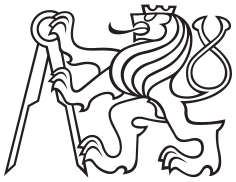
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Data-Driven Sequential Dynamic Pricing in Mobility

Ondřej Stejskal

Supervisor: Ing. Jan Mrkos
Field of study: Open Informatics
Subfield: Artificial Intelligence and Computer Science
May 2021

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Stejskal Ondřej** Personal ID number: **483739**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Data-Driven Sequential Dynamic Pricing in Mobility

Bachelor's thesis title in Czech:

Datově-řízená sekvenční dynamická cenotvorba v dopravě

Guidelines:

The goal of this project is to improve methods for solving sequential, resource-constrained dynamic pricing problems in mobility by incorporating historical data in the pricing decision process. To this end, student is expected to satisfy the following objectives:
1) Prepare a dataset of historical pricing and occupancy data from multiple transportation providers. Clean the dataset and analyze it. Propose features based on this dataset.
2) Survey solution techniques applicable to the pricing problems formulated as a Markov Decision Process. Focus on the heuristic solutions, especially Monte Carlo tree search algorithms.
3) Based on the literature survey and dataset analysis, propose a solution technique for the ticket pricing problem.
4) Propose evaluation metrics and evaluate the proposed method either on testing data and/or against benchmark solutions.

Bibliography / sources:

[1] Russell, Stuart J. and Norvig, Peter - Artificial Intelligence: A Modern Approach (2nd Edition) – 2002
[2] Browne et al. - A Survey of Monte Carlo Tree Search Methods - 2012
[3] Couëtoux et al. - Continuous Upper Confidence Trees – 2011
[4] Cazenave et al. - Playout Policy Adaptation for Games - 2015

Name and workplace of bachelor's thesis supervisor:

Ing. Jan Mrkos, Artificial Intelligence Center, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **08.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. Jan Mrkos	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. | Date of assignment receipt

Student's signature

Acknowledgements

I would like to thank my supervisor Ing. Jan Mrkos for the guidance, patient support, and expertise which was invaluable.

I also would like to thank my family for the patience they have and the support they gave me.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 21. May 2021

Abstract

This thesis deals with the problem of sequential dynamic pricing in mobility. It focuses on the domains with a limited number of resources, where the customers are buying products constructed from these resources. When we set a price for any product, we need to consider the possible revenue we might get from the resources that are included in the product if we did not sell that product. The problem is inspired by dynamic pricing for the long-haul bus routes with multiple stations. We model the problem using Markov Decision Process, and we use Monte Carlo Tree Search to solve the problem.

We use domain knowledge to improve the standard Monte Carlo Tree Search algorithm. In this thesis, we propose heuristics that evaluate the states and replace the random rollout.

We show that these heuristics improve the results when we have limited computational resources, or we solve large space problems.

Keywords: dynamic pricing, monte carlo tree search, markov decision process, value estimation, rollout

Supervisor: Ing. Jan Mrkos

Abstrakt

Tato bakalářská práce se zabývá problémem sekvenční dynamické cenotvorby v dopravě. Konkrétně na případy cenotvorby, kde je limitovaný počet zdrojů a zákazníci kupují produkty složené z těchto zdrojů. Když volíme cenu produktu, potřebujeme zohlednit případné výnosy ze zdrojů, které by nám zůstali, pokud bychom daný produkt neprodali. Tento problém je inspirován dynamickou cenotvorbou pro dálkové autobusy s více stanicemi. Tento problém modelujeme pomocí Markovského rozhodovacího procesu a používáme Monte Carlo metodu stromového prohledávání pro řešení problému.

Pro vylepšení Monte Carlo metody stromového prohledávání použijeme znalost domény, za účelem vytvoření heuristiky, které následně hodnotí stavy našeho problému v Markovském rozhodovacím procesu. Tyto heuristiky nahradí náhodný rollout.

Ukážeme, že tyto heuristiky zlepší výsledky, když máme omezenou výpočetní kapacitu, nebo máme rozsáhlý problém.

Klíčová slova: dynamická cenotvorba, monte carlo metoda stromového prohledávání, markovův rozhodovací proces, odhad hodnoty, rollout

Překlad názvu: Datově-řízená sekvenční dynamická cenotvorba v dopravě

Contents

1 Introduction 1

1.1 Problem Overview 1

1.1.1 Solving the RPDSP Problem . 2

1.2 Outline 3

2 Related Work 5

2.1 Dynamic Pricing 5

2.2 MDP Formulation 6

2.3 Monte Carlo Tree Search 6

3 Theory 9

3.1 MDP 9

3.1.1 Definition 9

3.1.2 Policy 10

3.1.3 Optimal Policy 10

3.1.4 MDP Solvers 11

3.2 Monte Carlo Tree Search 13

3.2.1 Upper Confidence Bound Applied to Trees 15

3.2.2 Rollouts and Value Estimation 17

4 Problem Definition 19

4.1 RPSDP Problem 19

4.2 MDP definition 20

4.2.1 State Space - $\mathcal{S}$ 21

4.2.2 Action Space - $\mathcal{A}$ 21

4.2.3 Transition Function - T 22

4.2.4 Reward Function - $\mathcal{R}$ 22

5 Implementation 23

5.1 Value Estimation Methods 23

5.1.1 Static Evaluation Heuristics . 24

5.1.2 Rollout Value Estimation Methods 25

5.1.3 Linear Programming Heuristics 26

6 Experiments 29

6.1 Problem Settings 29

6.2 Parameters Tuning 31

6.2.1 UCT Parameters 31

6.2.2 Static Evaluation Heuristics Parameters 31

6.3 Results 33

7 Conclusion 37

Bibliography 39

Figures

3.1 Tree representing pricing problem 13

3.2 Four phases of MCTS 14

4.1 Illustration of one step in our simulation. 21

6.1 Revenue results for static heuristics and for static heuristics combined with the rollouts. 33

6.2 Revenue results for evaluation methods. 35

6.3 Revenue results for different number of iterations for Random Rollout Heuristic and Capacity Heuristic. 36

Tables

6.1 Parameters for the bus pricing problem. 30

6.2 Dataset features. 30

6.3 Used parameters for the MCTS. 31

6.4 Different parameters for static heuristics. 32

6.5 Mean runtime for methods ordered by the revenue results. 34

Chapter 1

Introduction

Dynamic pricing is a part of revenue management that examines product pricing strategies maximizing certain objectives, usually revenue. Dynamic pricing changes the prices of products throughout the time depending on various factors, for example, demand, products supply, purchase time, etc. Dynamic pricing is used in many domains such as retail, transportation, electricity supply, and hotel industry.

We are interested in the usage of dynamic pricing in mobility, more specifically, dynamic pricing for long-distance bus routes. However, the longest tradition and frequent usage has dynamic pricing in airline revenue management [16]. The bus and train carriers have also adopted dynamic pricing strategies to maximize revenues or to optimize the utilization of buses and trains [10].

One of the most significant differences between airline routes and bus routes is that bus routes often have a set of intermediate stations between the original and final stations. This means that we are not only pricing one product, but we are pricing multiple products that often overlap, in our case, sharing the seat between two stations. For example, consider a bus route from A to D with intermediate stations B and C, there the ticket from A to C shares a part with a ticket from B to D. That fact gives an additional aspect of the shadow price. And we are confronted with the question, what reward from other products we would have if we did not sell this product.

1.1 Problem Overview

It is helpful to describe the domain with the terminology we use. The resource is a seat on the bus between two stations that has no intermediate station between them. The product is a ticket for a seat on the bus between arbitrary

stations, so it is a combination of resources.

Resources have limited supply given by the capacity of the bus, so that we will call the products resource-constrained. The resources are also perishable, which means that we can only sell them until a specific time, in our case, until the departure of the bus.

Nowadays, customers buy tickets mainly online via mobile applications or web browser applications, which means that the customer can buy the product anytime before the bus's departure.We consider sequential pricing, meaning that the requests for products come up throughout the time, and we respond to that requests by setting the price.

The objective is usually to maximize the revenue, and we will aim to maximize the revenue. Still, other objectives can be reached, for example, maximizing the utility. Maximizing utility means maximizing the overall amount of transported passengers.

With all aspects described, we define our problem as a Resource-constrained product sequential dynamic pricing problem (RPDSP problem). For more detailed information and formal definition, please refer to Section 4.1.

1.1.1 Solving the RPDSP Problem

To solve the RPDSP problem, we use the Markov Decision Process (MDP) as a framework to model the RPDSP problem. MDP is a common framework for solving sequential decision-making under uncertainty. The uncertainty is given by the arrival of requests for products that we do not know beforehand, and we do not know what price is the customer willing to pay for the product.

Finding the optimal solution to MDP is a well-examined problem. There are solutions based on dynamic programming, reinforcement learning, or mathematical optimization. These approaches have their drawbacks, such as low scalability or add additional constraints to the definition of a problem.

With the advent of Monte Carlo Tree Search (MCTS) algorithms and their success on large MDPs, we will use this heuristic search algorithm to approximate the optimal solution for the MDP.

We will try to improve the most popular algorithm in the Monte Carlo family, the Upper Confidence Bound applied to trees (UCT), by adding domain knowledge to that algorithm. For more information on MCTS and UCT, please refer to Section 3.2.

We incorporate the domain knowledge by crafting heuristic functions to evaluate the states. The state represents the RPDSP problem at one given

timestep. The state contains information about free resources and requested product at the given timestep. By adding domain knowledge, we try to push more accurate estimates to the UCT, and by that, we try to achieve better results or make the algorithm faster while preserving the results.

1.2 Outline

In Chapter 2, we examine related works, firstly revenue management, especially dynamic pricing. Then we look into different MDP formulations for dynamic pricing problems and see related work for Monte Carlo Methods, especially UCT. Chapter 3 describes the theoretical background, mainly the MDP framework and Monte Carlo Tree Search. In Chapter 4, we formally define the RPDSP problem, and we describe our MDP definition. In Chapter 5, we present our heuristics and how we implemented the problem. In Chapter 6, we benchmark the results of the heuristics. Chapter 7 concludes our findings.

Chapter 2

Related Work

Dynamic pricing has become a common practice in today's world. There are many different approaches to the problem of dynamic pricing. In this section, we first describe the possible models for dynamic pricing problems known in the literature. Next, we discuss the literature considering MDP as a framework for dynamic pricing. Lastly, we analyze MCTS as a solution method for the MDP. We also investigate how domain knowledge can improve MCTS methods.

2.1 Dynamic Pricing

Dynamic pricing is studied in the field of revenue management, also called yield management.

Revenue management studies modeling of demand distributions, arrival processes, capacity control, and pricing [16]. One of the main subjects of research in revenue management is airline revenue management. Airline revenue management deals with constrained resources and finite horizon selling periods, sharing these properties with the bus problem we discuss in this thesis.

Our problem has several similar properties as network revenue management introduced by Gallego and Ryzin [11]. The network revenue management considers multiple products and adjusts the price of each product. The literature more accurately classifies this problem as a capacity control problem.

The capacity control problem and dynamic pricing mentioned before do not fully correspond to our problem because we need to find the optimal price for each time, not just statically set the price to one value. Moreover, in our case, the products are a combination of resources.

A more similar definition to our problem provides Liu and Ryzin in their work [19] where they consider a flight network. A flight network is a graph of multiple flight routes, where you can buy products consisting of multiple flights. For each timestep, they decide what price to offer for each product. This problem has been solved using dynamic programming techniques [21].

2.2 MDP Formulation

Sequential dynamic pricing is commonly modeled as MDP. For example, in the problem of overbooking and seat inventory control [20], dynamic pricing of on-demand services [12], or retail pricing [5].

Game theory has some similar aspects as MDP, and the problem of dynamic pricing where there is a non-monopolistic environment can also be described as a game [6].

The common techniques for solving MDPs representing dynamic pricing are dynamic programming [12] or reinforcement learning [20]. These techniques have their drawbacks in scalability or in terms of defining the problem.

Because our dynamic pricing problem has a large state space, we need to find a way to solve large MDPs. The common algorithms for solving the MDP, such as value iteration or policy iteration, do not scale well. These methods are called offline solvers. We will use online solvers, and a well-suited technique for solving MDP online is an MCTS [3].

2.3 Monte Carlo Tree Search

MCTS is a family of heuristic search algorithms. It combines Monte Carlo evaluations with tree search [7].

The MCTS has been widely used in game theory, where it lands spectacular results, for example, in the game of Go [4].

The success of Monte Carlo Tree Search in recent years is connected with a suitable tree policy. Tree policy decides what states are explored next. The most popular algorithm in the MCTS family is Upper Confidence Bound applied to trees (UCT) [14]. UCT uses the UCB1 tree policy based on the multi-arm bandit problem [1], solving the exploration-explotation dilemma.

In the base form, MCTS is a domain-independent search algorithm. Domain independence is beneficial as we can use the algorithm in any domain. However,

we can improve the performance of MCTS by incorporating domain knowledge.

Incorporating domain knowledge usually means assigning a value to the state depending on the attributes of the state or ordering the actions based on their quality.

Rollout randomly traverses the space state from the current node until the terminal state. Depending on the domain, returns the value obtained in the terminal state or accumulated in the traversing.

The random rollout is a computationally efficient method, but it can be successfully altered by biasing the moves using a heuristic [13]. The rollout can also be stopped at a certain depth, and the result can be evaluated from that state [2].

Chapter 3

Theory

To solve the problem of dynamic pricing, we formalize the problem as an Markov Decision Process (MDP), then we will use Monte Carlo Tree Search (MCTS) to find the optimal policy. We will use MCTS because classical methods such as value iteration or policy iteration do not scale well with the size of the problem.

3.1 MDP

MDP is a framework for modeling sequential decision-making in stochastic environments. MDP is an extension of Markov chains. MDP is widely used in robotics, economy and manufacturing, and it is a suitable framework for dynamic pricing.

3.1.1 Definition

Markov Decision Process gives a mathematical framework for modeling sequential decision problems with a stochastic outcome of the actions. We will use the Finite Discrete-Time Fully Observable Markov Decision Process (MDP) [15], which is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where:

- $\mathcal{S}$ is the finite set of states
- $\mathcal{A}$ is the finite set of actions available for an agent.
- $\mathcal{T}$ is a transition function giving probability $\mathcal{T}(s, a, s')$ of going to state s' from the state s by performing the action a

- $\mathcal{R}$ is a reward function returning reward value $\mathcal{R}(s, a, s')$ given by performing an action a, which leads to a transition from state s to state s'.

Other necessary components for MDP are the utility function and environment history. Environment history is a sequence of states and actions that is possible for a given MDP. This thesis only mentions the finite horizon MDP, meaning that the environment history will always be finite with a fixed time N. Utility function depends on that history. We can write:

$$U_h([s_0, a_0, s_1, a_1, \ldots, s_N])$$

We will consider the utility function equal to the sum of the rewards we obtain by following the history:

$$U_h = \sum_{i=0}^{N-1} \mathcal{R}(s_i, a_i, s_{i+1})$$

3.1.2 Policy

To find a solution, we need to know what action to choose in every reachable state. It means that we need a mapping from states to actions. That mapping is called a policy π.

$$\pi : \mathcal{S} \to \mathcal{A}$$

In the finite horizon, the action may depend on the timestep. Intuitively, when the simulation is at the end, optimal action might differ from the action at the start. Policies that depend on the time are called nonstationary. Stationary policies do not depend on the time and are used mainly in infinite horizon mdps.

Since we are using finite horizon MDP, we will not consider the discount factor. The discount factor is used to lower the rewards from the more distant future by multiplying the reward of the next action by a $\gamma \in (0, 1)$.

3.1.3 Optimal Policy

To solve MDP, we need to find an optimal policy π^*. An optimal policy is a policy maximizing the expected utility of the possible environment histories generated by that policy [15]. The expected utility of policy starting at the state s is:

$$U^{\pi}(s) = E[\sum_{i=0}^{\infty} \mathcal{R}(S_i, \pi(S_i), S_{i+1})]$$

Where expectation E depends on a probability distribution over state sequences determined by random variable S_i over the state space and π [18]. And the optimal policy is policy maximizing that utility:

$$\pi_s^* = \underset{\pi}{\operatorname{argmax}} U^\pi(s)$$

To solve the MDP, we need to define the utility of the state. The utility of a state is the expected reward for the next transition plus the discounted utility of the next state, assuming that the agent chooses the optimal action. Therefore the utility of the state is given by the following equation:

$$U^\pi(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)[\mathcal{R}(s,a,s') + \gamma U(s')]$$

That equation is called the Bellman equation. In this thesis, we consider only finite horizon MDP's, so we do not consider the discount factor. Thereby we assume $\gamma = 1$. The expected utilities of states are solutions of the set of bellman equations.

3.1.4 MDP Solvers

Offline Solvers

To find an optimal policy, we need to solve a set of Bellman equations. For each state, there is one equation. Due to the nonlinearity of the equations, we can not solve them using linear algebra methods. One approach is to use iterative methods. Value iteration is a standard iterative algorithm. We define $U_i(s)$ as the utility value for the state at the iteration i. The important concept *Bellman update* is defined as follows:

$$U_{i+1}(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)[\mathcal{R}(s,a,s') + \gamma U_i(s')]$$

At the start, the utility values U are initiated as zeros. Then the algorithm repeats the Bellman update for all the states simultaneously until the maximum difference between current and preceding utilities are smaller than some ϵ. The algorithm pseudocode is shown in Algorithm 1. The value iteration asymptotically converges to the optimal solution [18].

Value iteration and another standard method, policy iteration, are offline algorithms. They construct a complete policy before the simulation is started. A disadvantage of offline methods is that they do not scale well with the size of an MDP because offline algorithms construct policy for each state, and it takes significant time and memory. Generally, these algorithms are applicable only to small to mid-size domains [17].

Algorithm 1: Value Iteration

Function `Value_Iteration`(mdp, ϵ)**:**
- $U_i(s), U_{i+1}(s) \leftarrow 0$, for all $s \in S$;
- $\delta \leftarrow 0$;
- **while** $\delta < \epsilon$ **do**
  - $U_i(s) \leftarrow U_{i+1}(s)$;
  - $\delta \leftarrow 0$;
  - **foreach** $s \in S$ **do**
    - $U_{i+1}(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s'} P(s'|s,a)[\mathcal{R}(s,a,s') + \gamma U_i(s')]$;
    - **if** $|U_{i+1}(s) - U_i(s)| > \delta$ **then**
      - $\delta \leftarrow |U_{i+1}(s) - U_i(s)|$;
    - **end**
  - **end**
- **end**
- **return** U_i

Online Solvers

A better-suited approach for larger MDPs is to use online planning algorithms. Online planning algorithm does not construct policy beforehand, but it decides what action to execute in the current state visited by the simulation. The advantage of online planning algorithms is that it considers only the subset of states reachable from the current state.

Generally, the online algorithm is divided into two alternating phases: the planning and execution phases. In the planning phase, the algorithm decides what is the best action in the current state. The planning phase usually builds a tree of reachable states with the root node representing the current state.

There are two types of nodes: state and action. State node represents the state and value of the state. Action node represents action executed from the parent state. These two types alternate, as shown in the 3.1.

The tree is built iteratively, and when a new leaf state node with value is added, the value of nonterminal leaves can be estimated by estimation function or given default value. Then the value is backpropagated to the parent nodes, usually by averaging the values from all leaf nodes. The best action is the child node of a root with the highest value.

When the planning phase ends, the execution phase executes the best action and updates the current state. The tree has finite depth as we consider finite MDPs, but the depth and branching factor are large in problems we encounter in this thesis.

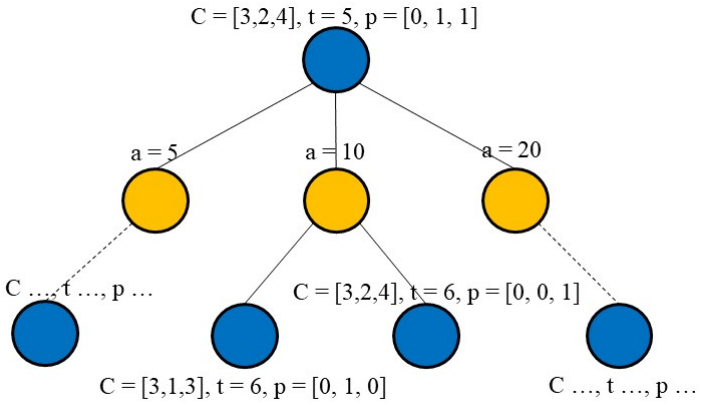


Figure 3.1: This tree represents our pricing problem, described in the Chapter 4. The tree has two types of nodes, state nodes, and action nodes. State nodes are blue, and action nodes are yellow. State nodes contain information about free capacity, time step, and requested product. The children of the state nodes are action nodes, and they are representing what action we can perform. Children of action nodes are state nodes, representing states obtained by different outcomes of executing an action.

Because the problems are large, we can not completely search the state space. We need to use an algorithm that approximates the search in the state space. We will use Monte Carlo Tree Search that is based on sampling from the space, and approximates the optimal decision.

3.2 Monte Carlo Tree Search

This section will describe Monte Carlo Tree Search (MCTS). MCTS finds optimal decisions, in our case MDP actions, by building a search tree based on the sampling from the domain space. MCTS is widely popular in games, and unlike other game tree searches, it does not use a heuristic evaluation function. Instead, the values of states are approximated by simulating moves from the current state and considering the results.

Usually, the nodes in the tree represent states, and directed links represent actions. However, due to the stochastic outcome of an action in MDP, we need to represent the different outcomes of the actions. In our case, there are two types of nodes, action node and state node. The algorithm always starts in the state node, and by executing action, it gets to the action node representing executed action. From that action node, the MDP simulates outcome of an action and yields a state node.

The state node includes information about the MDP model, state and

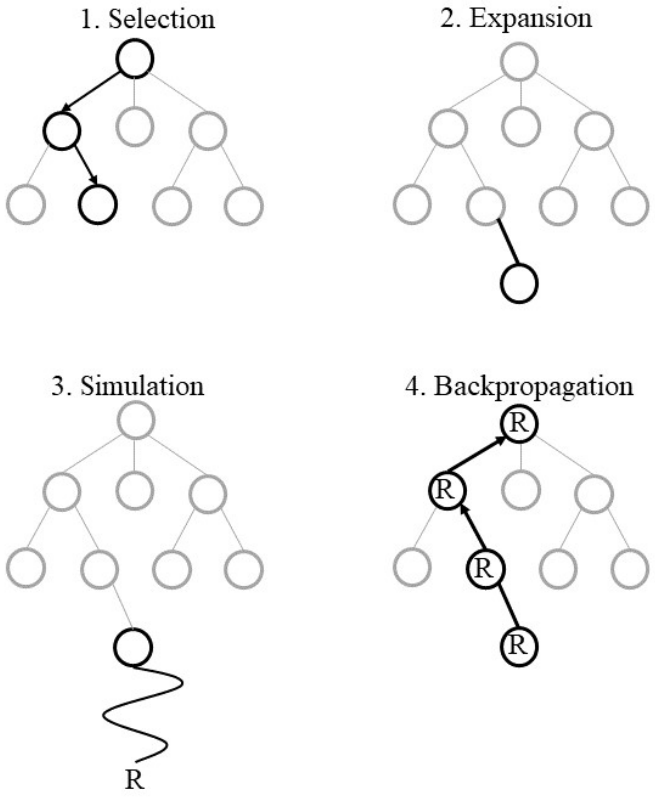


Figure 3.2: Four phases of MCTS

number of visits N, and the action node includes information about an action, number of visits N, and value of the node Q.

MCTS builds a tree from the current state node. MCTS builds the tree iteratively until it exceeds a computational budget or a number of iterations. Each iteration includes the following phases [3]:

- Selection: From the root state node t_0, a child is recursively selected until a state node t_n with a terminal state or expandable child is reached. Child nodes are selected using tree policy. In our case, selecting the child node means selecting a child action node that generates the child state node.
- Expansion: Child node t_c of the node t_n is added to the tree. Selection between the children is based on the tree policy.
- Simulation: The rollout starts from the expanded node t_c. The rollout is a simulation that chooses actions which transition states until a terminal state or a certain depth is reached. From that state, the reward is taken, and it gives value Q to the action node, which leads to node t_c.

- Backpropagation: The rewards from the simulation and transitioning between t_0 and t_c are used to update values Q of the action nodes between the root t_0 and the expanded node t_c.

After the given number of iterations is executed or the time/resource limit is exceeded, the MCTS selects the best child node of the root. That child node is an action node. As a criterion for choosing the best child action node, we use the most visited child.

There are also other methods for selecting actions in the root, such as choosing the child with the highest value, but that child might not be sufficiently explored so that the outcomes would not be that robust. It is important to define a tree policy for selecting the action nodes in the selection phase. We will use Upper Confidence Bound applied to trees (UCT).

3.2.1 Upper Confidence Bound Applied to Trees

The UCT's tree policy aims to approximate the values of actions from the current state. It addresses the exploration-exploitation dilemma and decides whether it is better to explore lesser-explored nodes or exploit nodes with high rewards.

The UCT selects nodes based on the upper confidence bound formula UCB1 [14]. The UCB1 is based on the multi-armed bandit problem. The choice of the child node is treated as a choice, which arms to pull. The UCB1 formula is following:

$$UCB1 = v_i + C_p\sqrt{\frac{\ln N}{n_i}}$$

Where v_i is the mean value of action node i, which we obtain by simulation and backpropagation part of the MCTS algorithm, C_p is an exploration constant, N is a number of visits of the parent state node, and n_i is a number of visits of action node i. The action with the highest UCB1 value is selected. As we can see, the v_i is an exploitation part, and the term with a number of visits is an exploration part.

We show the of the UCT in Algorithm 2. This pseudocode is an UCT for MDPs.

The main difference between this algorithm and UCT that appears in the literature [3] is that we need to take into account the stochastic outcome of the actions. We model this by using two types of nodes: action nodes denoted

Algorithm 2: UCT Search

Function `UCTSearch`(s_0)**:**
- $t_0 \leftarrow$ node(s_0) ;
- **while** *within computional budget* **do**
  - $t_n \leftarrow$Selection(t_0);
  - Expansion(t_n);
  - $R \leftarrow 0$;
  - $R \leftarrow$Simulation($s(t_n), R$);
  - Backpropagation(t_n, R);
- **end**
- **return**;

Function `Selection`(t)**:**
- **while** t *is nonterminal* **do**
  - **if** t *is not in tree* **then**
    - **return** t;
  - **else**
    - $n \leftarrow \underset{n' \in \textit{children of } t}{\operatorname{argmax}} \frac{Q(n')}{N(n')} + c\sqrt{\frac{\ln N(t)}{N(n')}}$;
    - $t \leftarrow$ generate transition to state node from action node n ;
  - **end**
- **end**
- **return** t;

Function `Expansion`(t)**:**
- add t to the tree

Function `Simulation`(s, R)**:**
- **while** s *is nonterminal* **do**
  - choose random $n \in$ child t;
  - $s' \leftarrow$ Generate state from action node n;
  - $R \leftarrow R + \mathcal{R}(s, a(n), s')$;
  - $s \leftarrow s'$
- **end**
- **return** R;

Function `Backup`(t, R)**:**
- **while** t *is not null* **do**
  - n parent of t $Q(n) \leftarrow \frac{Q(t)N(t)+R}{N(t)+1}$;
  - $N(n) \leftarrow N(t) + 1$;
  - $t \leftarrow$ parent of n;
- **end**

n, and state nodes denoted t. They are alternating, meaning that the child of the action node is a state node and vice versa.

The next difference is that when we choose an action node, then the underlying MDP generates from that action node the next state node.

Due to these differences, we need to work with information on what nodes are in the tree, and when there is a new unvisited node, we save that node in the expansion part.

3.2.2 Rollouts and Value Estimation

In the simulation phase, we gain the reward that is then backpropagated to all preceding action nodes. That node value is then used in the selection part in the UCB1 formula.

When we explore a new node in the exploration part, we have to determine the reward that is then backpropagated. It is done by the simulation part.

The standard method for simulation is to use random rollout. It takes random actions until it reaches the terminal state, and the cumulative reward from executing all actions is the reward. That simulation is useful because we do not need to traverse the subtree completely, which is impossible when the depth is large as the time to traverse is exponential with depth, but the rollout is linear with depth.

We can improve the rollout with domain knowledge. With domain knowledge, we can engineer evaluation functions for given states. We can use the evaluation function to replace the rollout, or we can stop the rollout in a given depth, and then we can use the evaluation function. Or the rollout can be informed and use domain knowledge to navigate him in choosing the actions by biasing the possible actions.

We will describe our heuristics in Chapter 5.

Chapter 4

Problem Definition

In this section, we describe sequential dynamic pricing for buses. Then we formalize the problem. Specifically, we consider Resource-Constrained Product Sequential Dynamic Pricing Problem (RPSDP problem). Then we model RPSDP problem as MDP.

We consider bus traveling from the first station to terminal station through a set of stations on the way. We divide the bus trip into segments between neighboring stations. A resource is a seat for one segment.

Every resource has supply limited by the capacity of the bus. The resources are perishable, which means that they have a limited selling period given by departure of the bus from the station, where the resource starts.

The product is a combination of resources. We assume a single monopolistic seller and multiple customers. Customers send requests for products to the seller. The seller offers a price, and if the price is within the buyer's budget, the product is sold.

4.1 RPSDP Problem

Formally the RPSDP problem is specified by a tuple (R, P, c, S_p, B), where R is the set of resources, P is the set of products, c is the initial supply of each resource, S_p is the selling period of each resource, and B is the set of requests.

There is a sequence of n stations on the bus route $(st_1, st_2, \ldots, st_n)$. R is a set of $n-1$ resources r where the r is a ticket for the seat between the adjacent stations st_i and st_{i+1}. The product p is a combination of these resources. We will consider only combinations of adjacent resources that

means that the product is a sequence of k resources $(r_j, r_{j+1}, \ldots, r_{j+k})$.

We discretize the selling period into a sequence of natural numbers $\tau = (1, 2, \ldots, t_{max})$, which are timesteps of equal length.

Each resource has its selling period s_p, which means that we cannot sell this product when the timestep t is higher than s_p. Each resource also has its initial supply c_0. We consider the same value of initial supply for each resource, as it is the number of seats on the bus.

B is the sequence of customer's requests. Each request is composed of the timestep t, demanded product p, and customer's budget b for the product p at the time t.The customer's budget is the maximum price the customer is willing to pay for the product p at the time t.

We can model the arrival of the requests as a sampling from categorical distribution, where every product, including the empty product, has its probability. The sum of the probabilities must be 1. We sample from the distribution at each time step. The model is memory-less, which means that the sampling at each timestep is independent of the previous samplings.

Now we describe our simulation. The simulation models interactions between the seller and customers.

First, we simulate the arrival of the customer's requests for products. If there is no request, we increment the timestep t and simulate the arrival of the customer's requests again. If there is a customer request for a product, we increment the t and decide what price we offer. Then the customer accepts or rejects the price. The customer decides based on his budget. If the customer accepts the price, we sell the product, increase the revenue, decrease the free spaces, and increase the t, and we wait until the next product is requested. If the customer does not accept the price, we increase the t and wait for the next request. We end the simulation after the given number of timesteps is reached. It means that the bus has departed from the penultimate station.

We can see one step of the simulation in Figure 4.1.

Our objective is to maximize the revenue we obtain in the simulation. We get revenue from customers who accepted the price.

4.2 MDP definition

Because we do not know the customer's budget when we offer the price, the outcome of our action (the price offer) is not deterministic. And the problem is sequential, as is the arrival of the request. With that said, the suitable

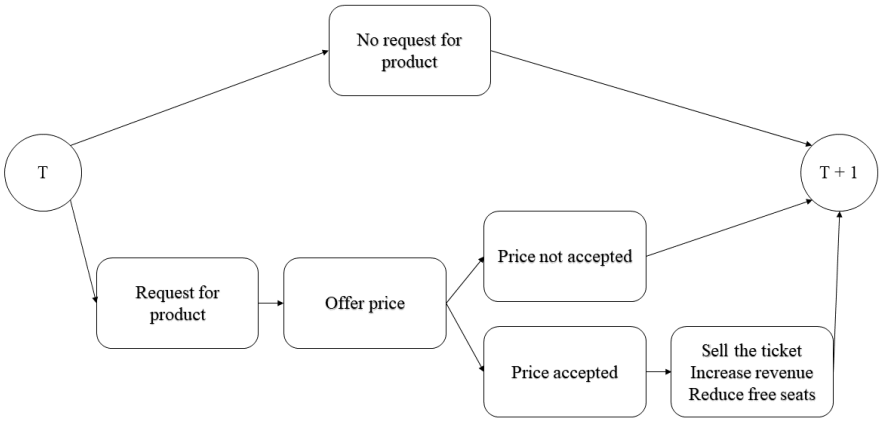


Figure 4.1: Illustration of one step in our simulation.

framework is MDP. In Chapter 3, we defined MDP as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$. In this section, we specify each component of the tuple for RPSDP problem.

4.2.1 State Space - $\mathcal{S}$

State space $\mathcal{S}$ consists of states $s = (c, p, t)$, where c is a supply of all resources. It is a vector consisting of free spaces for all resources. p is the requested product. It is a vector composed of ones and zeros. If in the ith position is one, it means that the product contains resource r_i. Otherwise, it does not include r_i. t is a timestep from the selling period $t \in \tau$.

For example, when we are in the state $([2, 5], [1, 0], 4)$, it means that we have two free spaces between the first and second station and five free spaces between the second and third station, and the requested ticket is between first and second station. And we are in the fourth step of the simulation. In the following text we will use $s_t = (c, p)$ as a equal replacement of $s = (c, p, t)$.

4.2.2 Action Space - $\mathcal{A}$

Action a is the price we offer in the state s. For our simulation, we consider a finite set of actions. It is convenient only to offer a finite set of integer-like prices. Overall, bus companies often offer prices from that set. For example, if $(a = 7)$, it means that we offer price seven for the product requested in the state s.

4.2.3 Transition Function - T

Transition function $\mathcal{T}(s_t, a, s_{t+1})$ is composed of two parts. Firstly, it simulates the customer's budget, and if the budget is higher than the action a, the product p_t is purchased, then the capacity of new state s_{t+1} is the capacity of the original state s_t minus the product demanded, $c_{t+1} = c_t - p_t$. If the product is not purchased then $c_{t+1} = c_t$.

After this acceptance part, the new product p_{t+1} is demanded. The second part decides which product is demanded. It depends on the sample from the categorical distribution of all products, including the empty product. The empty product represents no request for a product in the current timestep. The new state s_{t+1} is (c_{t+1}, p_{t+1}).

4.2.4 Reward Function - $\mathcal{R}$

Reward function $\mathcal{R}$ yields reward equal to the action if the price is accepted and the product is sold in the transition. Otherwise, the reward is equal to 0. We can say that when the new state's capacity is lesser than the original capacity by the original product, then the reward is a. If not reward is zero.

Chapter 5

Implementation

In this chapter, we describe the implementation of the problem. Then we describe methods for value estimation of nodes in MCTS.

We use Julia programing language. To implement MDPs, we use the POMDPs.jl library [9], which is an interface for working with MDPs. As a core UCT algorithm, we use the implementation from the MCTS.jl that is part of the POMDPs.jl library. We use the MDP simulation based on the POMDPS.jl, that is developed by the supervisor of this thesis.

We will implement different methods for the simulation part of the MCTS. The default is random rollout implemented in the MCTS.jl.

We need to set multiple hyperparameters for MCTS. The setting will be described in the Chapter 6. *depth* specifies the maximum depth of the tree built by the MCTS. The *number_of_iterations* specifies how many iterations of building the tree is executed. Exploration constant C_p is a hyperparameter in the UCB1 formula. With higher C_p, the algorithm is more exploratory. The last hyperparameter is max_time. It denotes a maximum time for MCTS to build the tree.

5.1 Value Estimation Methods

The simulation part of the common MCTS performs rollout to estimate the value for the current node. We try to use domain knowledge to craft estimation functions that would perform better than the random rollout.

Both game theory and planning uses the concept of domain knowledge and heuristic value evaluation. Using domain knowledge, algorithms try to find the most accurate estimate of the state. We will try to replicate this

approach in the pricing problem. In our case, the real value of the node is the accumulated reward from the current state until the end of the simulation.

We try different value estimation heuristics, and we divide them into categories.

The first category we consider is static evaluation heuristics that directly evaluate the state node from the information about the state included in the node. The calculation is fast, but the information from the state might not be sufficient.

The next category is rollout heuristics. The standard UCT uses random rollout, but rollouts can also be informed to prefer taking certain actions. The rollout traverses through the state space, so the information about value is not limited to the current state node. That is an advantage in the problems, where the properties of the current state do not provide sufficient information about the future rewards.

We can combine these two categories using rollout, which is stopped in a given depth, from where we statically evaluate the reached state.

5.1.1 Static Evaluation Heuristics

The first category of methods is estimating the values directly in the current node. We have implemented the following methods.

- Constant value heuristic: The value of the node is statically set to a constant value. We describe the choice of the *constant* in Chapter 6.

$$v(s) = constant$$

- Capacity Heuristic: The value of the MCTS node is set as a sum of free spaces remaining in the given state multiplied by k, which is a hyperparameter. k is there because this heuristic counts the node's value from free spaces, but the value should be given by revenue. By the k, we try to adjust that discrepancy. We describe the choice of the k in Chapter 6

$$v(s) = k \sum c_i$$

  The idea is that the more free spaces, the better the potential reward in the future.

- Time-Capacity Heuristic: The node's value is set as a sum of free spaces remaining in a given state multiplied by k, which is a hyperparameter with the same reason as hyperparameter in the previous case and divided by a current timestep.

$$v(s) = k \sum c_i / t$$

The idea from the previous heuristic is extended by the idea that the lesser the remaining time, the lesser the potential reward.

- Expected Reward Heuristic: Because we have information from the simulation about the user budgets and information about the demand, we can use it to evaluate tree nodes. We set the value of the tree node to the following value:

$$v(s) = \Delta T \sum_{i=1} pr_i * \mu B_i$$

Where pr_i is the probability that product i is sampled, ΔT is the number of remaining timesteps from the current state. The user budget is a normal distribution, and μB_i is the mean.

- Minimized Expected Reward Heuristic: The expected reward heuristic does not consider a situation when there are more requested resources than the capacity. We tackle that problem by taking minimum of the expected reward heuristic and the number of free resources multiplied by the mean value of the budget for a product consisting of one resource. Value of the tree node:

$$v(s) = \min(\Delta T \sum_{i=1} pr_i * \mu B_i, \sum c_i * \mu B_{|p|=1})$$

All the variables are the same as in the previous heuristic, and $\mu B_{|p|=1}$ is the mean for budget for a product consisting of one resource.

This heuristic does not need a hyperparameter k as the value from this heuristic corresponds to the reward from selling the products.

These heuristics are less computationally demanding than the rollouts because rollouts are traversing N_r nodes where N_r is a number of remaining nodes until a node with a finite state. We expect that the more complex heuristics yield better results because they better approximate the real value of the state. We also expect that in MCTS with fewer iterations, these heuristics give better results than the random rollout because they are not based on randomness.

5.1.2 Rollout Value Estimation Methods

To estimate the value of the node, we can simulate traversing the MDP until the terminal state is reached. We take the cumulative reward obtained by traversing the MDP and assign it to the value of the node.

- Random Rollout Heuristic: The baseline method where actions are chosen randomly. This method is used in the base version of the UCT

because it not computationally demanding. This method ensures that UCT coverges to the global maximum with an increasing number of simulations because the outcome of the simulation is random.

- Informed Rollout Heuristic: Choosing the action is based on sampling from the user budget B distribution for the product requested in the given state. We take that value from the distribution and use that value as an action instead of a random action.
- Cutoff Mean Heuristic: This is a method where we choose random actions until a defined number of steps is executed. Then we count the value as an average reward for one unit of capacity and multiply it by the original capacity before the rollout. The formula for estimating value is following:

$$v(s) = \frac{\sum_{i=1}^{d} r_i}{c_1 - c_d} * c_1$$

Where r_i is a reward obtained in step i and c_i is a number of free spaces available in step i. This method can be considered as a static evaluation method because we do not take the cumulative reward, but we replace the cumulative reward with an approximation based on shallower rollout.

5.1.3 Linear Programming Heuristics

One problem with the mentioned heuristics is that they neglect the fact that we try to find the optimal solution for setting the prices of different products, which influences how the capacity is used. We try to estimate the upper bound of the possible revenue by solving the linear program.

The linear program tries to maximize the reward by finding the optimal proportion of all products based on the mean budget for the product and expected demand. The capacity constraints the linear program. The linear program is given as:

$$\begin{aligned} \text{maximize} \quad & \sum_{p} \mu B_p * x_p * ed_p \\ \text{subject to} \quad & \sum_{p} x_p * res_p(i) * ed_p \leq c_i, \qquad i = 1, \ldots, |r| \\ & 0 \leq x_p \leq 1, \quad p = 1, \ldots, |P| \end{aligned}$$

Where x_p is the free variable, it gives the ratio of the appearance of the product in the final result. For example, if the x_p is 0.5, it means that we choose the product half the times from the expected appearances of the product. μB_p is a mean for budget distribution for product p, ed_p is an expected demand for the product p, it is a sum of probabilities of arrival

request, for the product p, over each timestep. In other words, it is the total expected number of requests for product p. $res_p(i)$ is requirement of resource r_i in the product p. c_i is a supply o resources r_i.

For solving the linear program, we use JuMP.jl [8] framework with GLPK linear program solver.

- Linear Program Heuristic: The value of the node is estimated as the maximized value from the linear program. We add Linear Program Heuristic to the static evaluation heuristics

Chapter 6

Experiments

In this chapter, we present how different heuristic methods performed for the problem of dynamic pricing. We specify the problem settings and show the results.

6.1 Problem Settings

We need to specify the instances of the problem and the parameters of the RPSDP problem.

The following attributes define the instance: number of resources, number of products, number of timesteps, initial capacity (supply of resources), demand for each product at each timestep, set of actions, and user budget for all products.

The number of resources gives the number of products because the product consists of the adjacent resources. The number of products is a triangular number given by $N_r(N_r + 1)/2$, where N_r is a number of resources.

Demand for products is given by a categorical distribution for each timestep, where all products have assigned their probability. We calculate these probabilities from the parameter expected resources, which gives the total amount of resources requested. From that amount, we calculate the product probabilities (products are composed of the resources).

The user budget is a distribution from which the simulation samples the budgets of the customers. We consider the normal distribution, where the number of resources in the product determines the mean of the distribution. The mean of the product's budget distribution is simply the price of the product with one resource multiplied by the number of resources in the

timesteps	capacity	number of resources	expected resources
1000	55	3	400

Table 6.1: Parameters for the bus pricing problem.

name	description
col_depart	departure time
col_arive	arrival time
col_space	current free spaces in bus
col_price	current price of the ticket
date	date of bus depart
scrape_time	scraping timestamp

Table 6.2: Dataset features.

product.

We set the variables to be similar to the bus line from Prague to London with two intermediate stations, as we can see in Table 6.1.

We set the expected resources higher than the available resources because, with higher demand than supply, the dynamic pricing can more strategize with the price, expecting better requests in the future.

Data

To evaluate the pricing method on a real-world dataset, we have collected information about ticket prices and remaining free seats before the departure of the buses.

The dataset was obtained from the Student Agency and Flixbus reservation systems by using web scraping techniques. The dataset features are in Table 6.2

Unfortunately, after cleaning and preparing the dataset, it turned out that there is only a very limited number of usable datapoints that could generate only few distinct simulations runs. Evaluating stochastic methods such as MCTS on such a small dataset would not provide convincing results as to the effectiveness of the proposed heuristics. Therefore, instead, we have opted to evaluate the MCTS heuristics in fully simulated experiments, where the number of distinct simulated runs is virtually unlimited, and the statistical significance of our results is therefore much higher. However, we have used

number_of_iterations	depth	C_p	max_time
100	∞	40	1 second

Table 6.3: Used parameters for the MCTS.

the dataset to select the parameters for these simulated runs (see Table 6.1).

6.2 Parameters Tuning

6.2.1 UCT Parameters

The exploration constant in the UCT algorithm is a hyperparameter. We have tried different exploration constants. The best results UCT with standard rollout yielded with C_p set to 40. For the static evaluation heuristics, the differences between the performances with different C_p were insignificant.

We have tried to vary the depth and number of iterations. It yielded predictable results. For every method, the higher the depth or number of iterations, the better the results.

When we were trying the different number of iterations and depth, the change had a similar impact on all value estimation methods, and the exploration constant was inert to the non-rollout methods. Therefore, we can fix the hyperparameters of the UCT to the following values 6.3:

We set the depth to infinity because we do not want to constrain the MCTS, and we set the C_p to 40 as the best performing value for the rollouts, and *max_time* for one iteration is set to 1 second because that time is enough even for the most complex heuristics.

6.2.2 Static Evaluation Heuristics Parameters

We need to set the *constant* for the Constant Value Heuristic and k for the Capacity Heuristic and Time-Capacity Heuristic.

When we estimate the value of the tree node in a non-terminal, state we are replacing the cumulative sum of rewards gained by traversing to the terminal state by value, which might not correspond to the cumulative sum of rewards. We tried to use different values for $constant \in (0.1, 1, 2, 5, 10, 20, 50, 100, 1000)$ in the Constant Value Heuristic.

For the very same reason, we tried different multipliers $k \in (0.01, 0.1, 0.2, 0.5, 0.75, 1.25, 1.5, 2, 5, 10, 100)$ for Capacity Heuristic and Time-Capacity Heuristic. Furthermore, we tried to square and square root the value instead of multiplying it.

As we can see in Table 6.4 the best k for Capacity heuristic was $k = 2$ as well as for the Time-capacity heuristic. And the best performing constant for the Constant Value Heuristic was 50. However, the differences between the results were insignificant, except for the situation when the value of the Capacity Heuristic was significantly increased. Then the method performed poorly because the algorithm was only exploiting the best moves.

	function	revenue
1	f(x)=$0.01 * x$	1359.65
2	f(x)=$0.1 * x$	1358.90
3	f(x)=$0.2 * x$	1355.75
4	f(x)=$0.5 * x$	1359.05
5	f(x)=$0.75 * x$	1358.50
6	f(x)=$1.25 * x$	1356.40
7	f(x)=$1.5 * x$	1371.00
8	f(x)=$2.0 * x$	1382.00
9	(x)=$5.0 * x$	1318.55
10	f(x)=$10.0 * x$	503.10
11	f(x)=$100.0 * x$	465.15
12	f(x)=x^2	489.00
13	f(x)=$\sqrt{x}$	1358.15

(a) : Capacity Heuristic

	function	revenue
1	f(x)=$0.01 * x$	1352.90
2	f(x)=$0.1 * x$	1355.40
3	f(x)=$0.2 * x$	1356.75
4	f(x)=$0.5 * x$	1356.90
5	f(x)=$0.75 * x$	1347.50
6	f(x)=$1.25 * x$	1345.85
7	f(x)=$1.5 * x$	1347.90
8	f(x)=$2.0 * x$	1351.10
9	f(x)=$5.0 * x$	1345.25
10	f(x)=$10.0 * x$	1336.80
11	f(x)=$100.0 * x$	1306.05
12	f(x)=x^2	1333.50
12	f(x)=$\sqrt{x}$	1351.05

(b) : Time-Capacity Heuristic

	constant	revenue
1	0.1	1346.6
2	1	1341.3
3	2	1342.75
4	5	1340.45
5	10	1342.6
6	20	1346.8
7	50	1347.85
8	100	1343.9
9	1000	1334.55

(c) : Constant Heuristic

Table 6.4: Different parameters for static heuristics.

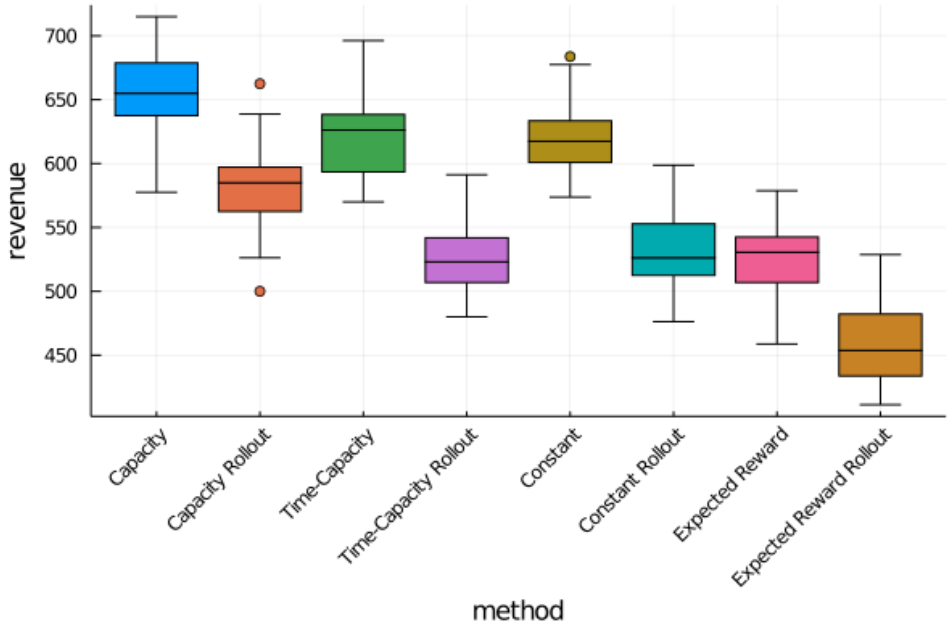


Figure 6.1: Revenue results for static heuristics and for static heuristics combined with the rollouts.

6.3 Results

We run the methods on the described problem with set hyperparameters from the previous section. We used a machine with a processor Intel Core i5-8250U CPU @ 1.60-3.40GHz with four cores and 8 GB of RAM. We run the simulation 50 times for each method.

Rollouts With Static Evaluation

Firstly, we examine whether the combination of static evaluation and rollout is useful. We benchmark all the static evaluation heuristics against the rollout that randomly traverses the space until the specified $rollout\_depth$ and evaluate the state in that depth using static evaluation heuristics. We set $rollout\_depth = 10$.

We can see the results in Figure 6.1. The experiment shows that for all static evaluation heuristics, the rollout decreases the revenue, therefore, impairing the results. A possible explanation is that we used low $number\_of\_iterations$. Therefore, the rollouts end up in a set of states that do not generalize well the real state space, and it is better to generalize the state space from the current states.

	method name	mean revenue	mean runtime
1	Capacity	657.900	0.442018
2	Time-Capacity	625.050	0.426530
3	Constant	620.825	0.460681
4	Linear Program	569.875	58.056911
5	Informed Rollout	542.050	14.924433
6	Minimized Expected Reward	537.450	0.594055
7	Expected Reward	519.800	0.572911
8	Random Rollout	518.550	15.220228
9	Cutoff Mean	423.350	3.371367

Table 6.5: Mean runtime for methods ordered by the revenue results.

Results for Heuristic

Now we show how all static evaluation heuristics and rollout heuristics performed. We exclude the combinations as the static evaluations outperform them. For the results, see the boxplots in Figure 6.2. And for their runtime see Table 6.5.

Surprisingly, the best performing heuristic was simple Capacity Heuristics outperforming even the more complex static evaluation heuristic for the same number of iterations. We also tried different problems settings with different number of resources, timesteps and demands. And for all settings the Capacity Heuristics remained the best performing method.

The additional information in the more complex heuristics might create noise that does not provide information for the real value of the states. The Linear Program Heuristics, Expected Reward Heuristic, and Minimized Expected Reward Heuristic probably overestimate the values of the states.

The Constant Value Estimation performed well against other heuristics. Constant value estimation reduces the MCTS because the values of the nodes depend only on the rewards from transition within the built tree and do not consider the simulation phase, which might mean that in our settings of the problem, the longer horizon is not that important for the decisions.

The longer horizon might matter if we set the budgets to be increasing over time. It would also model the real situation that the people buying the tickets later are willing to pay more, and from the data, we know that the bus companies using dynamic pricing are increasing the prices over time.

The static evaluation heuristics generally outperformed the rollout heuristics. The reason might be that the number of iterations was low to approximate precisely the values of the states in the built tree.

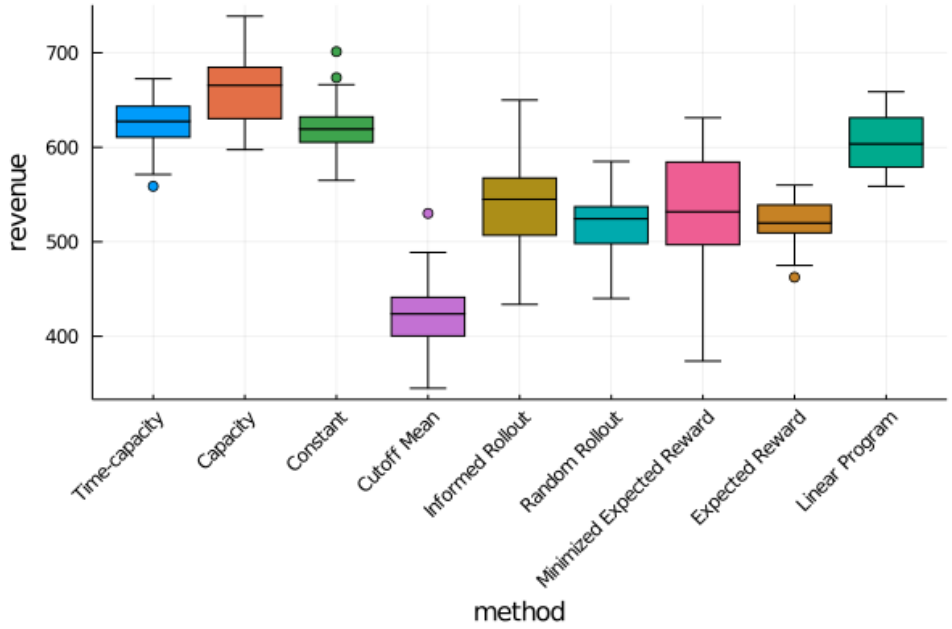


Figure 6.2: Revenue results for evaluation methods.

Number of Iterations

We find the Capacity Heuristic as the best method, outperforming the Random Rollout Heuristic. It can be due to the low number of iteration. We compare how they perform with a different number of iterations. We can see the result in Figure 6.3.

With the higher number of iterations, the rollout heuristic is improving more than the static evaluation method. Static evaluation yields a solid approximation of the state value even with a small number of iterations. The rollout needs more iterations to approximate the real value of the state closely. With the high number of iterations, the rollout outperforms the static evaluation, but the simulation is significantly slower.

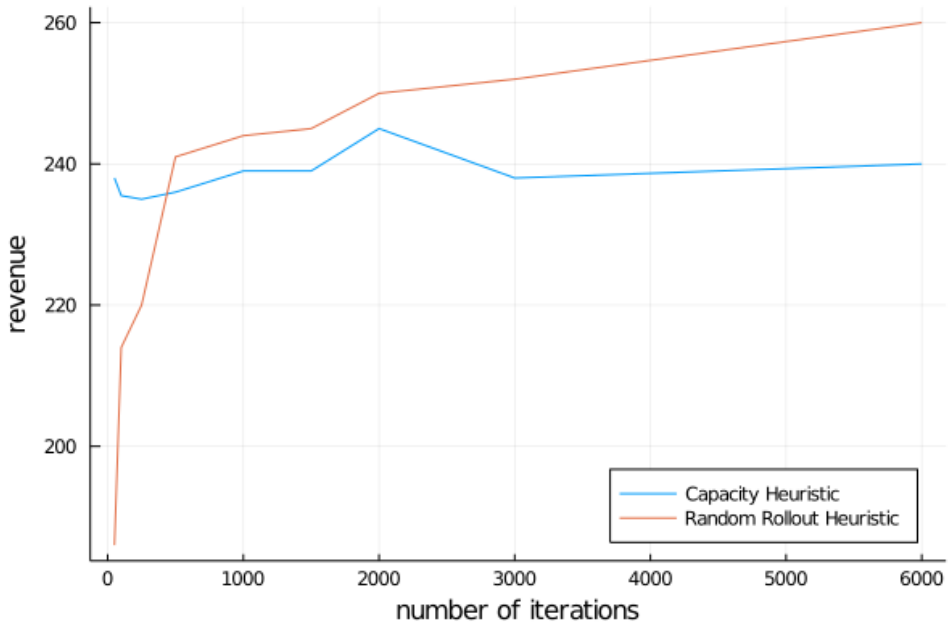


Figure 6.3: Revenue results for different number of iterations for Random Rollout Heuristic and Capacity Heuristic.

Chapter 7

Conclusion

In this thesis, we had the following main objective: Improve the methods for solving sequential, resource-constrained dynamic pricing in mobility.

We examined the related literature in the domain of dynamic pricing. We formally defined the problem as RCSDP and identified the MDP framework as a well-suited tool for modeling our problem of dynamic pricing. With the size of the problem, we identify online solvers as the more sensible method for solving MDP. We used MCTS as a solution technique and incorporated domain knowledge into the MCTS algorithm, specifically by substituting random rollout by function evaluating the state or by informed rollouts. As the main contribution of this thesis, we propose multiple value estimation heuristics using the domain knowledge we have.

The experiments show that the less complex heuristics yield better results than the complex heuristics. Overall the best heuristic was heuristic working only with the remaining capacity. The other finding was that with the increasing number of iterations of MCTS, the rollout heuristics performed better, while the static, less complex heuristics did not record such an increase in the performance. The takeaway is that when we have limited computational power, it is better to use static evaluation heuristics because they perform well even with a small number of iteration, and the heuristic is less computationally demanding itself than the rollout.

There are many possibilities for future work. First, we can enhance the simulation by using real-world data to better simulate the demand model. We can enhance the model to enable the cancelations or simulate requests for more products at one timestep.

There is also room to improve the MCTS. Other heuristics can be crafted, for example, by classifying the states based on the results from previous simulations. Nested Monte Carlo Tree Search might improve the results as it

yielded good results in large MDPs.

Bibliography

[1] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2):235–256, May 2002.

[2] Hendrik Baier and Mark H. M. Winands. Monte-Carlo Tree Search and Minimax Hybrids with Heuristic Evaluation Functions. In Tristan Cazenave, Mark H. M. Winands, and Yngvi Björnsson, editors, *Computer Games*, Communications in Computer and Information Science, pages 45–63, Cham, 2014. Springer International Publishing.

[3] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012. Conference Name: IEEE Transactions on Computational Intelligence and AI in Games.

[4] Keh-Hsun Chen, Dawei Du, and Peigang Zhang. Monte-Carlo Tree Search and Computer Go. In Zbigniew W. Ras and William Ribarsky, editors, *Advances in Information and Intelligent Systems*, Studies in Computational Intelligence, pages 201–225. Springer, Berlin, Heidelberg, 2009.

[5] Yu. A. Chizhov and A. N. Borisov. Markov decision process in the problem of dynamic pricing policy. *Automatic Control and Computer Sciences*, 45(6):361–371, December 2011.

[6] A Collins and L Thomas. Comparing reinforcement learning approaches for solving game theoretic models: a dynamic airline pricing game example. *Journal of the Operational Research Society*, 63(8):1165–1173, August 2012. Publisher: Taylor & Francis.

[7] Rémi Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H.

L. M. (Jeroen) Donkers, editors, *Computers and Games*, Lecture Notes in Computer Science, pages 72–83, Berlin, Heidelberg, 2007. Springer.

[8] Iain Dunning, Joey Huchette, and Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[9] Maxim Egorov, Zachary N. Sunberg, Edward Balaban, Tim A. Wheeler, Jayesh K. Gupta, and Mykel J. Kochenderfer. POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research*, 18(26):1–5, 2017.

[10] Alberto A. Gaggero, Lukas Ogrzewalla, and Branko Bubalo. Pricing of the long-distance bus service in Europe: The case of Flixbus. *Economics of Transportation*, 19:100120, September 2019.

[11] Guillermo Gallego and Garrett van Ryzin. A Multiproduct Dynamic Pricing Problem and Its Applications to Network Yield Management. *Operations Research*, 45(1):24–41, February 1997. Publisher: INFORMS.

[12] Yue Guan, Anuradha M. Annaswamy, and H. Eric Tseng. Towards Dynamic Pricing for Shared Mobility on Demand using Markov Decision Processes and Dynamic Programming. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7, September 2020.

[13] S. James, Benjamin S. Rosman, and G. Konidaris. An investigation into the effectiveness of heavy rollouts in UCT. July 2016. Accepted: 2017-02-23T09:59:05Z.

[14] Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, Lecture Notes in Computer Science, pages 282–293, Berlin, Heidelberg, 2006. Springer.

[15] Andrey Kolobov Mausam. Planning with markov decision processes: an ai perspective. *Morgan & Claypool Publishers*, 2012.

[16] Jeffrey I. McGill and Garrett J. van Ryzin. Revenue Management: Research Overview and Prospects. *Transportation Science*, 33(2):233–256, May 1999. Publisher: INFORMS.

[17] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa. Online Planning Algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32:663–704, July 2008.

[18] Stuart J. Russell, Peter Norvig, and Ernest Davis. *Artificial intelligence: a modern approach*. Prentice Hall series in artificial intelligence. Prentice Hall, Upper Saddle River, 3rd ed edition, 2010.

[19] Garrett Van Ryzin and Qian Liu. *On the Choice-Based Linear Programming Model for Network Revenue Management*. 2004.

[20] Syed Arbab Mohd Shihab, Caleb Logemann, Deepak-George Thomas, and Peng Wei. Autonomous Airline Revenue Management: A Deep Reinforcement Learning Approach to Seat Inventory Control and Overbooking. *arXiv:1902.06824 [cs]*, June 2019. arXiv: 1902.06824.

[21] Dan Zhang and Daniel Adelman. An Approximate Dynamic Programming Approach to Network Revenue Management with Customer Choice. *Transportation Science*, 43(3):381–394, June 2009. Publisher: INFORMS.