0.401	0.599
2	False	(1, 1, 2)	0.579	0.421
2	False	(1, 2, 1)	0.199	0.801
2	False	(1, 2, 2)	0.359	0.641
1	True	(1, 1, 1)	0.47	0.53
2	True	(1, 1, 1)	0.468	0.532
2	True	(1, 1, 2)	0.579	0.421
2	True	(1, 2, 1)	0.199	0.801
2	True	(1, 2, 2)	0.359	0.641
2	True	(2, 1, 1)	0.486	0.514
2	True	(2, 1, 2)	0.458	0.542
2	True	(2, 2, 1)	0.718	0.282
2	True	(2, 2, 2)	0.539	0.461

Figure 5-3. 2-café transition probability

- **trajectory**: The visiting history that has been considered for the next step's decision.
- **1**: The probability of going to café 1.
- **2**: The probability of going to café 2.

Note that there exist identical probability distributions in this table. For example, when the trajectory is $(1, 1, 2)$ (so, the agent must have explored both cafés), whatever it is a partial trajectory or not a partial trajectory, we extract the probability distribution from the same subset of parameters (in this case, the subset of parameters have dimension $2^3 \times 2$ where 3 represents the length of visiting history that is taken into consideration and the last 2 represents the two choices of the next step). Thus, the resulting transition probabilities are the same. Later, by conducting simulations of rendezvous search based on the strategy described in this table, the average rendezvous time is 2.0311 with 95% confidence interval $[2.0161, 2.0461]$, which is close to the optimal result given by SPSA. We ascribe the difference between the optimal result given by SPSA and the average rendezvous time estimated from the table to numerical error in scientific computing.

5.1.2 Strategy Interpretation

We interpret the optimal strategy found by SPSA according to the probability values shown in Figure 5-3. For any agent, when he has just explored one café, there is always a larger probability for going to café 2 than staying still. That is to say, the strategy prefers exploring unvisited places rather than conservatively waiting for the other agent to come. Meanwhile, we observe that when **n_explored** is 2, **partial** is True, and the trajectory is $(1, 1, 1)$ or $(2, 2, 2)$, there is a significantly larger probability of going to the other café in the next step. Thus, the same ideology applies to the case in which the agent has visited both cafés but have stayed in the same one for several consecutive time steps. However, when it comes to cases such that in the previous three steps, the agent stayed in one café for two steps while stayed in the other for one step, more probability mass is put on the café that he has stayed for two steps. One possible conjecture for this phenomenon is that the strategy is 3-Markovian, and thus the agent may stay consecutively for three steps before moving to the other café. To some extent, this is similar to the AW strategy. (Recall that the agent may stay consecutively in the AW strategy.)

5.2 $n = 3$ Case

5.2.1 Settings and Results

Similarly to the $n = 2$ case, we adjust the admissible length from 0.03 to 0.5, and for each value of admissible length, we compute the average running time for one estimate of the objective function out of 50 independent samples (all with random values for $\mathbf{\Theta}$). We obtain the relationship between the admissible length and the running time for one estimation of the objective function in the $n = 3$ case as shown in Figure 5-4. From the figure, we observe that it takes over 18 seconds when the admissible length is 0.03. Since one iteration of SPSA requires two measurements of the objective function, i.e., to measure $\hat{\mathbb{E}}(T_{\mathbf{\Theta}+\text{perturbation}})$ and $\hat{\mathbb{E}}(T_{\mathbf{\Theta}-\text{perturbation}})$, and it took 800 iterations in the $n = 2$ case, we take 0.05 as the admissible length.

For the hyper-parameters in k-Markovian model, we set $k = 4$. For the hyper-parameters in SPSA, we set $a = 0.2$, $c = 0.008$, and the maximum number of iterations to be 800. The value of c is equal to the standard deviation of 10 independent estimates of the objective function at the initial value of $\mathbf{\Theta}$. The optimization process is shown in Figure 5-5. We stop the optimization process at the 800th iteration because we find that there is not much change in the estimated expected rendezvous time after

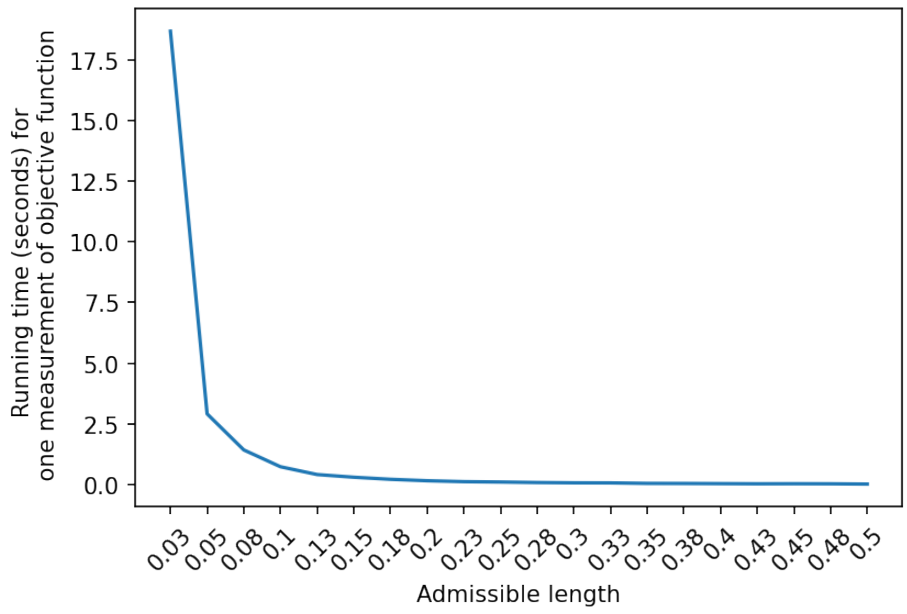


Figure 5-4. Effect of admissible length on running time (3-café)

700 iterations. The optimal solution found by SPSA is $\hat{\mathbb{E}}(T_{\mathbf{\Theta}^\star}) = 2.8525$ with 95% confidence interval $[2.8130, 2.8629]$.

To verify that the optimal strategy and optimal estimated expected rendezvous time are correct, we extract the transition probabilities from $\mathbf{\Theta}^\star$. Unlike the transition probabilities in the $n = 2$ case, the number of combinations of (**n_explored**, **partial**, **trajectory**) is too large to be presented in the main body of this thesis. We present part of the table in Figure 5-6, and the whole table can be found in the author's GitHub repository. By conducting simulation for rendezvous search based on the strategy described in this table, the average rendezvous time is 2.8874 with 95% confidence interval $[2.8625, 2.9124]$, which is close to the optimal result given by SPSA.

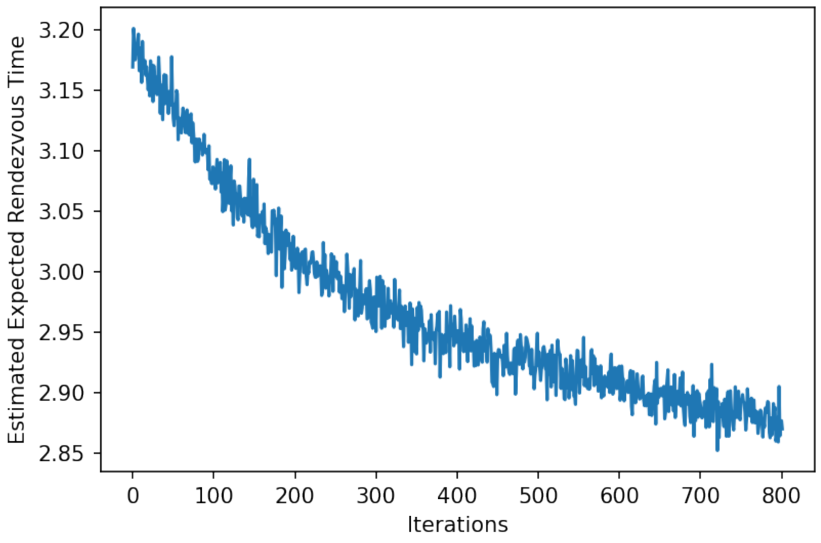
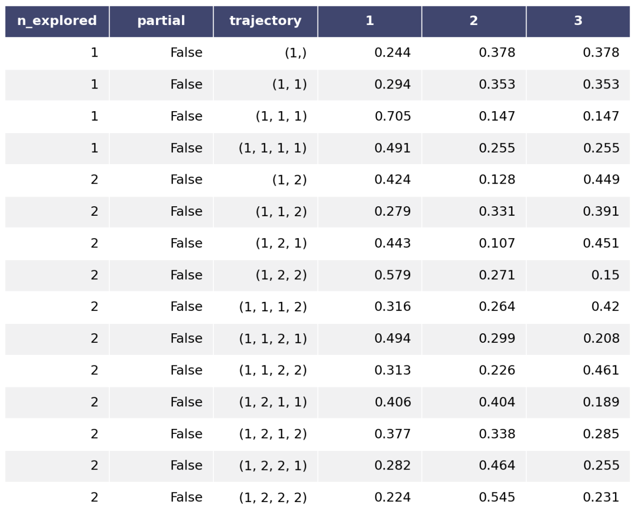


Figure 5-5. Optimization process (3-café)



n_explored	partial	trajectory	1	2	3
1	False	(1,)	0.244	0.378	0.378
1	False	(1, 1)	0.294	0.353	0.353
1	False	(1, 1, 1)	0.705	0.147	0.147
1	False	(1, 1, 1, 1)	0.491	0.255	0.255
2	False	(1, 2)	0.424	0.128	0.449
2	False	(1, 1, 2)	0.279	0.331	0.391
2	False	(1, 2, 1)	0.443	0.107	0.451
2	False	(1, 2, 2)	0.579	0.271	0.15
2	False	(1, 1, 1, 2)	0.316	0.264	0.42
2	False	(1, 1, 2, 1)	0.494	0.299	0.208
2	False	(1, 1, 2, 2)	0.313	0.226	0.461
2	False	(1, 2, 1, 1)	0.406	0.404	0.189
2	False	(1, 2, 1, 2)	0.377	0.338	0.285
2	False	(1, 2, 2, 1)	0.282	0.464	0.255
2	False	(1, 2, 2, 2)	0.224	0.545	0.231

Figure 5-6. 3-café transition probability

5.2.2 Strategy Interpretation

Since the table of transition probabilities is too large to present in the passage, we plot the data from the table to draw some illustrations from the strategy. We have divided the data into two groups. The first group is for the cases with **partial**=False and the plots are shown in Figure 5-7, while the second group is for the cases with **partial**=True and the plots are shown in Figure 5-8. Within each group, we plot the average probabilities of going to each café respectively. That is to say, one plot summarizes the average probability of going to café i ($i = 1, 2, 3$) in different "cases". Here, the term "case" includes the number of cafés that have been explored (i.e., **n_explored**) and the number of occurrences for i in the trajectory. Thus, in each plot, we have partitioned the data into three groups based on **n_explored**. In each group, we plot the average probability of going to café i based on the number of its occurrences in the trajectory. We should note that there are some missing bars in the plots and that is because such cases are impossible. For example, in plot (a) of Figure 5-7, if two cafés are explored, café 1 cannot occur four times in the trajectory because otherwise café 2 will not be visited in the first four steps.

The three plots in Figure 5-7 are for full trajectories, i.e., trajectories with length no more than 4 since $k = 4$. We observe that there is a roughly upward-sloping relationship between the number of occurrences in the trajectory and the average probability. However, in Figure 5-8 where the three plots are for partial trajectories, we observe that there is a roughly downward-sloping relationship between them. We conjecture that the k-Markovian strategy searched by SPSA tends to be conservative when the agent has not visited all cafés, and thus the more occurrences of a café in the trajectory, the more likely the agent will stay still. However, once the agent has visited all cafés, the strategy becomes radical, and thus the more occurrences of a café, the less likely the agent will stay still.

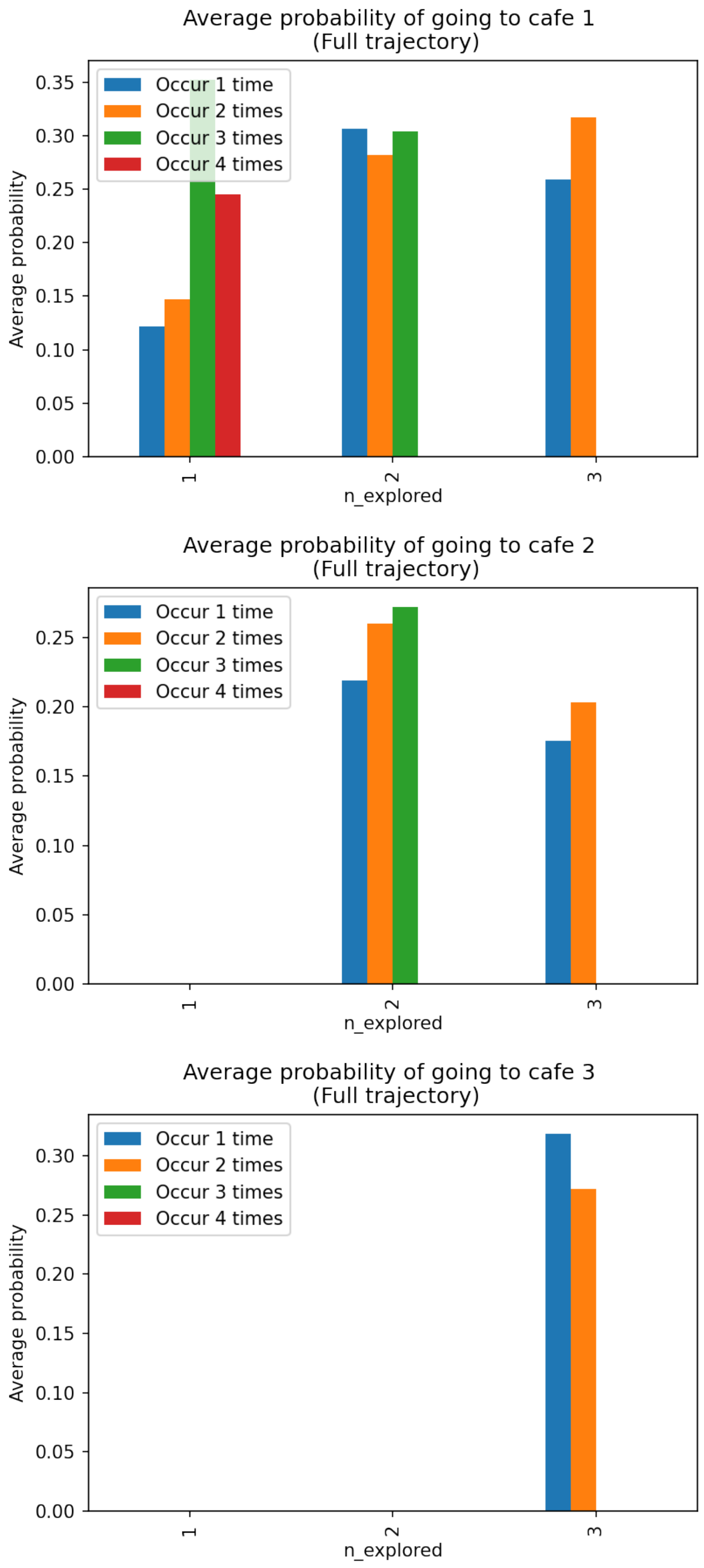


Figure 5-7. n=3, partial=False

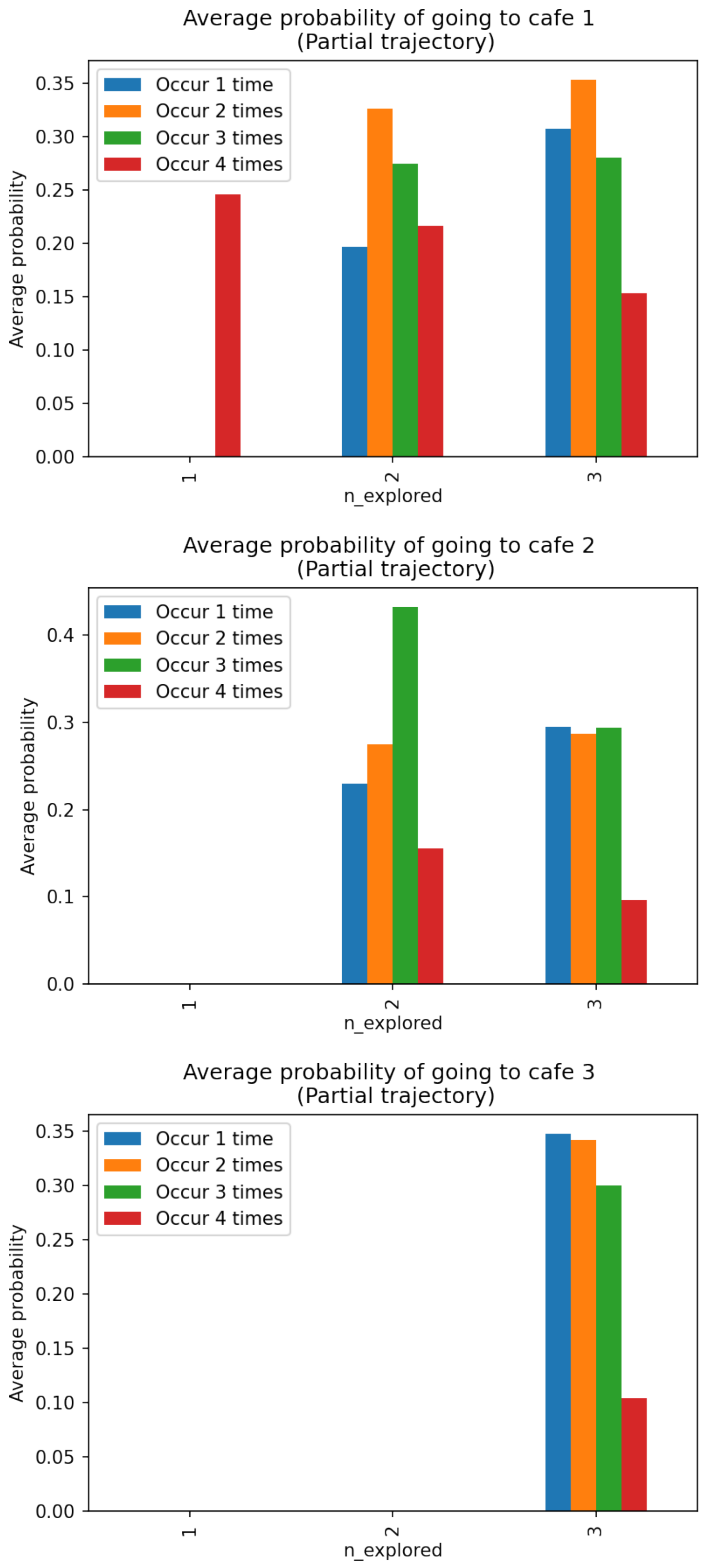


Figure 5-8. n=3, partial=True

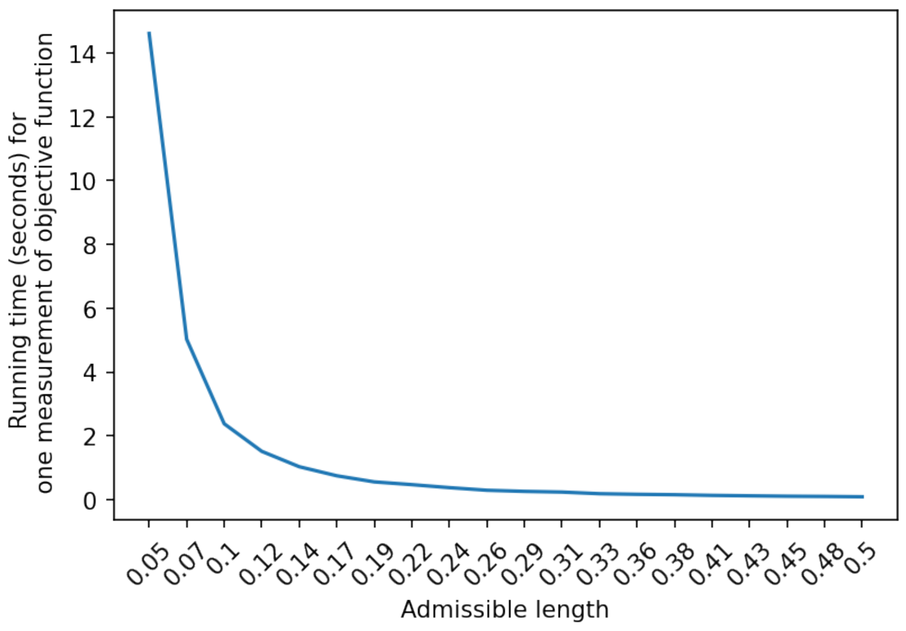


Figure 5-9. Effect of admissible length on running time (4-café)

5.3 $n = 4$ Case

5.3.1 Settings and Results

In the $n = 4$ case, we compute the average running time for one measurement of the objective function for each value of admissible length which ranges from 0.05 to 0.5. We no longer test 0.03 because the computing time is too long. The relationship between the admissible length and the average running time for one measurement of objective function is shown in Figure 5-9.

We observe that the average running time is above 14 seconds when the admissible length is 0.05, but it drops sharply to around 5 seconds when the admissible length is 0.07. Thus, we take 0.06 as the admissible length to strike a balance between the efficiency and accuracy. By running 1000 iterations, we obtain the curve of the optimization process as shown in Figure 5-10.

The least estimated expectation for the rendezvous time is 3.9267 with 95% confidence interval $[3.9017, 3.8517]$. Part of the transition probabilities is shown

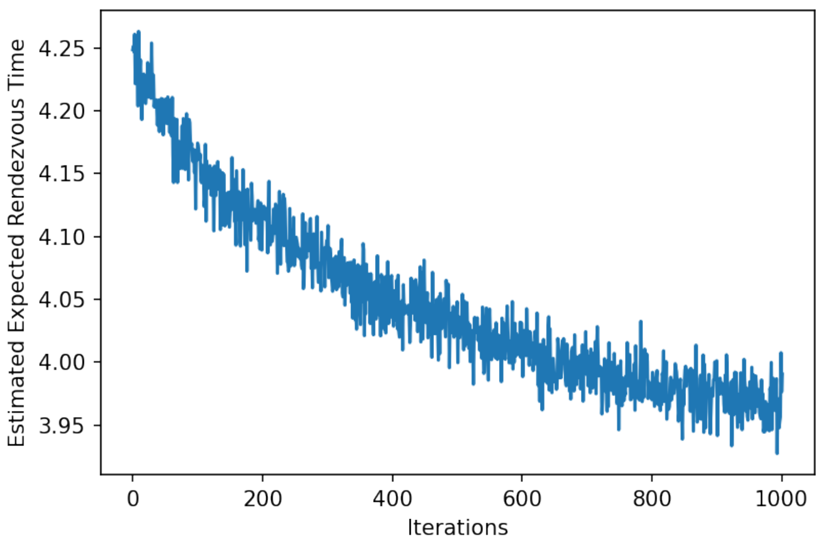


Figure 5-10. Optimization process (4-café)

in Figure 5-11. The full set of transition probabilities is provided in the author's GitHub repository. By testing directly on the transition probabilities, we get average rendezvous time 3.9574 with 95% confidence interval $[3.9274, 3.9874]$. This result is close to the result given by the optimal value of the k-Markovian model.

5.3.2 Strategy Interpretation

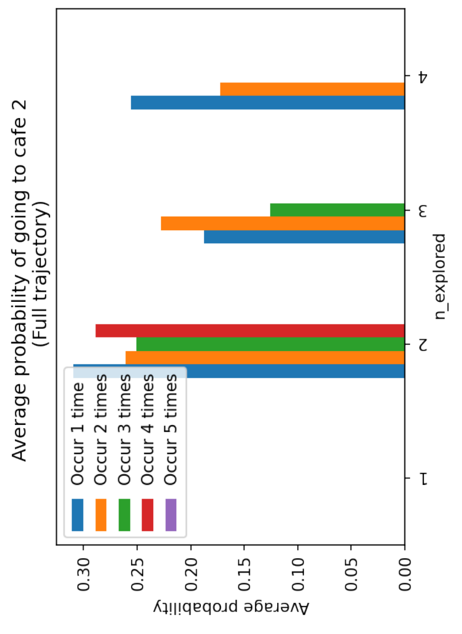
Similar to the data analysis that we have done for the $n = 3$ case, the average probabilities in the $n = 4$ case are shown in Figures 5-12 (for full trajectories) and 5-13 (for partial trajectories). Each plot in a figure represents the average probabilities of going to a specific café in the next step, and thus there are four plots in each figure. Since we have 4 cafés in total, there are 4 groups of bars in each plot that are partitioned by **n_explored**. As this is a 5-Markovian strategy, each café may appear at most 5 times in the trajectory, and hence there are at most 5 bars for each value of **n_explored**.

Unlike the results in Section 5.2.2, in Figure 5-12, we observe a roughly downward-sloping relationship between the number of occurrences of café i ($i = 1, 2, 3, 4$) in the

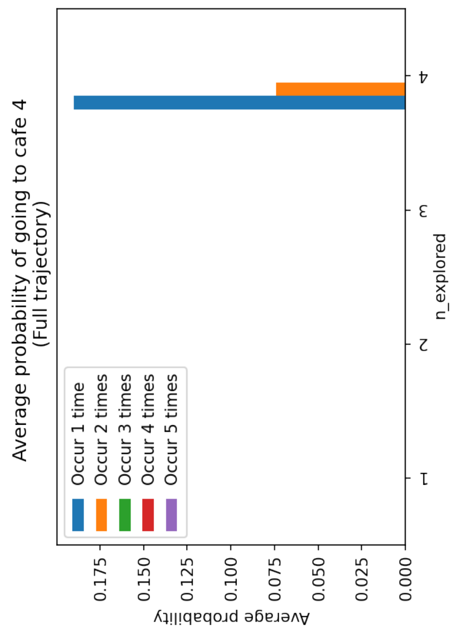
n_explored	partial	trajectory	1	2	3	4
1	False	(1,)	0.196	0.268	0.268	0.268
1	False	(1, 1)	0.348	0.217	0.217	0.217
1	False	(1, 1, 1)	0.734	0.089	0.089	0.089
1	False	(1, 1, 1, 1)	0.677	0.108	0.108	0.108
1	False	(1, 1, 1, 1, 1)	0.389	0.204	0.204	0.204
2	False	(1, 2)	0.215	0.383	0.201	0.201
2	False	(1, 1, 2)	0.619	0.202	0.089	0.089
2	False	(1, 2, 1)	0.154	0.598	0.124	0.124
2	False	(1, 2, 2)	0.196	0.161	0.321	0.321
2	False	(1, 1, 1, 2)	0.402	0.331	0.134	0.134
2	False	(1, 1, 2, 1)	0.36	0.476	0.082	0.082
2	False	(1, 1, 2, 2)	0.328	0.308	0.182	0.182
2	False	(1, 2, 1, 1)	0.458	0.409	0.067	0.067
2	False	(1, 2, 1, 2)	0.153	0.194	0.327	0.327
2	False	(1, 2, 2, 1)	0.561	0.256	0.091	0.091
2	False	(1, 2, 2, 2)	0.504	0.284	0.106	0.106
2	False	(1, 1, 1, 1, 2)	0.577	0.212	0.105	0.105
2	False	(1, 1, 1, 2, 1)	0.176	0.257	0.284	0.284
2	False	(1, 1, 1, 2, 2)	0.249	0.208	0.271	0.271
2	False	(1, 1, 2, 1, 1)	0.206	0.396	0.199	0.199
2	False	(1, 1, 2, 1, 2)	0.16	0.433	0.203	0.203
2	False	(1, 1, 2, 2, 1)	0.161	0.367	0.236	0.236
2	False	(1, 1, 2, 2, 2)	0.462	0.229	0.154	0.154
2	False	(1, 2, 1, 1, 1)	0.327	0.146	0.264	0.264
2	False	(1, 2, 1, 1, 2)	0.386	0.366	0.124	0.124
2	False	(1, 2, 1, 2, 1)	0.236	0.398	0.183	0.183
2	False	(1, 2, 1, 2, 2)	0.198	0.457	0.172	0.172
2	False	(1, 2, 2, 1, 1)	0.178	0.174	0.324	0.324
2	False	(1, 2, 2, 1, 2)	0.229	0.411	0.18	0.18
2	False	(1, 2, 2, 2, 1)	0.382	0.123	0.247	0.247

Figure 5-11. 4-café transition probability

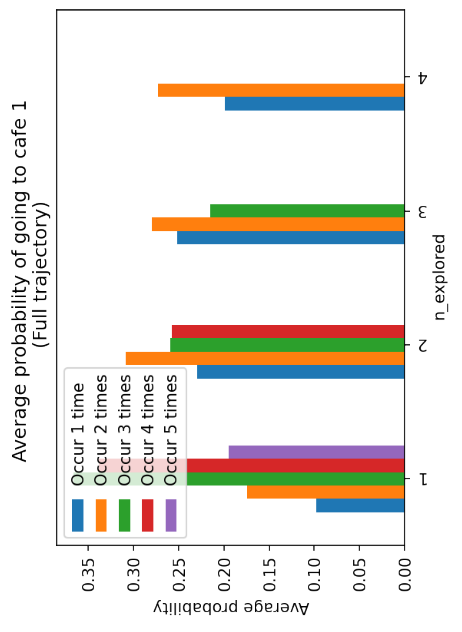
trajectory and the average probability of going to café i whether the trajectory is partial or not. That is to say, in the past 5 steps, the fewer times an agent visits a café, the more likely he will get there in the next step. Thus, the strategy searched by SPSA in the $n = 4$ case is more radical that that of $n = 3$ case.



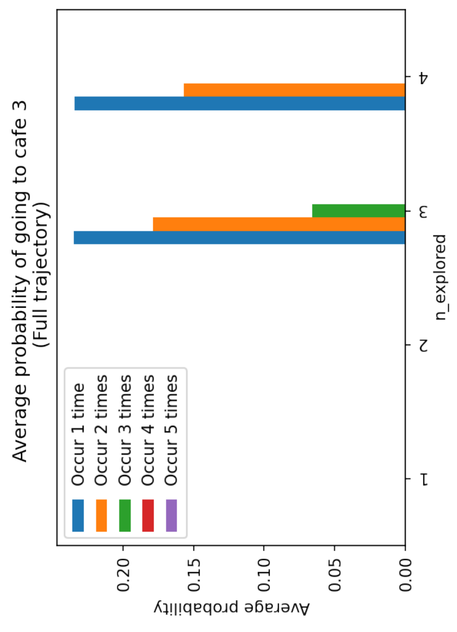
(a)



(b)

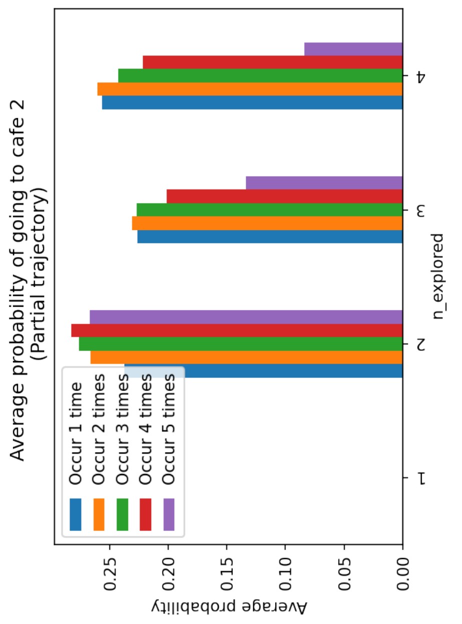


(c)

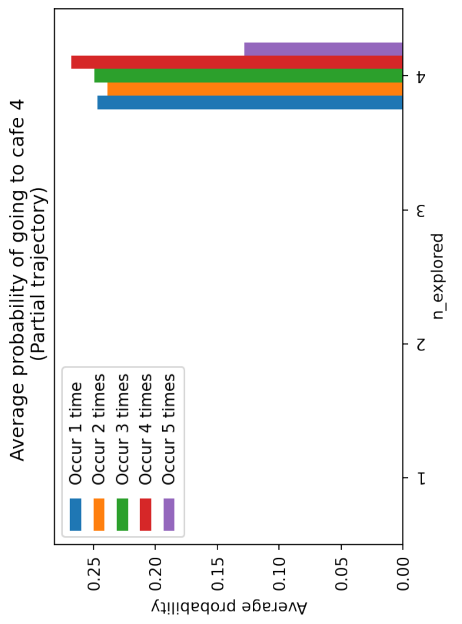


(d)

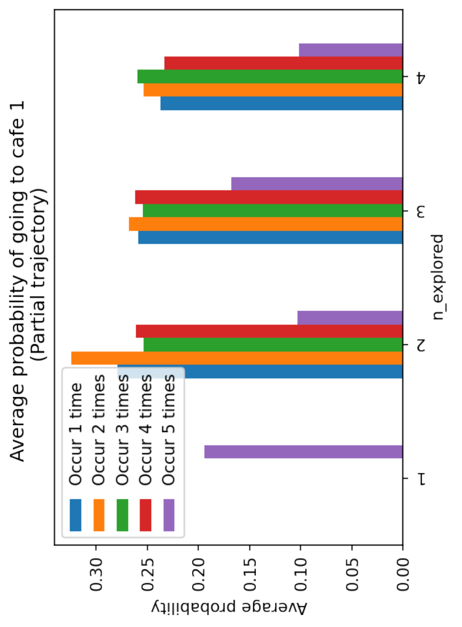
Figure 5-12. n=4, partial=False



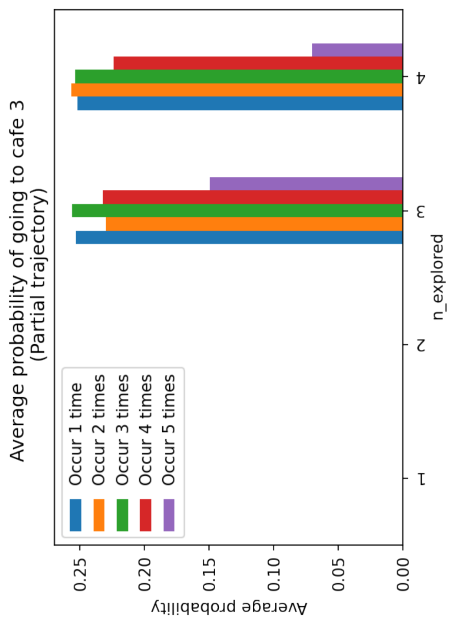
(a)



(b)



(c)



(d)

Figure 5-13. n=4, partial=True

Chapter 6

Discussions

Based on the results in Chapter 5, we realize that there exist some issues in the k-Markovian modelling since the optimal estimated expectation of rendezvous time is significantly larger than the AW strategy when $n = 3$ or 4. (Nevertheless, they are both better than random search.) We ascribe the failure for k-Markovian to surpass AW(3) and AW(4) strategies to

- Possible ignorance of some necessary conditions for an optimal strategy.
- Low efficiency of Monte Carlo estimation.
- Limited computing power.

In our k-Markovian modelling, the only necessary condition for the optimal strategy that we have considered is the equal probability for unexplored cafés. This condition is reasonable in that the agent knows nothing about the unexplored cafés, and thus there should be no priority. Indeed, we did not even consider this condition in the first version of k-Markovian modelling, and the results were significantly worse than what we have presented in this thesis. That is to say, in the first version, for whatever step before rendezvous, there were n independent parameters in $\mathbf{\Theta}$ that were used to calculate the transition probability, and thus there was no guarantee that the probabilities of going to unexplored cafés would converge to the same value. Although the model can represent more strategies in this way, the ignorance of equal probability for unexplored cafés has largely increased the difficulty of optimization. Therefore, for future works on the general Mozart Café problem, it will be better if researchers can prove more necessary conditions for the optimality of a strategy. Then, they may modify the k-Markovian strategy so that fewer parameters can be used and it will be less likely to encounter the issue of the curse of dimensionality when n and k become larger.

Another issue that has limited our exploration is the lack of efficiency for the Monte Carlo simulations. It always takes tens of thousands of sample runs before the length of the confidence interval for $\mathbb{E}(T_{\boldsymbol{\Theta}})$ to be less than the admissible length. As we had presented in Chapter 5, one measurement of the loss function takes more than 10 seconds when $n = 4, k = 5$. The slow computation together with the limitation of computing power have limited our exploration. We did not have the chance to try larger k's while keeping the admissible length to be sufficiently small since the computing time grows sharply. We encourage future researchers to investigate more efficient methods to obtain $\hat{\mathbb{E}}(T_{\boldsymbol{\Theta}})$ with a sufficiently small 95% confidence interval instead of using Monte Carlo simulation.

Conclusion and Future Work

In this thesis, we first introduced some preliminary work on the token approach, and showed that a lower bound for the minimum expected rendezvous time in the $n = 4$ case is 3.5685. Then, we introduced the k-Markovian model for the general Mozart Café problem, which is also the first parameterized model for this problem. The expected rendezvous time for any certain model parameters are estimated by Monte Carlo simulation and the model parameters are optimized by SPSA due the lack of first order gradient. To some extent, the k-Markovian model is a method of exhaustion since it uses parameters to represent all possible trajectories (within k steps). The advantage of such modelling is that the model is able to represent all possible strategies within the first k steps since each parameter is used to compute the transition probability for one possible of trajectory. But the disadvantage is also obvious, which is the curse of dimensionality. As each parameter is used for one possible trajectory, the number of parameters grows exponentially when n increases, which makes the parameter optimization much harder. However, for an optimal solution, there should be some trajectories that have very tiny probability to happen, and thus the corresponding parameters are essentially useless for the model.

For future research work tackling the general Mozart Café problem, the most important aspect that can be improved is to find and show more necessary conditions for the optimality of a strategy. This should be very helpful to reduce the number of parameters. Besides, many more experiments can be made if future researchers can find methods that are more efficient than Monte Carlo simulation when evaluating the expected rendezvous time.

References

1. Stone, L. D. *Theory of Optimal Search* (Elsevier, 1976).
2. Mortensen, D. T. Job search and labor market analysis. *Handbook of Labor Economics* **2,** 849–919 (1986).
3. Alpern, S. Rendezvous search: A personal perspective. *Operations Research* **50,** 772–795 (2002).
4. Isaacs, R. *Differential Games* (John Wiley, New York, 1965).
5. Alpern, S. in *Search Theory* 223–230 (Springer, 2013).
6. Anderson, E. J. & Weber, R. The rendezvous problem on discrete locations. *Journal of Applied Probability* **27,** 839–851 (1990).
7. Weber, R. Optimal symmetric rendezvous search on three locations. *Mathematics of Operations Research* **37,** 111–122 (2012).
8. Weber, R. The Anderson-Weber strategy is not optimal for symmetric rendezvous search on K_4. *arXiv:0912.0670* (2009).
9. Gagniuc, P. A. *Markov Chains: From Theory to Implementation and Experimentation* (John Wiley & Sons, 2017).
10. Powell, M. J. A view of algorithms for optimization without derivatives. *Mathematics Today-Bulletin of the Institute of Mathematics and its Applications* **43,** 170–174 (2007).
11. Sharpe, W. *Investments* ISBN: 9780135046050. https://books.google.com/books?id=4UgWAQAAMAAJ (Prentice-Hall, 1978).
12. Spall, J. C. *et al.* Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control* **37,** 332–341 (1992).
13. Spall, J. C. An overview of the simultaneous perturbation method for efficient optimization. *Johns Hopkins APL Technical Digest* **19,** 482–492 (1998).
14. Rojas, R. in *Neural Networks* 149–182 (Springer, 1996).
15. Fabian, V. in *Optimizing Methods in Statistics* 439–470 (Elsevier, 1971).
16. Spall, J. C. & Cristion, J. A. Model-free control of nonlinear stochastic systems with discrete-time measurements. *IEEE Transactions on Automatic Control* **43,** 1198–1210 (1998).
17. Blum, J. R. Approximation methods which converge with probability one. *The Annals of Mathematical Statistics,* 382–386 (1954).

18. Blum, J. R. Multidimensional stochastic approximation methods. *The Annals of Mathematical Statistics,* 737–744 (1954).

19. Spall, J. C. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on Aerospace and Electronic Systems* **34,** 817–823 (1998).

20. Chin, D. C. Comparative study of stochastic algorithms for system optimization based on gradient approximations. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **27,** 244–249 (1997).

Appendix I

Python Code for Simulation Study

In this appendix, we attach the Python code that is used for simulation studies. The description is provided in Section 3.2.3.

A. k-Markovian Model

```python
import numpy as np
import math
import statistics
import random
import scipy.stats as st
import pandas as pd
import tool_box
from tool_box import get_prob, next_step_decision, generate_trajectory,
    check_trajectory

# in this version, the cafes are labelled by the order of visiting
# the first one you visit is labelled as one, the second distinct one you visited
    is labelled as two, and so on


# to store the model parameters
class k_markov_model:
    # initialize the parameters in kmm
    def __init__(self, k_step, n_cafe, random_init = False, ci_width = 0.05):
        self.k_step = k_step
        self.n_cafe = n_cafe
        self.random_init = random_init
        self.ci_width = ci_width
        # store all parameters in a list
        self.params = list()
        # i represents the level of exploration (number of distinct cafes you have
            visited)
        for i in range(self.n_cafe):
            # your visiting history includes no more than k distinct cafes, the
                length of visiting history might be <= k
            if i+1 <= self.k_step:
```

```
            dimensions = [tuple([i+1 for t in range(j)]+[min(i+2, self.n_cafe)])
                 for j in range(i+1,self.k_step+1)]
        # your visiting history includes more than k distinct cafes, so the
            length of visiting history > k
        else:
            dimensions = [tuple([i+1 for t in range(self.k_step)]+[self.n_cafe])
                ]
        # the parameters in this level of exploration
        params_level = list()
        for dim in dimensions:
            if self.random_init:
                params_level.append(np.random.uniform(-1,1,dim))
            else:
                params_level.append(np.ones(dim))
        self.params.append(params_level.copy())
    self.init_params = self.params.copy()
    self.best_param = None
    self.best_CI = None

# given a trajectory, extract the probability distribution over all cafes
# give equal probabilities to unvisited cafes
# long_traj means that this trajectory is a sample long_trajectory, level is
    specified
def extract_prob(self, trajectory, use_best = False, long_traj = False,
    level_long_traj = 1):
    distinct_cafes = set(trajectory)
    # level of exploration
    if not long_traj:
        level = len(distinct_cafes)
    else:
        level = level_long_traj
    # use full trajectory to determine
    if len(trajectory) <= self.k_step:
        if not use_best:
            use_param = self.params[level-1][len(trajectory)-level].copy()
        else:
            use_param = self.best_param[level-1][len(trajectory)-level].copy()
        for t in trajectory:
            use_param = use_param[t]
    # use partial trajectory to determine
    else:
        if not use_best:
            use_param = self.params[level-1][-1].copy()
        else:
            use_param = self.best_param[level-1][-1].copy()
        for t in trajectory[-self.k_step:]:
            use_param = use_param[t]
    if level <= self.n_cafe - 2:
        prob = get_prob(use_param)
        n_unvisit = self.n_cafe - level
        unvisited_prob = [prob[-1]/n_unvisit for i in range(n_unvisit)]
        return prob[:-1] + unvisited_prob
    else:
        prob = get_prob(use_param)
        return prob

# to compute a sample rendezvous time
def sample_run(self, use_best = False):
    # the labellings of each agent. {agent's idx: real idx}
    agent1 = {i: None for i in range(self.n_cafe)}
    agent2 = {i: None for i in range(self.n_cafe)}
    # first day for each agent
    init1 = np.random.randint(self.n_cafe)
    init2 = np.random.randint(self.n_cafe)
    agent1[0] = init1
    agent2[0] = init2
    actual_visited1 = [init1]
    actual_visited2 = [init2]
    if init1 == init2:
        return 1
    all_cafes = set(list([i for i in range(self.n_cafe)]))
    # the trajectory is indexed by their OWN labelling
    trajectory1 = [0]
    trajectory2 = [0]
    timer = 1
    while agent1[trajectory1[-1]] != agent2[trajectory2[-1]]:
        prob1 = self.extract_prob(trajectory1, use_best)
        next1 = next_step_decision(prob1)
        # to go to an unvisited one
        label1 = len(set(trajectory1))
        if next1 >= label1:
            unvisited1 = list(all_cafes - set(actual_visited1))
            # randomly pick a cafe from those that are unvisited (actual cafe)
            next1_actual = random.choice(unvisited1)
            # update agent1's labelling
            agent1[label1] = next1_actual
            trajectory1.append(label1)
            actual_visited1.append(next1_actual)
        # go to a cafe that is already visited
        else:
            trajectory1.append(next1)

        # now, do the same for agent2
        prob2 = self.extract_prob(trajectory2, use_best)
        next2 = next_step_decision(prob2)
        # to go to an unvisited one
        label2 = len(set(trajectory2))
        if next2 >= label2:
            unvisited2 = list(all_cafes - set(actual_visited2))
            # randomly pick a cafe from those that are unvisited (actual cafe)
            next2_actual = random.choice(unvisited2)
            # update agent1's labelling
            agent2[label2] = next2_actual
            trajectory2.append(label2)
            actual_visited2.append(next2_actual)
        # go to a cafe that is already visited
        else:
            trajectory2.append(next2)
        # update timer
        timer += 1
    return timer

# repeat the sample run and obtain the average rendezvous time
def loss(self, use_best = False):
    sample_rendezvous = list()
    counter = 0
    conf_interval = (-np.infty, np.infty)
    while conf_interval[1]-conf_interval[0] > self.ci_width:
        sample_rendezvous.append(self.sample_run(use_best))
        if counter % 100 == 0 and counter > 0:
            conf_interval = st.t.interval(alpha=0.95, df=len(sample_rendezvous)
                -1,
                                          loc=np.mean(sample_rendezvous),
                                          scale=st.sem(sample_rendezvous))
        counter += 1
    # print(counter)
    sample_rendezvous = np.array(sample_rendezvous)
    return np.mean(sample_rendezvous), conf_interval

# reset the kmm simulator
def reset(self):
    self.params = self.init_params.copy()

# compute the transition probability based on the best param
def transition_prob(self):
    df = pd.DataFrame(columns = ['n_explored', 'partial', 'trajectory'] + [str(
        i+1) for i in range(self.n_cafe)])
    # iterate all level of exploration
    for n in range(self.n_cafe):
        n_explored = n+1
        # iterate all trajectory length
        for t in range(self.k_step):
            len_trajectory = t+1
            # length of trajectory is at least the number of visited cafes
            if len_trajectory >= n_explored:
                # generate all trajectories
                all_traj = generate_trajectory(n_explored, len_trajectory)
                for traj in all_traj:
                    # if n_explored <= n_cafe, need to check if this trajectory
                        is valid
                    if n_explored <= self.n_cafe and check_trajectory(traj,
                        n_explored):
                        transition_prob = self.extract_prob(traj, use_best = True
                            )
                        new_row = dict()
                        new_row['n_explored'] = n_explored
                        new_row['partial'] = False
                        new_row['trajectory'] = tuple([c + 1 for c in traj])
                        for i in range(self.n_cafe):
                            new_row[str(i+1)] = transition_prob[i]
                        df = df.append(new_row, ignore_index=True)
    long_trajectories = generate_trajectory(self.n_cafe, self.k_step)
```

```
        for traj in long_trajectories:
            # no need to check if this is a valid trajectory. Beyond k_step,
                everything is possible
            # in traj, count from 0, but for level, count from 1, so add 1
            level = max(traj) + 1
            if level == self.n_cafe:
                # add one more element to traj so that it uses the correct parameter
                new_row = dict()
                new_row['n_explored'] = level
                new_row['trajectory'] = tuple([c+1 for c in traj])
                augmented_traj = ([0] + traj).copy()
                transition_prob = self.extract_prob(traj, use_best = True, long_traj
                    =True, level_long_traj = level)
                new_row['partial'] = True
                for i in range(self.n_cafe):
                    new_row[str(i+1)] = transition_prob[i]
                df = df.append(new_row, ignore_index=True)
            else:
                last_k_step = tuple([c+1 for c in traj])
                original_traj = traj.copy()
                for l in range(level, self.n_cafe+1):
                    new_row = dict()
                    new_row['n_explored'] = l
                    new_row['trajectory'] = last_k_step
                    augmented_traj = ([i for i in range(l)] + original_traj).copy()
                    transition_prob = self.extract_prob(augmented_traj, use_best =
                        True, long_traj=True, level_long_traj=l)
                    new_row['partial'] = True
                    for i in range(self.n_cafe):
                        new_row[str(i+1)] = transition_prob[i]
                    df = df.append(new_row, ignore_index=True)

        df = df.drop_duplicates()
        return df
```

B. SPSA

```
import numpy as np
import simulator3
from simulator3 import k_markov_model
import math
import random
import statistics

# general spsa optimizer for k_markov_model
class spsa_optimizer:
    def __init__(self, max_itr = 200, perturbation_scale = 1, alpha = 0.602, gamma
         = 0.101):
        self.max_itr = max_itr
        self.perturbation_scale = perturbation_scale
        self.alpha = alpha
        self.gamma = gamma
        self.A = 0.1 * max_itr


    # do a perturbation and evaluate the loss
    # pass in the kmm_model
    def perturbation_eval(self, kmm_model):
        # params is a list of list of numpy arrays
        # create two new_params, one add perturbation, one minus perturbation
        new_params_1 = list()
        new_params_2 = list()
        c_k = self.c/((self.itr+1)**self.gamma)
        perturb = list()
        for i in range(len(kmm_model.params)):
            # params for a certain level of exploration (number of distinct cafes
                that have been visited)
            new_params_1.append(list())
            new_params_2.append(list())
            perturb.append(list())
            for j in range(len(kmm_model.params[i])):
                # generate perturbation array
                dim = kmm_model.params[i][j].shape
                perturbation = np.random.uniform(0, 1, dim)
                perturbation[perturbation<=0.5] = -self.perturbation_scale
                perturbation[perturbation>0.5] = self.perturbation_scale
                new_params_1[i].append(kmm_model.params[i][j].copy() + c_k *
                    perturbation.copy())
                new_params_2[i].append(kmm_model.params[i][j].copy() - c_k *
                    perturbation.copy())
                perturb[i].append(perturbation.copy())
        new_model_1 = k_markov_model(kmm_model.k_step, kmm_model.n_cafe)
        new_model_2 = k_markov_model(kmm_model.k_step, kmm_model.n_cafe)

        new_model_1.params = new_params_1.copy()
        new_model_2.params = new_params_2.copy()

        new_loss_1, ci1 = new_model_1.loss()
        new_loss_2, ci2 = new_model_2.loss()
        del new_model_1, new_model_2
        return new_loss_1, new_loss_2, ci1, ci2, perturb, c_k, new_params_1,
            new_params_2

    # pass in the model.params into this function
    # set additional as True if you want additional iterations
    def optimize(self, kmm_model, a = 0.1, c = 0.07, additional=False,
        additional_itr=100):
        self.a = 0.1
        self.c = 0.07
        if additional == False:
            self.itr = 0
            current_loss, conf_interval = kmm_model.loss()
            self.least_loss = current_loss
            kmm_model.best_param = kmm_model.params.copy()
            self.loss_history = [(current_loss, conf_interval)]
            n_itr = self.max_itr
            self.itr_up_to_now = 0
        else:
            n_itr = additional_itr
        itr = 0
        while itr <= n_itr:
            self.itr_up_to_now += 1
            if additional == False:
                self.itr = itr
            else:
                self.itr = self.itr_up_to_now + itr
            if itr % (int(n_itr/10)) == 0:
                print(str(itr/(int(n_itr/10))*10) + "␣percent␣done")

            new_loss_1, new_loss_2, ci1, ci2, perturb, c_k, new_params_1,
                new_params_2 = self.perturbation_eval(kmm_model)
            if new_loss_1 < self.least_loss:
                self.least_loss = new_loss_1
                kmm_model.best_CI = ci1
                kmm_model.best_param = new_params_1.copy()

            if new_loss_2 < self.least_loss:
                self.least_loss = new_loss_2
                kmm_model.best_CI = ci2
                kmm_model.best_param = new_params_2.copy()

            del new_params_1, new_params_2

            multiple = (new_loss_1 - new_loss_2)/(2*c_k)
            gradient_est = list()

            for i in range(len(perturb)):
                gradient_est.append(list())
                for j in range(len(perturb[i])):
                    grad = multiple * np.reciprocal(perturb[i][j].copy())
                    gradient_est[i].append(grad.copy())

            a_k = self.a/((self.A+self.itr+1)**self.alpha)
            self.update_params(kmm_model, a_k, gradient_est)
            current_loss, conf_interval = kmm_model.loss()
            if current_loss < self.least_loss:
                self.least_loss = current_loss
                kmm_model.best_CI = conf_interval
                kmm_model.best_param = kmm_model.params
            self.loss_history.append(kmm_model.loss())

            itr += 1

    def update_params(self, model, multiplier, adder):
        for i in range(len(adder)):
            for j in range(len(adder[i])):
                model.params[i][j] = model.params[i][j] - multiplier * adder[i][j].
                    copy()
```

C. Other Functions

```
# The other functions are stored in tool_box.py
import numpy as np
import math
import statistics
import random
import scipy.stats as st
import pandas as pd

# given a probability distribution for the next step
# return the decision for the next step
def next_step_decision(prob_distribution):
    decision = np.random.uniform(0,1)
    summation = 0
    for i,p in enumerate(prob_distribution):
        summation += p
        if summation >= decision:
            return i

# extract the transition probability from a dataframe of transition rules
# in trajectory, count from 0
def get_prob_from_df(df, trajectory, k_step):
    reindex_trajectory = [i+1 for i in trajectory]
    n_explored = len(set(reindex_trajectory))
    if len(reindex_trajectory) > k_step:
        partial = True
        traj = tuple(reindex_trajectory[-k_step:])
    else:
        partial = False
        traj = tuple(reindex_trajectory)
    try:
        return df[(df['n_explored']==n_explored) & (df['partial']==partial) & (df['
            trajectory']==str(traj))].iloc[0,3:].to_numpy()
    except:
        print(traj)

# given a table of transition rules, compute a sample rendezvous time
def sample_rendezvous(df, n_cafe, k_step):
    # the labellings for each agent. {agent's idx: real idx}
    agent1 = {i: None for i in range(n_cafe)}
    agent2 = {i: None for i in range(n_cafe)}
    # first day for each agent
    init1 = np.random.randint(n_cafe)
    init2 = np.random.randint(n_cafe)
    agent1[0] = init1
    agent2[0] = init2
    actual_visited1 = [init1]
    actual_visited2 = [init2]
    if init1 == init2:
        return 1
    else:
        all_cafes = set(list([i for i in range(n_cafe)]))
        # the trajectory is indexed by their OWN labelling
        trajectory1 = [0]
        trajectory2 = [0]
        timer = 1
        while agent1[trajectory1[-1]] != agent2[trajectory2[-1]]:
            prob1 = get_prob_from_df(df, trajectory1, k_step)
            next1 = next_step_decision(prob1)
            # to go to an unvisited one
            label1 = len(set(trajectory1))
            if next1 >= label1:
                unvisited1 = list(all_cafes - set(actual_visited1))
                # randomly pick a cafe from those that are unvisited (actual cafe)
                next1_actual = random.choice(unvisited1)
                # update agent1's labelling
                agent1[label1] = next1_actual
                trajectory1.append(label1)
                actual_visited1.append(next1_actual)
            # go to a cafe that is already visited
            else:
                trajectory1.append(next1)

            prob2 = get_prob_from_df(df, trajectory2, k_step)
            next2 = next_step_decision(prob2)
            # to go to an unvisited one
            label2 = len(set(trajectory2))
            if next2 >= label2:
                unvisited2 = list(all_cafes - set(actual_visited2))
                # randomly pick a cafe from those that are unvisited (actual cafe)
                next2_actual = random.choice(unvisited2)
                # update agent1's labelling
                agent2[label2] = next2_actual
                trajectory2.append(label2)
                actual_visited2.append(next2_actual)
            # go to a cafe that is already visited
            else:
                trajectory2.append(next2)
            timer += 1
    return timer

# check the expected rendezvous time based on a dataframe of transition rules
def rendezvous_exp_check(df, n_cafe, k_step, ci_width = 0.05):
    sample_rendezvous_time = list()
    counter = 0
    conf_interval = (-np.infty, np.infty)
    while conf_interval[1]-conf_interval[0] > ci_width:
        sample_rendezvous_time.append(sample_rendezvous(df, n_cafe, k_step))
        if counter % 100 == 0 and counter > 0:
            conf_interval = st.t.interval(alpha = 0.95, df = len(
                sample_rendezvous_time)-1,
                                          loc = np.mean(sample_rendezvous_time),
                                          scale = st.sem(sample_rendezvous_time))
        counter += 1
    # print(counter)
    sample_rendezvous_time = np.array(sample_rendezvous_time)
    return np.mean(sample_rendezvous_time), conf_interval
```

```
# get the probability distribution given the extracted parameters
# first do tanh to each element, then compute by softmax
def get_prob(param):
    processed = [math.exp(math.tanh(p)) for p in param]
    denominator = sum(processed)
    return [p/denominator for p in processed]

# generate all possible trajectories by recursion
# count from 0
def generate_trajectory(n_explored, len_trajectory):
    # base case
    if len_trajectory == 1:
        return [[i] for i in range(n_explored)]
    # recursive relation
    else:
        last_result = generate_trajectory(n_explored, len_trajectory-1)
        new_result = list()
        for t in last_result:
            for i in range(n_explored):
                new_t = (t + [i]).copy()
                new_result.append(new_t)
        return new_result

# check if a trajectory is valid
def check_trajectory(trajectory, n_explored):
    # visited n_explored number of distinct cafes
    if len(set(trajectory)) != n_explored:
        return False
    # check cases like: you visit cafe 2 before cafe 1
    current_max = -1
    for c in trajectory:
        if c > current_max + 1:
            return False
        if c == current_max + 1:
            current_max = c
    return True
```

Curriculum Vitae # Cheng (James) Peng **May, 2022**

LinkedIn: www.linkedin.com/in/jc-peng **Email**: cpeng25@jhu.edu
GitHub: www.github.com/jamespengcheng

EDUCATION

Johns Hopkins University | Baltimore, MD 1/21-5/22
Master of Science in Engineering (Focus: Operations Research and Optimization)

University of Illinois at Urbana-Champaign | Champaign, IL 1/19-5/19
Exchange Student in Data Science

The University of Hong Kong | Hong Kong 9/16-5/20
Bachelor of Science (Major: Decision Analytics; Minors: Mathematics & Finance)

RESEARCH EXPERIENCE

Approximation to optimal strategy in Mozart-Café problem 5/21-5/22
Johns Hopkins University | Supervisor: Prof. John C. Wierman

- Designed novel k-Markovian parameterization technique as generic representation for any symmetric rendezvous search strategy in Mozart-Café problem
- Utilized Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm on parameter optimization in k-Markovian modelling and reproduced optimal strategies in small size cases

Classical and model-free controllers in nonlinear problems 5/21-5/22
Johns Hopkins University | Supervisor: Prof. James C. Spall

- Implemented and evaluated powerful model-free controller and PID controller in driftless affine nonlinear discrete time systems by Python
- Proved sufficient conditions for such systems to be uncontrollable by PID, but controllable by model-free controller

Reinforcement learning in sports scheduling 1/21-10/21
Johns Hopkins University | Supervisor: Dr. Antwan D. Clark

- Developed custom reinforcement learning environment for round-robin tournament scheduling problems by OpenAI Gym
- Trained reinforcement learning agents by ACKTR algorithm and successfully generated sample schedules with least number of breaks in single round-robin cases

A New Effective InsurTech Tool: CIBer 6/20-10/21
The Chinese University of Hong Kong | Supervisors: Profs. Sheung Chi Phillip Yam and Ka Chun Cheung

- Designed and developed supervised learning model named Comonotone-Independence Bayesian Classifier (CIBer)
- Compared CIBer with classical supervised learning models including CART, MLP, SVM on several insurance-related datasets, obtaining out-of-sample accuracy over 10% higher than any other model and AUC values over 0.08 higher than any other model on every dataset

PUBLICATIONS

- Chen, Y., Cheung, K. C., Fan, N. S., **Peng, J. C.**, and Yam, S. C. P. (2021). A New Simple Effective InsurTech Tool: Comonotone-Independence Bayes classifier (CIBer). The 11th International Conference of the Financial Engineering and Banking Society (Accepted).
- **Peng, J. C.**, Clark, A. D., and Dahbura, A. (2021, December). Introducing Human Corrective Multi-Team SRR Sports Scheduling via Reinforcement Learning. 2021 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, 2021.
- Li, Q., **Peng, J. C.**, Mohan, D., Lake, B., Ruiz, A., Weir, B., Kan, L., Yang, C., Labrique, A. (2022). Effectiveness of "It's Up To You" Campaign in Promoting COVID-19 Vaccine Uptake: a spatiotemporal analysis of 3,104 counties in the United States. Submitted to Journal of Medical Internet Research.