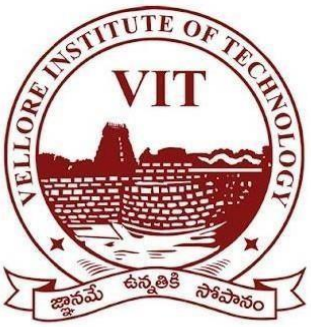


VIT-AP UNIVERSITY

TITLE OF THE PROJECT: **Reinventing Grocery Shopping with Reinforcement Learning**

CSE4037 - Reinforcement Learning

School of Computer Science and Engineering

Name of Students:

21BCE7286 Mandadi Geetha Pallavi

21BCE7418 Mallampati Bhavishya

21BCE9039 Gandla Sreeja

2023 -2024

TABLE OF CONTENTS

Abstract

Grocery shopping stands as a quintessential task in everyday life, yet it often involves complex decision-making processes influenced by factors such as individual preferences, budget constraints, and time limitations. In response to the intricacies of this routine chore, the integration of artificial intelligence, particularly reinforcement learning (RL), has emerged as a promising avenue for optimization. RL empowers autonomous agents to navigate the grocery shopping landscape by learning optimal strategies through iterative interactions with the environment. By employing RL algorithms, these agents can adapt and refine their decision-making processes over time, aiming to maximize cumulative rewards such as cost savings, time efficiency, and customer satisfaction. Such personalized and dynamic approaches hold significant potential for revolutionizing traditional grocery shopping experiences, offering tailored recommendations, mitigating decision fatigue, and ultimately streamlining the overall shopping journey for consumers. In this context, the application of reinforcement learning (RL) techniques serves as a powerful tool to optimize the process of grocery shopping. By formulating the problem within a framework of states, actions, transition probabilities, and rewards, RL algorithms enable an autonomous agent to learn an optimal strategy for navigating through a network of shops to fulfill a given shopping list efficiently. In this project, through the implementation of value iteration, Q-learning, and representation policy iteration methods, the agent can learn and refine its decision-making policy, ultimately simplifying and enhancing the grocery shopping experience for consumers.

List of Abbreviations

- Shops= [S_1, S_2 ,….., S_n]
- buying_status =[I_1,I_2,…..I_m] where I_i be; longs to {Y,N} depicting whether the item i has been bought or not
- States={current_shop, current_buying_status}
- State Space=No of shops x 2^(no: Of items to buy)=n*(2^m)
- Actions=[S_1, S_2 ,….., S_n]

Item Availability Model

Function: availability_in_shop

We will use a Bernoulli Distribution for each item in each shop. Either the item is available or it's not. (X=0 or 1) This function returns the probability P(X= next_status) using the distribution for that particular item and shop.

M function

The M function used ensures that if the action Si chosen ,then the probability of ending up in shop Sj is greater than that of ending up in shop Sk if and only if Sj is closer to Si than Sk.

The function is described as:

$$M(i,j) = \frac{\left(\sum_{p=1}^{n}\sum_{q=1}^{n} distance(S_p,S_q)\right) - distance(S_i,S_j)}{(n-1)\sum_{p=1}^{n}\sum_{q=1}^{n} distance(S_p,S_q)}$$

Price of Item Model

Function: price_penalty

We will use a Gaussian Distribution for price of each item in each shop. The MRP, the price fluctuates slightly around the MRP. Based on the price a penalty (cost) is added.

Distance Travelled Model

Function: distance_penalt

We will have fixed distances between the shops. To account for real world conditions like traffic congestions, we use a Poisson Distribution- parameter passed is distance travelled.

1 Introduction

The landscape of grocery shopping has undergone a profound transformation in recent years, with the advent of online platforms revolutionizing the way consumers acquire essential goods. In this era of digital convenience, users are presented with a vast array of choices, making the process of selecting the right products a complex and often overwhelming task. To address this challenge and elevate the grocery shopping experience, this project proposes a groundbreaking approach that integrates Q- learning, a reinforcement learning algorithm, into the realm of sentiment-driven product recommendations.

Traditional grocery shopping has been characterized by routine visits to physical stores, where consumers make choices based on personal preferences, brand loyalty, and real-time considerations. However, the emergence of online grocery platforms has introduced new dimensions to this process, enabling users to explore and select products from the comfort of their homes. Despite this convenience, the abundance of choices in the digital realm can lead to decision fatigue, prompting the need for intelligent systems that assist users in making informed and personalized selections.

The primary objective of this project is to reinvent the grocery shopping experience by seamlessly incorporating Q-learning, an iterative reinforcement learning algorithm, with sentiment-driven product recommendations. By leveraging user-generated reviews, sentiment analysis, and the adaptive nature of Q-learning, the system aims to optimize and refine product suggestions over time, providing users with tailored recommendations aligned with their preferences and sentiments.

The rationale behind integrating sentiment analysis into the recommendation system lies in the wealth of information embedded in user reviews. Each review encapsulates not only the user's opinion but also the sentiment associated with specific products, offering valuable insights into the preferences and experiences of fellow shoppers. Q-learning, with its iterative learning approach, allows the system to adapt and evolve continuously, learning from user interactions and feedback to improve the precision and relevance of recommendations.

This project stands at the intersection of artificial intelligence, reinforcement learning, and e- commerce, promising a dynamic solution to the challenges posed by the vastness of digital grocery platforms. The integration of Q-learning in the sentiment-driven recommendation system is poised to offer users a more personalized, efficient, and enjoyable grocery shopping journey. As we delve into the details of the project, we will explore the methodology, implementation, and expected outcomes of this innovative approach, with the ultimate goal of redefining the grocery shopping experience in the digital age.

1.1 Objective of the Project

The primary objectives of the project are:

1. Integrating Q-Learning with Sentiment Analysis: - Develop a robust integration of Q-learning, a reinforcement learning algorithm, with sentiment analysis techniques to create a dynamic and adaptive recommendation system for grocery shopping.

2. Sentiment-Driven Product Recommendations: - Leverage sentiment analysis on user-generated reviews to extract valuable insights into the sentiments associated with various grocery products. - Utilize this sentiment data to drive personalized product recommendations, aligning with individual user preferences.

3. Iterative Learning Process: - Implement an iterative approach with Q-learning to continuously refine and optimize the recommendation model based on user interactions and feedback. - Ensure that the system adapts to changing user behaviors, preferences, and evolving sentiment trends over time.

4. Seamless Integration with E-Commerce Platforms: - Design and implement a user-friendly interface that seamlessly integrates with existing online grocery shopping platforms. - Facilitate easy navigation and exploration of sentiment-driven product recommendations, enhancing the overall user experience.

5. Efficiency and Accuracy in Recommendations: - Strive to minimize the learning curve for the recommendation system, providing users with accurate and relevant grocery product suggestions from the outset. - Focus on enhancing the precision of recommendations to improve user satisfaction and engagement with the grocery shopping platform.

6. Increased User Satisfaction and Engagement: - Aim to create a more personalized and enjoyable grocery shopping experience for users by offering recommendations that resonate with their sentiments and preferences. - Measure and evaluate user satisfaction and engagement metrics to assess the impact of the Q- learning-driven sentiment-based recommendation system.

7. Insights for Retailers: - Provide valuable insights for retailers into evolving sentiment trends and customer preferences within the grocery domain. - Enable retailers to make informed decisions regarding product assortment, marketing strategies, and inventory management based on the sentiment-driven data.

8. Optimization of Grocery Supply Chain: - Explore the potential impact of sentiment-driven recommendations on optimizing the grocery supply chain by influencing inventory levels, reducing waste, and enhancing sustainability.

1.2 Need of the Work

The application of reinforcement learning (RL) to optimize grocery shopping addresses several key needs and challenges in today's consumer landscape:

1. **Personalized Recommendations:** Consumers increasingly seek personalized experiences tailored to their preferences, dietary restrictions, and budget constraints. RL algorithms can analyze individual shopping behaviors and preferences to provide customized recommendations, leading to more satisfying and efficient shopping experiences.

2. **Time Efficiency:** Grocery shopping can be time-consuming, particularly for individuals with busy schedules. RL-powered systems can optimize shopping routes, prioritize items, and minimize time spent in-store, allowing consumers to complete theirshopping quickly and efficiently.

3. **Cost Savings:** Rising food prices and budget constraints prompt consumers to seek ways to save money on groceries. RL algorithms can identify cost-effective purchasing strategies, such as selecting discounted items or suggesting budget-friendly alternatives,thereby helping consumers optimize their spending.

4. **Decision Support:** The abundance of choices in grocery stores can overwhelm consumers and lead to decision fatigue. RL-powered systems can provide decision support by recommending products based on past purchases, nutritional preferences, and price sensitivity, thereby simplifying the decision-making process and reducing cognitive load.

5. **Sustainability:** There is growing awareness of the environmental impact of food production and consumption. RL algorithms can promote sustainable shopping behaviors by encouraging the purchase of locally sourced, organic, or eco-friendly products, as well as minimizing food waste through optimized purchasing and meal planning.

6. **Adaptability:** The grocery shopping landscape is dynamic, with changes in product availability, pricing, and consumer preferences. RL algorithms can adapt and learn from new data in real-time, enabling retailers to respond quickly to market trends and consumer demand.

Overall, applying RL to grocery shopping addresses the need for personalized, efficient, cost-effective, and sustainable shopping experiences, ultimately enhancing consumer satisfaction and driving business success for retailers.

1.3 Scope and Motivation

Scope:

The scope of this project extends beyond the traditional boundaries of online grocery shopping, aiming to redefine the user experience by seamlessly integrating Q-learning with sentiment analysis in a recommendation system. The project envisions a dynamic platform that not only assists users in navigating the expansive array of grocery products but also understands and adapts to their individual sentiments and evolving preferences. By leveraging advanced machine learning techniques, the system aspires to deliver a more personalized, efficient, and engaging grocery shopping experience, ultimately contributing to the evolution of e-commerce platforms.

The scope encompasses the integration of Q-learning, a reinforcement learning algorithm, with sentiment analysis techniques to drive an iterative and adaptive recommendation system. The application of Q-learning allows the system to continuously refine and optimize recommendations based on user interactions, fostering a more accurate and personalized experience. Sentiment analysis, applied to user-generated reviews, adds a layer of depth to the recommendation process, enabling the system to consider the emotions and experiences associated with specific grocery items.

Moreover, the project aspires to transcend individual user experiences and extend its impact to benefit retailers. By providing retailers with insights into evolving sentiment trends and customer preferences, the project aims to contribute to better decision-making in inventory management, product assortment, and marketing strategies. This holistic approach positions the project as a comprehensive solution with the potential to reshape the landscape of online grocery shopping and enhance the synergy between users and retailers.

Motivation:

The motivation for this project stems from the recognition of existing challenges in the online grocery shopping domain and the potential for innovative solutions to address them. The surge in online grocery platforms has democratized access to a vast array of products, but the lack of personalization often results in a suboptimal user experience. The motivation is grounded in the belief that by integrating Q-learning and sentiment analysis, we can elevate the user experience, providing shoppers with recommendations that are not only tailored to their preferences but also resonate with their sentiments and emotions.

The ever-expanding landscape of e-commerce, coupled with the advancements in machine learning, presents an opportune moment to explore new paradigms in recommendation systems. The motivation further lies in the desire to harness the power of reinforcement learning and sentiment analysis to create a system that learns and adapts, ensuring that the recommendations remain relevant in the face of evolving user behaviors and sentiment trends.

Additionally, the project is motivated by the potential benefits it offers to retailers. By providing retailers with insights derived from sentiment-driven data, the project aims to foster a symbiotic relationship between users and retailers. Retailers stand to gain a deeper understanding of customer preferences, enabling them to optimize inventory, enhance customer engagement, and tailor marketing strategies more effectively.

1.4 Organization of the Report

Who the report is about and/or prepared for

The report is about optimizing grocery shopping through reinforcement learning (RL) algorithms and is prepared for stakeholders in the retail industry, including grocery store managers, retailers, AI developers, and researchers interested in consumer behavior and optimization techniques.

What was done, what problems were addressed, and the results, including conclusions and/or recommendations

The report details the application of RL algorithms to address challenges in traditional grocery shopping, such as personalization, time efficiency, cost savings, decision support, sustainability, and adaptability. By formulating the problem within an RL framework and implementing algorithms like value iteration, Q-learning, and representation policy iteration, the project aims to optimize shopping strategies and enhance the overall shopping experience for consumers.

Where the subject studied occurred

The subject studied occurred in various contexts, including physical grocery stores, online grocery platforms, and simulated environments for research and development purposes.

Why the report was written (function), including under what authority, for what reason, or by whose request

The report was written to fulfill several functions:
To present the findings and insights gained from applying RL to grocery shopping optimization.
To offer recommendations for retailers and AI developers on implementing RL algorithms to improve operational efficiency and customer satisfaction.
To identify future research directions and opportunities for further exploration in the field of AI-enabled grocery shopping optimization.

The subject operated by employing RL algorithms to analyze consumer preferences, optimize shopping routes, recommend products, and adapt strategies in real-time based on changing market conditions and consumer behavior. This approach enhances decision-making processes, streamlines operations, and ultimately improves the shopping experience for consumers.

2 Literature Survey

2.1 Basics

The modern era witnesses an ever-increasing demand for convenience, efficiency, and personalization in various aspects of daily life. One such area is grocery shopping, which traditionally involves navigating through multiple stores to find desired items at competitive prices. This process is not only time-consuming but also poses challenges for individuals with busy schedules or specific preferences. While online grocery platforms offer a degree of convenience, they often fall short in providing personalized recommendations tailored to individual preferences effectively. To address these challenges and enhance the grocery shopping experience, researchers and developers have explored various technologies and methodologies, one of which is the utilization of agent-based systems powered by reinforcement learning (RL) algorithms.

2.2 Preceding works

In their work, Kwang Hyoun Joo et al. **[** 1**]**introduce an agent-based grocery shopping system designed to automate the process of grocery shopping while considering user preferences. Their system utilizes a network of agents to gather information from multiple grocery stores, compare it with user preferences, and adapt over time based on user feedback.

The concept of utilizing agent-based systems in e-commerce, as discussed by Kwang Hyoun Joo et al., resonates with the research conducted by T. Kinoshita. In their research, Kinoshita explores the potential of agent-based systems in organizing and coordinating various tasks within e-commerce environments. Their work highlights the role of organization agents in managing primitive agents and facilitating communication, which aligns with the organizational structure proposed in the agent-based grocery shopping system.

Furthermore, the approach of integrating user preferences into decision-making processes bears resemblance to the research conducted by N. Shiratori. In their work on personalized recommendation systems, Shiratori et al. emphasize the importance of considering user preferences and feedback to enhance the effectiveness of recommendation algorithms. This aligns with the goal of the agent-based grocery shopping system to adapt to user preferences over time by learning from previous shopping results.

By synthesizing insights from the works of Kwang Hyoun Joo, T. Kinoshita, and N. Shiratori, it becomes evident that the proposed agent-based grocery shopping system represents a convergence of ideas from the fields of agent-based systems, e-commerce organization, and personalized recommendation algorithms. The system

aims to address the challenges of traditional grocery shopping by automating the process, considering user preferences, and facilitating efficient decision-making.

In their research, Hongying Du and Michael N. Huhns introduce **[2]** an innovative approach to social grocery shopping, where customers exchange information on item prices and quantities to identify the best deals and most convenient shopping plans. This strategy aims to empower consumers by enabling them to make more informed purchasing decisions and potentially achieve significant cost savings. Central to their approach is the utilization of agents representing individual customers, which helps facilitate practical implementation and fosters trust among participants by potentially learning which agents can provide reliable information. By leveraging real-world shopping lists based on the U.S.Consumer Price Index, the authors ensure the realism and relevance of their findings.

The proposed system addresses a common challenge faced by consumers in traditional grocery shopping: the lack of easily accessible price comparison tools and the inconvenience of visiting multiple stores to find the best deals. By envisioning an online platform where customers can share price information and obtain pointers to stores offering the lowest total prices for their desired items, Du and Huhns aim to democratize the shopping experience and encourage fairer interactions between customers and stores. However, the success of such a system hinge on the willingness of customers to actively participate and contribute accurate price data, highlighting the importance of effective implementation and incentivization strategies.

A key aspect of the study involves assessing the robustness of the proposed multi-agent shopping system in the face of potential errors and manipulation, such as inaccuracies in reported prices or changes in store pricing strategies. Through simulations incorporating random and systematic errors, the authors evaluate the system's ability to withstand various challenges and maintain its effectiveness in optimizing cost savings for customers. This evaluation provides valuable insights into the feasibility and reliability of the proposed approach, helping to inform discussions on its practical implementation and potential benefits for consumers in real-world grocery shopping scenarios.

The key difference between this paper and the previous one lies in the approach to grocery shopping optimization. While both papers aim to enhance the shopping experience by leveraging agent-based systems, the focus here is on social interaction among customers to share price information and optimize savings. In contrast, the previous paper emphasizes automation and adaptation of agents to user preferences. Additionally, this paper highlights the potential savings and robustness of the proposed multi-agent shopping system in the presence of errors in reported prices, which is a unique aspect not explored in the previous work.

In their work, Kwang Hyoun Joo, T. Kinoshita, and N. Shiratori **[3]** introduce an agent-based grocery shopping system aimed at automating the shopping process by gathering information from multiple stores and comparing it with user preferences. Their system utilizes role agents, including user agents, information management agents, and store server agents, to facilitate cooperation and achieve user-defined goals.

The proposed system focuses on reducing user effort and saving time by purchasing the best groceries based on user preference. It covers functional requirements of a grocery shopping system and supports the five stages of consumer buying behavior model. However, it's worth noting that this paper does not explicitly mention the incorporation of reinforcement learning for adapting to user preferences over time, which could be a potential difference compared to other research works.

The third paper could build upon the findings of the first paper, incorporating feedback, addressing limitations, or presenting new developments in the agent-based grocery shopping system. Overall, while the core concept of an agent-based grocery shopping system remains consistent across both papers, differences in scope, detail, publication venue, and temporal context could contribute to variations in their content and contributions.

In their work, L. Benedicenti, Xuguang Chen, Xiaoran Cao, and R. Paranjape **[4]** present an agent- based shopping system designed to assist supermarket shoppers both at home and during their shopping trips. The system utilizes lightweight agent implementation called TEEMA (TRLabs Execution Environment for Mobile Agents), which is built upon a microkernel concept providing basic services for agent communication, migration, and location. Additional services such as name services, storage services, security services, and database services can be added on top of TEEMA to enhance functionality.

The proposed system facilitates supermarket shopping by enabling users to send agents with shopping lists to be selected supermarkets, where the agents retrieve limited price lists. The system also incorporates a residential gateway to protect user information during agent travel. Upon returning to the user, the system informs them of the search results, and if the user decides to visit a supermarket, an agent is sent there through the residential gateway to access complete price lists. The system architecture is distributed, with logical components located at the user's location, residential gateway, mobile terminal, and participating supermarkets.

The authors highlight the advantages of their agent-based approach, emphasizing its ability to automate inventory management, assist in supermarket selection based on price and availability, and provide real-time updates on special offers. Additionally, they discuss the integration strategy to make the system compatible

with legacy database and server software. However, they also acknowledge the challenges and limitations, particularly regarding privacy protection, user authentication, and system robustness in the face of potential errors or manipulation. The comparison with other supermarket shopping systems underscores the unique strengths and weaknesses of the proposed agent-based approach in optimizing the shopping experience for consumers.

E-commerce has increased tremendously in recent decades because of improvements in information and telecommunications technology along with changes in societal lifestyles. As highlighted by Adrian Serrano-Hernandez et al. (2017) **[5]** , horizontal cooperation among companies can lead to significant benefits, such as reducing transportation costs, improving service quality, diminishing environmental impact, mitigating risk, and enhancing market share. More recently, e-grocery (groceries purchased online) including fresh vegetables and fruit, is gaining importance as the most-efficient delivery system in terms of cost and time. Javier Faulin and Rocio de la Torre have noted the logistics challenges associated with e-groceries, including food safety issues, differences in storage temperatures, and perishability over time (Fredriksson and Liljestrand 2015).

In this study, the authors evaluate the effect of cooperation-based policies on service quality among different supermarkets in Pamplona, Spain. Concerning the methodology, they deploy, firstly, a detailed survey in Pamplona to model e-grocery demand patterns. As highlighted by Luis Cadarso, consumers and sellers often have conflicting preferences regarding product perishability, with consumers preferring longer shelf lives and sellers benefiting from shipping shorter-lived items first to reduce food waste (Teller et al. 2018; Fikar 2018). Secondly, the authors develop an agent-based simulation model for generating scenarios in cooperative and non-cooperative settings, considering the real data obtained from the survey analysis. Thus, a Vehicle Routing Problem is dynamically generated and solved within the simulation framework using a biased-randomization algorithm. Finally, the results show significant reductions in lead times and better customer satisfaction when employing horizontal cooperation in e-grocery distribution.

Table 1: Contributions in E-Grocery, Horizontal Cooperation, and Agent-Based Simulation

S.No	Title	Date of Conference	Authors	Proposed Method
1	Agent-based grocery shopping system based on user's preference.	04-07 July 2000	Kwang Hyoun Joo; T. Kinoshita; N. Shiratori	An agent-based grocery shopping system automates shopping by gathering information from multiple stores, comparing it with user preferences, and adapting over time through feedback.
2	A Multiagent System Approach to Grocery Shopping	2011	Hongying Du & Michael N. Huhns	The proposed method entails customers exchanging information on item prices and quantities to optimize savings and convenience, facilitated by agents representing customers.
3	Design and implementation of an agent-based grocery shopping system.	25-11 Nov 2000	Kwang Hyoun JOO Tetsuo KINOSHITA Norio SHIRATORI	The proposed method involves an agent-based grocery shopping system that consists of three role agents: a user agent, an information management agent, and a store server agent, which cooperate to purchase groceries according to user preferences.
4	An agent-based shopping system	02-05 May 2004	L.Benedicenti; Xuguang Chen; Xiaoran Cao; R. Paranjape	The method involves employing TEEMA an agent, to facilitate agent-based shopping, enabling users to send agents with shopping lists to select supermarkets, retrieve price lists, and receive real-time updates on special offers, ensuring
				privacy and compatibility with legacy software.

5	Agent-based simulation improves E-grocery deliveries using horizontal cooperation.	14-18 December 2020	Adrian rrano-Hernandez; Javier Faulin; Rocio e Torre; la Cadarso Luis	An agent-based simulation model to evaluate the effect of horizontal cooperation on lead times and customer satisfaction in e-grocery distribution.
6	Agent-based simulation of consumer behavior in grocery shopping on a regional level	August 2007	Tilman A. Schenk, Günter Löffler, Jürgen Rauh	The proposed method is an agent-based micro model for simulating spatial choice in grocery shopping behavior based on individual population.

3 Existing Method & Disadvantages

The proposed development of an agent-based grocery shopping system addresses longstanding challenges inherent in traditional grocery shopping processes. Traditional methods often demand significant time and effort from consumers, requiring them to navigate through multiple stores to find desired items at competitive prices. Moreover, these methods may not sufficiently cater to individual preferences, leaving consumers with suboptimal choices. Recognizing these inefficiencies, the proposed system aims to revolutionize the grocery shopping experience by leveraging advanced technologies and algorithms to streamline the process.

Existing literature in e-commerce and consumer behavior underscores the importance of personalized recommendations in enhancing user satisfaction and engagement. Studies by researchers like N. Shiratori et al. have delved into the realm of personalized recommendation systems, which aim to tailor product suggestions to individual user preferences. However, traditional recommendation approaches often lack real-time adaptability, failing to consider dynamic factors such as price fluctuations and item availability. By integrating reinforcement learning (RL) algorithms into the proposed system, it bridges this gap by enabling agents to learn and adapt their decision-making processes based on evolving conditions, thereby providing more personalized and relevant recommendations to users.

Furthermore, research in RL, particularly the foundational work by Sutton and Barto, has demonstrated the efficacy of RL algorithms in optimizing decision-making processes in dynamic environments. By modeling the grocery shopping problem as a Markov Decision Process (MDP) and employing Q- learning, the proposed system empowers agents to make informed decisions regarding store selection and item purchases. This approach capitalizes on the inherent ability of RL algorithms to learn from experience and iteratively refine decision-making policies, ultimately optimizing the overall grocery shopping experience for consumers.
The integration of RL algorithms in the proposed system represents a significant departure from traditional e-commerce platforms, which often rely on static recommendation systems. By harnessing the power of RL, the system can adapt to changing user preferences, market conditions, and store offerings in real-time, thereby ensuring a more personalized and efficient shopping experience. Moreover, the proposed system aligns with broader trends in automation and optimization within the e-commerce landscape, reflecting a concerted effort to leverage cutting-edge technologies to address longstanding challenges and enhance consumer satisfaction.

In summary, the proposed agent-based grocery shopping system represents a convergence of insights from e-commerce, personalized recommendation systems, and reinforcement learning. By leveraging advanced algorithms and real-time data, the system seeks to revolutionize the grocery shopping experience, offering consumers unprecedented levels of convenience, personalization, and efficiency. Through iterative refinement and adaptation, the system holds the promise of reshaping the future of grocery shopping, making it more seamless, tailored, and enjoyable for consumers worldwide.

Disadvantages:

While the proposed agent-based grocery shopping system holds promise for revolutionizing the grocery shopping experience, it also faces several potential drawbacks and challenges that warrant consideration:

Complexity and Scalability: Implementing an agent-based system can introduce complexity, particularly in terms of system architecture and scalability. As the number of users and transactions increases, managing a large network of agents and ensuring efficient communication between them may become challenging. This complexity could hinder the system's ability to scale effectively to accommodate a growing user base and handle increasing transaction volumes.

Data Privacy and Security Concerns: The system's reliance on gathering and analyzing user data to tailor recommendations raises significant privacy and security concerns. Collecting sensitive information about users' preferences, purchasing habits, and location data may pose risks if not adequately protected. Ensuring robust

data privacy measures and compliance with regulations such as GDPR (General Data Protection Regulation) is essential to maintain user trust and mitigate potential security breaches.

Algorithm Bias and Fairness: The use of reinforcement learning algorithms introduces the risk of algorithmic bias, where the system's recommendations may reflect and perpetuate existing societal biases or preferences. Without careful design and oversight, the system may inadvertently favor certain demographics or product categories over others, leading to disparities in recommendation accuracy and fairness. Addressing algorithmic bias requires ongoing monitoring, evaluation, and mitigation strategies to ensure equitable treatment for all users.

Limited User Control and Transparency: While the system aims to automate and optimize the grocery shopping process, it may limit user control and transparency over decision-making. Users may feel disenfranchised if they perceive the system as making decisions on their behalf without sufficient input or explanation. Providing users with greater control over preferences, recommendations, and decision-making processes, as well as enhancing transparency in how the system operates, is essential to foster trust and user acceptance.
Dependency on External Factors: The effectiveness of the system is contingent on various external factors, including the availability and accuracy of data from grocery stores, fluctuations in market conditions, and the reliability of communication networks. Any disruptions or inaccuracies in these external factors could compromise the system's performance and user experience. Mitigating such dependencies may require redundancies, fallback mechanisms, and continuous monitoring to ensure robustness and resilience.

4 Project Flow/ Framework of the Proposed System

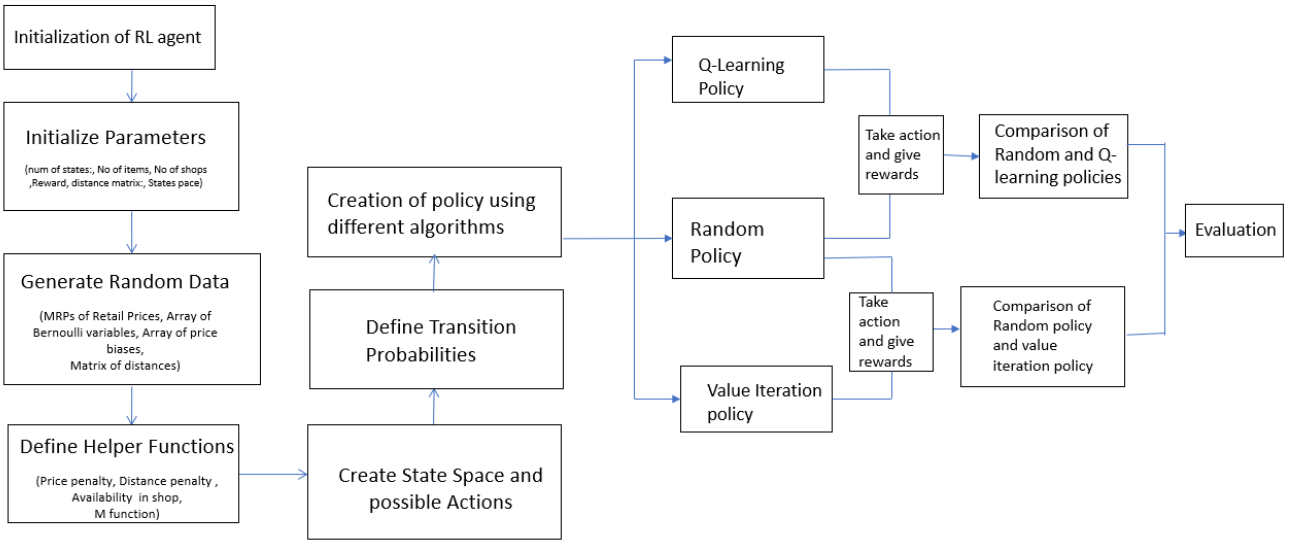


Fig (1). Complete process flow Diagram

Data Setup:

In this part, the code initializes various parameters required for the problem. This includes the prices of items (MRPs), the number of shops, the number of items, etc. MRPs are stored in a dictionary where each item is associated with its price.
State Space Definition:
The state space is a set of all possible states the agent can be in. In this case, each state consists of two components: the shop number and the buying status of items. The buying status of items is represented as a binary vector indicating whether each item is bought or not.

Transition Probabilities and Rewards:

Transition probabilities determine the likelihood of transitioning from one state to another when an action is taken. Rewards represent the immediate benefit (or cost) associated with taking an action from a particular state to another state.

The transition probabilities and rewards for each possible action (selecting a shop) from a given state to another state. It considers factors such as availability of items in shops, prices of items, and distances between shops.

Q-learning Algorithm:

Q-learning is a model-free reinforcement learning algorithm for learning optimal policies. It learns a Q-value for each state-action pair, representing the expected future reward of taking that action in that state.
The Q-learning function iterates over episodes, where each episode consists of multiple steps. In each step, the agent selects an action based on an epsilon-greedy policy (balancing exploration and exploitation), updates Q-values based on the observed reward and transitions, and collects rewards.

Testing:
After learning the optimal policy using Q-learning, the code tests this policy against a random policy.
It does so by simulating shopping scenarios and comparing the rewards obtained using both policies.

Visualization:

The rewards obtained over episodes during Q-learning and rewards obtained from testing the learned policy against the random policy. This visualization helps in understanding the learning progress and performance of the learned policy.
The best action (shop) for each state, indicating the learned optimal policy. It also displays plots of rewards obtained during learning and testing, providing insights into the learning process and policy performance.

Functions:

These are utility functions used in the Q-learning algorithm and reward calculation process. They calculate penalties based on item prices, shop distances, item availability probabilities, and parameters for transition probabilities.
This executes the Q-learning algorithm for a specified number of episodes and steps. It updates Q- values based on observed rewards and transitions, iteratively improving the learned policy.
This section tests the learned policy against a random policy for multiple scenarios and collects rewards. It helps evaluate the performance of the learned policy in comparison to a baseline (random) policy.
Overall, aim of this project comprehensive Q-learning-based solution to optimize shopping decisions across multiple shops, considering factors such as item availability, prices, and shop distances.

5 Hardware and Software requirements

Hardware Requirements:

Processor:
A multi-core processor with a clock speed of 2.5 GHz or higher is recommended to handle the computational demands of reinforcement learning algorithms, particularly during training phases.

Memory (RAM):
A minimum of 16 GB RAM is advised to accommodate the memory-intensive nature of reinforcement learning tasks, especially when dealing with large datasets.

Graphics Processing Unit (GPU):
An NVIDIA GPU with CUDA support is highly beneficial for accelerating the training of reinforcement learning models. A GPU with at least 4 GB of VRAM is recommended.

Storage:
An SSD with a minimum of 512 GB storage capacity is recommended for faster data access, model storage, and retrieval.

Internet Connectivity:
A stable and high-speed internet connection is essential for downloading datasets, model updates, and potential cloud-based training or deployment.

Software Requirements:

Operating System:
Linux-based operating systems like Ubuntu 18.04 or higher are preferable for their compatibility with many machine learning libraries and frameworks.

Python:
Python 3.x is the programming language of choice for implementing reinforcement learning algorithms.
Essential libraries include NumPy, TensorFlow (or PyTorch), and OpenAI Gym.

Reinforcement Learning Libraries:
OpenAI Gym: A toolkit for developing and comparing reinforcement learning algorithms.
TensorFlow or PyTorch: Deep learning frameworks for implementing neural network architectures used in reinforcement learning.

Development Environment:
Jupyter Notebook or Visual Studio Code can serve as the integrated development environment (IDE) for coding, testing, and debugging reinforcement learning algorithms.

Simulation Environment (Optional):
If the project involves simulating a grocery shopping environment, platforms like Unity ML-Agents or custom environments within OpenAI Gym can be used.

Version Control:
Git and a version control platform like GitHub for tracking changes, collaborating with team members, and maintaining the project's version history.

Modules:

Simulation Environment Module: This module will contain functions/classes to simulate the environment, including shop availability, pricing, and distances between shops.

Agent Module: This module will contain classes/functions representing the agents, their decision- making processes, and interaction with the environment.

Reward Module: Contains functions/classes to calculate rewards for different actions taken by the agent.
Transition Probability Module: Contains functions/classes to calculate transition probabilities based on actions and outcomes.

Value Iteration Module: Implements the value iteration algorithm to find the optimal policy.

Q-Learning Module: Implements the Q-learning algorithm to learn the optimal policy through trial and error.

Representation Policy Iteration Module: Implements the representation policy iteration algorithm for policy approximation.

6 Proposed System

6.1 Tentative proposed Model

To address the above challenges in grocery shopping faced by the people, this project has proposed a solution. In this we created an environment representing of various shops available for grocery shopping. Each shop will have its own inventory of items along with their availability (whether the item is in stock or not) and pricing patterns (the price of each item). Here we also define the distances between each pair of shops in the environment. This could be based on real-world distances or simplified distances for simulation purposes. And then the state of the agent will include the current shop it is in and the buying status, which indicates whether each item on the shopping list has been bought or not. The agent can take actions to move to different shops based on the current state. The available actions will be a list of shops that the agent can visit from the current shop. The agent will make decisions based on a policy, which determines the action to take given a state. This policy can be learned through reinforcement learning algorithms like Q-learning or policy iteration methods.

If the agent finds the required item in a shop, it receives a positive reward. The reward could be a fixed value or based on the price of the item. If the item is not found in the shop, the agent might receive a penalty to incentivize finding the item elsewhere. The agent might incur a penalty for traveling between shops to encourage minimizing travel distance. The agent might receive a penalty based on the prices of the items bought, encouraging it to find cheaper options.

The random model defines the probabilities of moving from one state to another based on the action taken by the agent and the outcome. The probability of finding an item in a shop could be based on historical data or modeled using a distribution (e.g., Bernoulli distribution). The probability of moving from one shop to another could depend on the distance between the shops and possibly other factors like traffic conditions. The transition probabilities might also consider the prices of items in different shops, favoring shops with lower prices if available and shows the best available option to the user so that they can save time and won’t get confused in selecting the required products from different vendors.

By incorporating these elements into the simulated environment and agent design, you can create a realistic and effective system for optimizing grocery shopping. The rewards and transition probabilities play a crucial role in guiding the agent's decision-making

process, while the environment provides a realistic setting for testing and evaluating different strategies.

6.2 Problem Description

Transition Probabilities and Rewards(for 1 item to buy):

Transition Rules:

- If an item has been bought, indicating the current status as "item bought" (current_status is Y), the session ends.
- If the decision is made to try the same shop again, meaning the action is to stay at the current shop (action is current_shop), the transition probability P depends on whether the item is available in that shop.
- If the decision is made to go to another shop:
- If the next shop is the chosen shop (next_shop is action), the probability of transitioning to that shop is determined by its item availability.
- If the next shop is not the chosen shop (next_shop is not action), the probability of transitioning to that shop depends on its item availability and the preference for transitioning between shops.

Rewards:

- Item Found:
    - When the item is found in the chosen shop, the reward consists of:
    - A reward for purchasing the item.
    - A penalty based on the distance traveled from the current shop to the next shop.
    - A penalty based on the price of the item in the next shop.
    - If the item is not found in the chosen shop, only the distance penalty applies.
- Item Not Found:
    - When the item is found in the next shop while choosing another shop, the reward consists of:
    - A reward for purchasing the item.
    - Penalties based on the distances traveled from the current shop to the chosen shop and from the chosen shop to the next shop.
    - A penalty based on the price of the item in the next shop.
    - If the item is not found in the next shop, only the distance penalty applies.

6.3 Reinforcement Learning Methods

Q-learning:

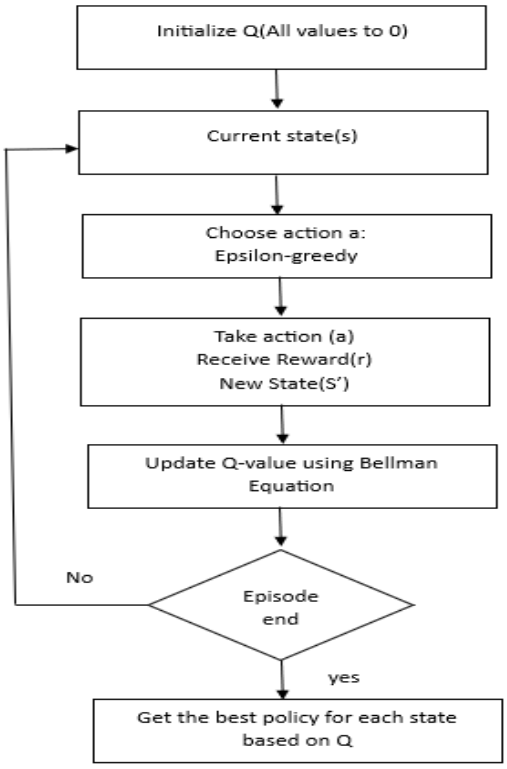


Fig (2) : Implementation of Q_learning Algorithm

- Q-learning is a model-free RL algorithm that learns the optimal action- selection policy for a given environment without requiring a model of the environment's dynamics.
- In the context of grocery shopping optimization, Q-learning is used by the autonomous agent to learn the Q-values, which represent the expected cumulative rewards for taking a particular action in a given state.
- The agent explores the grocery shopping environment by interacting with it, making decisions (actions) based on its current state and receiving feedback (rewards) based on the outcomes of its actions.
- During each interaction, the agent updates its Q-values using the Q- learning update rule, which incorporates the observed reward and the maximum Q-value of the next state.
- Over time, through repeated interactions and updates, the agent learns the optimal Q-values, enabling it to make informed decisions about which actions to take in different states to maximize cumulative rewards, such as cost savings, time efficiency, and customer satisfaction.

Value Iteration:

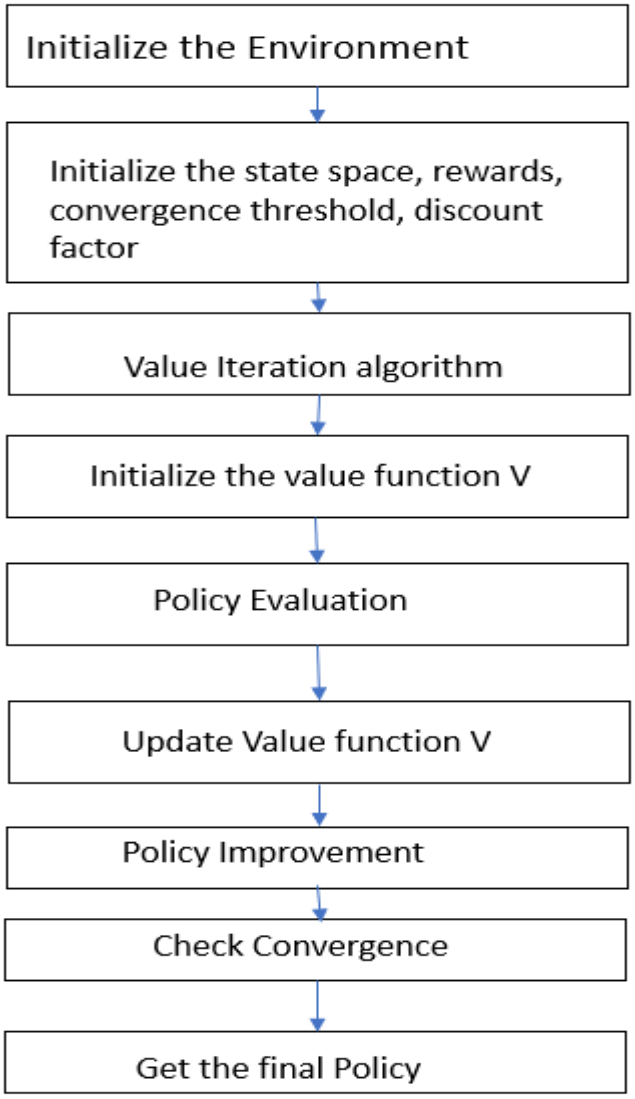


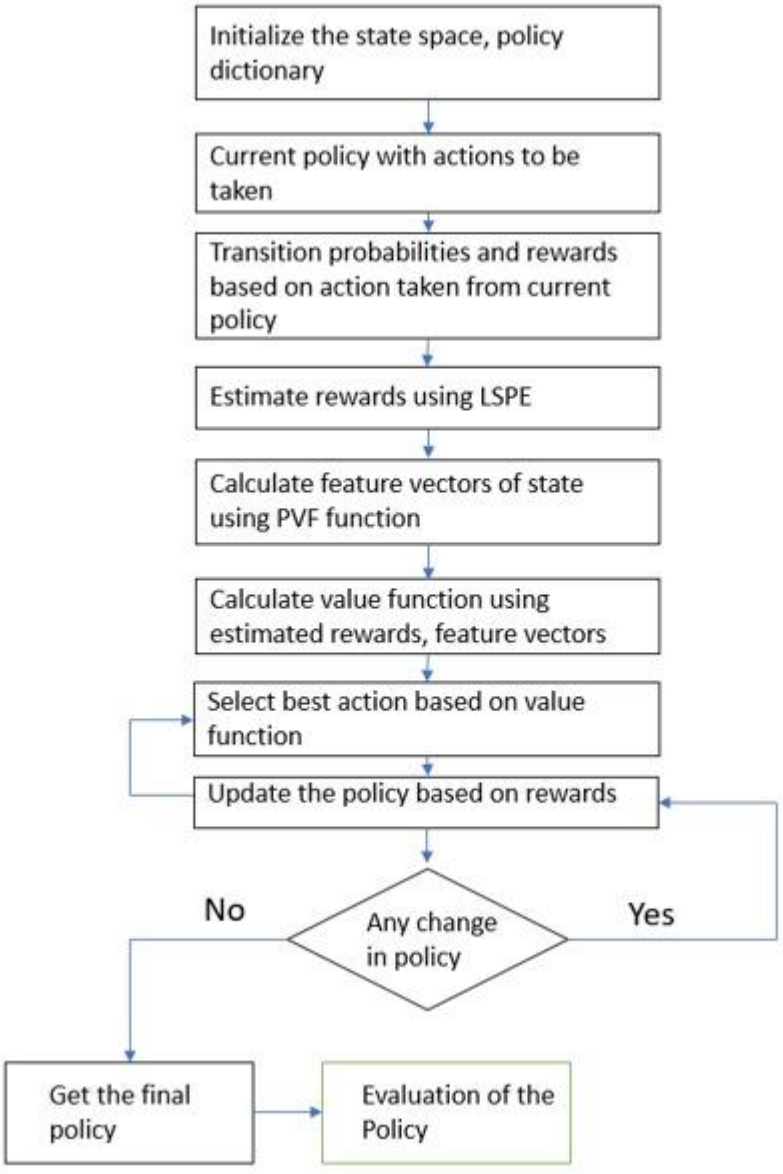
Fig (3). Implementation of Value Iteration Algorithm

- Value iteration is a model-based RL algorithm that iteratively computes the optimal value function for each state in the environment, representing the expected cumulative rewards achievable from that state onwards.
- In the grocery shopping optimization project, value iteration is used to compute the optimal value function by iteratively updating the value

each state based on the Bellman equation, which expresses the relationship between the value of a state and the values of its neighboringstates.

- The agent initializes the value function arbitrarily and then iteratively updates it until convergence, with each iteration bringing the values closer to their optimal values.
- Once the optimal value function is computed, the agent can derive the optimal policy by selecting the action with the highest expected value in each state, enabling it to make decisions that maximize cumulativerewards.
- Value iteration provides a principled approach to finding the optimal policy in environments with known dynamics, making it particularly useful for grocery shopping optimization when the structure of the shopping environment is well-defined.

Random Policy:



The RPI algorithm iteratively improves an agent's policy for navigating an environment for grocery shopping

Initialization:

The algorithm starts by setting up the environment's representation. This includes defining all possible states (combinations of shop statuses and items bought) in the state_space.
A dictionary policy is created to store the current action (shop to visit) for each state. Initially, this policy might be random.
Other necessary parameters like reward for buying an item and discount factor (gamma, not explicitly mentioned here) might also be initialized.

Policy Iteration Loop:

This loop continues until the policy converges (no further improvements are found).
Sample Generation: The agent follows the current policy (policy) by interacting with the environment for a fixed number of steps. This generates a set of samples (samples), which include the state transitions and rewards experienced during these interactions.

Policy Evaluation:

Reward Estimation: A method like Least-Squares Policy Evaluation (LSPE) is used to estimate the expected rewards for each state-action pair based on the collected samples. This helps understand the long-term value of taking specific actions in different states.

Feature Extraction (optional): Feature vectors (phi) are potentially created to represent each state in the samples. These features capture essential information about the state that can be used for better value function estimation.

Value Function Calculation: The estimated rewards (r) are combined with the feature vectors (phi) to calculate the value function (V) for each state. The value function represents the expected future reward the agent can accumulate starting from a particular state.

Policy Improvement:

The algorithm iterates over each state in the state space.
For each state, a function like T(V, state) is used to find the best action based on the estimated value function (V). This action leads to the state with the highest expected future reward.
An improved policy dictionary (next policy) is created, where each state maps to the newly identified best action.

Policy Update (Convergence Check):

The algorithm checks if the improved policy (next policy) differs from the current policy (policy).
If there are no changes (meaning the best actions haven't improved), the loop has converged, and the current policy is considered optimal (within the limitations of the samples and estimations). The algorithm ends.

If any state's policy changed (indicating better actions were found), a flag (flag) is set to continue iterating. The loop repeats from step 2 (Sample Generation) with the improved policy as the new starting point.

Final Policy Update (if converged):

If the loop exits due to convergence, the current policy (policy) is updated to the final improved policy (next policy). This represents the agent's learned policy for navigating the environment.

Evaluation:

At last we make comparison of rewards produced by the random policy, Q learning and Value Iteration Models. We also made the visualization plot of the comparison of rewards between the Q learning and Random policy as one plot and between random policy and value iteration as other plot.

7 Conclusion and Future Work

Conclusion:

In conclusion, the application of reinforcement learning (RL) techniques offers a promising avenue for optimizing the grocery shopping experience. Through iterative interactions with the environment, RL-powered agents can learn and refine optimal strategies tailored to individual preferences, budget constraints, and time limitations. By maximizing cumulative rewards such as cost savings, time efficiency, and customer satisfaction, these agents streamline the shopping journey for consumers and comparing the Q learning and value iteration policy with the random policy , Q learning policy is more optimal and are able to suggest the best action to be taken on the next state.

Future Scope:

Moving forward, the evolution of AI-enabled grocery shopping optimization presents several exciting opportunities for further exploration and development:

1. Enhanced Personalization: Continuously refining RL algorithms to provide even more personalized recommendations, considering factors like dietary preferences, health goals, and cultural backgrounds, to offer a truly tailored shopping experience.

2. Multi-Agent Systems: Advancing research in multi-agent systems to model complex interactions among shoppers, retailers, and other stakeholders, thereby facilitating collaborative optimization strategies and improving overall system efficiency.

3. Real-Time Adaptation: Developing adaptive RL algorithms capable of dynamically adjusting shopping strategies in response to real-time changes in factors such as product availability, pricing fluctuations, and store layouts.

4. Sustainability Integration: Incorporating sustainability metrics into RL frameworks to promote environmentally conscious shopping behaviors, including reducing food waste, selecting eco-friendly products, and optimizing transportation routes to minimize carbon footprint.

5. Integration with E-commerce: Expanding RL-based optimization techniques to online grocery platforms, enabling seamless integration between physical and digital shopping experiences, and leveraging data-driven insights to enhance customer satisfaction and operational efficiency.

By embracing these future directions and continuing to innovate in the field of AI-enabled grocery shopping, we can strive towards creating more efficient, convenient, and sustainable shopping experiences for consumers worldwide.

References

1) Joo, Kwang Hyoun, Tetsuo Kinoshita, and Norio Shiratori. "Agent-based grocery shopping system based on user's preference." *Proceedings Seventh International Conference on Parallel and Distributed Systems: Workshops*. IEEE, 2000.

2) Du, Hongying, and Michael N. Huhns. "A Multiagent system approach to grocery shopping." *Advances on Practical Applications of Agents and Multiagent Systems: 9th International Conference on Practical Applications of Agents and Multiagent Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

3) Joo, Kwang Hyoun, Tetsuo Kinoshita, and Norio Shiratori. "Design and implementation of an agent- based grocery shopping system." *IEICE TRANSACTIONS on Information and Systems* 83.11 (2000): 1940-1951.

4) L. Benedicenti, Xuguang Chen, Xiaoran Cao and R. Paranjape, "An agent-based shopping system," *Canadian Conference on Electrical and Computer Engineering 2004 (IEEE Cat.No.04CH37513)*, Niagara Falls, ON, Canada, 2004, pp. 703-705 Vol.2, doi: 10.1109/CCECE.2004.1345210.

5) Serrano-Hernandez, Adrian, et al. "Agent-based simulation improves E-grocery deliveries using horizontal cooperation." 2020 Winter Simulation Conference (WSC). IEEE, 2020.

6) Schenk, Tilman A., Günter Löffler, and Jürgen Rauh. "Agent-based simulation of consumer behavior in grocery shopping on a regional level." Journal of Business research 60.8 (2007): 894-903.

7) Yu-San Chen, Chang-Franw Lee, Proceedings of the 2009 International Conference on Artificial Intelligence (ICAI'09), “Agent-Based Simulation of Consumer Purchasing Behaviour in aVirtual Environment”.

8) Giuseppe Mangioni, Rosario Sinatra, Vincenzo Nicosia, Vito Latora, Journal of Artificial Societies and Social Simulation, “A Multi-Agent System for Modelling Consumer Behaviour in Supermarkets”.

9) Yuri Kaniovski, Martin Summer, Journal of Artificial Societies and Social Simulation, “Agent- based Modeling of Store Choice Dynamics”.

10) Kozma Zsolt, Máté Csorba, 7th International Conference on Applied Informatics,2015, “Agent- based Simulation of Consumer Behaviour in the Retail Sector”.

11) Michele Piccione, Annual Review of Economics, 2016, “Agent-Based Modeling in Consumer Economics”.

12) Pavle Boskovic, Srdjan Boskovic, Aleksandar Ivanovic, 2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH), “Agent-Based Model of Consumer Decision Making Process in Online Grocery Shopping”.

13) Jungsoo Park, Hyunju Park, 2014 International Conference on Control, Automation and Information Sciences (ICCAIS), “Agent-Based Simulation for Understanding Consumers' Shopping Behavior in Virtual Supermarket”.

Appendix

Source Code

Q-learning:

```
import itertools
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import sys
import copy
import random
#Data
MRPs=dict()
MRPs[0]=20
MRPs[1]=100
MRPs[2]=50
MRPs[3]=50
MRPs[4]=100
MRPs[5]=60
MRPs[6]=35
MRPs[7]=216
MRPs[8]=27
MRPs[9]=130
MRPs[10]=160
MRPs[11]=89
MRPs[12]=73
MRPs[13]=27
MRPs[14]=185
MRPs[15]=249
MRPs[16]=199
MRPs[17]=46
MRPs[18]=55
MRPs[19]=99
'''
MRPs['chocolate']=20
MRPs['shampoo']=100
MRPs['chips']=50
MRPs['pulses']=50
MRPs['apples']=100
MRPs['grapes']=60
MRPs['pepsi']=35
MRPs['bournvita']=216
MRPs['bread']=27
MRPs['jam']=130
MRPs['buscuits']=160
MRPs['butter']=89
MRPs['toothpaste']=73
MRPs['eggs']=27
MRPs['flour']=185
MRPs['peanut_butter']=249
MRPs['cheese']=199
MRPs['milk']=46
MRPs['yoghurt']=55
MRPs['muffins']=99
'''

no_shops=6
no_items=3
```

```
actions=range(no_shops)

reward_buying=50

#Bernoulli variable for each shop
bernoulli=np.random.rand(no_shops,no_items)

#Price bias for each shop
bias=np.random.normal(0,5,no_shops)

#Distance Matrix
a = np.random.uniform(1,10,(no_shops,no_shops))
distance_matrix = np.tril(a) + np.tril(a, -1).T
np.fill_diagonal(distance_matrix,0)

print ('Distance between shops')
print (distance_matrix)
```

```
print ('Distance between shops')
print (distance_matrix)
```

```
Distance between shops
[[0.         7.9503272  9.57588406 7.52851984 5.97505066 3.57858795]
 [7.9503272  0.         1.24896096 7.33679371 9.24425488 2.41718701]
 [9.57588406 1.24896096 0.         2.02242334 3.2972181  8.11955806]
 [7.52851984 7.33679371 2.02242334 0.         1.49670572 8.6462322 ]
 [5.97505066 9.24425488 3.2972181  1.49670572 0.         6.88187458]
 [3.57858795 2.41718701 8.11955806 8.6462322  6.88187458 0.        ]]
```

```
def price_penalty(next_state):
    scaling=0.1
    shop=next_state[0]
    next_status=next_state[1]
    price=0
    for item_no in range(len(next_status)):
        if next_state[1][item_no]:
            price=price+np.random.normal(MRPs[item_no]+bias[shop],1)

    return -price*scaling
```

In [43]:

```
def distance_penalty(distance):
    return -distance
```

In [44]:

```
def availability_in_shop(current_state,next_state):
    old_status=current_state[1]
    new_status=next_state[1]
    next_shop=next_state[0]

    prob=1

    for item_no in range(len(old_status)):
        if old_status[item_no]==0:
            if new_status[item_no]==0:
                prob=prob*(1-bernoulli[next_shop][item_no])
            else:
                prob=prob*bernoulli[next_shop][item_no]
        else:
            if new_status[item_no]==0:
                prob=0

    return prob
```

In [45]:

```
def M(shop_b,shop_a):
    temp=sum(sum(np.triu(distance_matrix)))
    temp2=(temp-distance_matrix[shop_b,shop_a])/((no_shops-1)*temp)
    return temp2
```

In [46]:

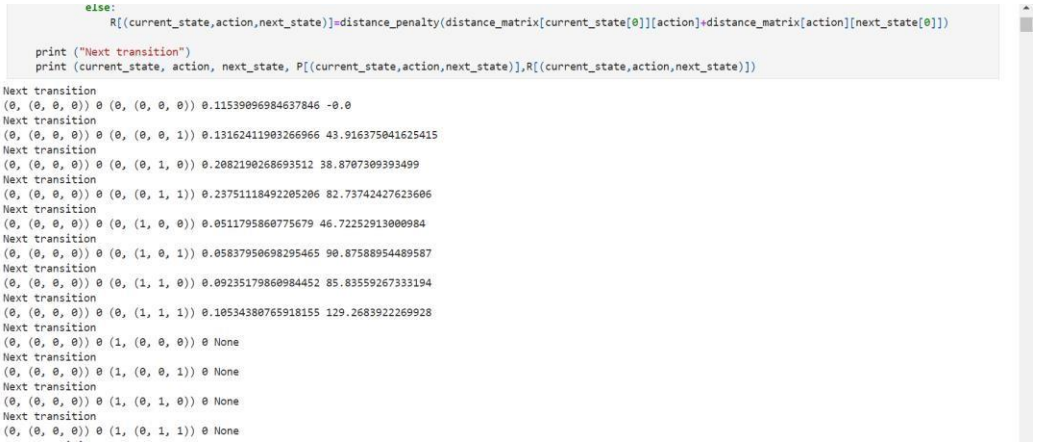
```
#Creating State Space
state_space=[]
all_possible_buying_statuses=list(itertools.product([0, 1], repeat=no_items))

for shop_no in range(no_shops):
    for buying_status in all_possible_buying_statuses:
        state=(shop_no,buying_status)
        state_space.append(state)

print ('State Space Size')
print (len(state_space))
```

```
        state_space.append(state)

print ('State Space Size')
print (len(state_space))
```

```
State Space Size
48
```

```
#Defining Transition Probabilities and Rewards
P=dict()
R=dict()

actions=range(no_shops)
for current_state, action, next_state in list(itertools.product(state_space,actions,state_space)):
    if current_state[1]==tuple(np.ones(no_items)):
        P[(current_state,action,next_state)]=0
        R[(current_state,action,next_state)]=None
        continue

    if action == current_state[0]: #action==current_shop
        if next_state[0]!=current_state[0]:
            P[(current_state,action,next_state)]=0
            R[(current_state,action,next_state)]=None
        else:
            P[(current_state,action,next_state)]=availability_in_shop(current_state,next_state)
            bought_items=sum(next_state[1])
            if bought_items>0:

R[(current_state,action,next_state)]=bought_items*reward_buying+distance_penalty(distance_m
atrix[current_state[0]][next_state[0]])+price_penalty(next_state)
        else:

R[(current_state,action,next_state)]=distance_penalty(distance_matrix[current_state[0]][next_sta
te[0]])
    else:
        if next_state[0]==action:
            P[(current_state,action,next_state)]=0.9*availability_in_shop(current_state,next_state)
            bought_items=sum(next_state[1])
            if bought_items>0:

R[(current_state,action,next_state)]=bought_items*reward_buying+distance_penalty(distance_m
atrix[current_state[0]][next_state[0]])+price_penalty(next_state)
        else:

R[(current_state,action,next_state)]=distance_penalty(distance_matrix[current_state[0]][next_sta
te[0]])
        else:

P[(current_state,action,next_state)]=0.1*availability_in_shop(current_state,next_state)*M(next_st
ate[0],current_state[0])
        bought_items=sum(next_state[1])
```

```
        if bought_items>0:

R[(current_state,action,next_state)]=bought_items*reward_buying+distance_penalty(distance_m
atrix[current_state[0]][action]+distance_matrix[action][next_state[0]])+price_penalty(next_state
)
        else:

R[(current_state,action,next_state)]=distance_penalty(distance_matrix[current_state[0]][action]+
distance_matrix[action][next_state[0]])

    print ("Next transition")
    print (current_state, action, next_state,
P[(current_state,action,next_state)],R[(current_state,action,next_state)])
```




```
def takeaction(current_state,action):
    global P
    global R
    global state_space
    r = random.random()
    for next_state in state_space:
        if(r<=0):
            break
        r-=P[(current_state,action,next_state)]
    return next_state, R[(current_state,action,next_state)]
```

In [49]:

```
def e_greedy(e,Q_s):
    x=random.randrange(1,11)
    if x<=e*10:
        return random.randrange(no_shops)
    else:
        return np.argmax(Q_s)
```

In [50]:

```
def q_learning(no_episodes,no_steps,alpha,discount,epsilon):
    Q=dict()
    Rewards=[]

    for e in range(no_episodes):
        S=random.choice(state_space)
        step=0
        Episode_Reward=0

        while(step<no_steps):
            if S not in Q.keys():
                Q[S]=np.zeros(no_shops).astype(int)
            if S[1]==tuple(np.ones(no_items)):
                break

            A=e_greedy(epsilon,Q[S])
```

```
        S_,r=takeaction(S,A)

        if S_ not in Q.keys():
            Q[S_]=np.zeros(no_shops).astype(int)

        A_=np.argmax(Q[S_])

        if r==None:
            r=0
        Q[S][A]=Q[S][A]+alpha*(r+discount*Q[S_][A_]-Q[S][A])
        S=S_
        step+=1
        Episode_Reward+=r

    Rewards.append(Episode_Reward)
  return Q,Rewards
```

In [51]:

```
result_Q,Rewards=q_learning(1000,200,0.1,0.9,0.5)
```

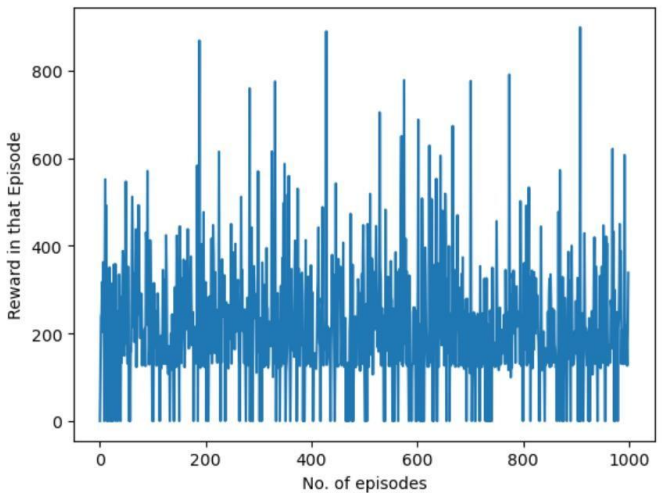
In [52]:

```
print ("Best Policy for each state")
print ("")
best_policy=dict()
for state in result_Q:
  best_action=np.argmax(result_Q[state])
  status=state[1]
  if status==tuple(np.ones(no_items)):
    best_action="End"
  best_policy[state]=best_action
  print ("State",state, "Best Action",best_action)


print ("")
print ("Rewards")
plt.plot(Rewards)
plt.xlabel('No. of episodes')
plt.ylabel('Reward in that Episode')
plt.show()
```

```
plt.xlabel('No. of episodes')
plt.ylabel('Reward in that Episode')
plt.show()
```

```
Best Policy for each state

State (2, (1, 1, 1)) Best Action End
State (0, (0, 1, 0)) Best Action 2
State (2, (1, 0, 0)) Best Action 5
State (3, (0, 0, 0)) Best Action 2
State (0, (0, 0, 1)) Best Action 1
State (1, (1, 1, 0)) Best Action 5
State (3, (1, 1, 1)) Best Action End
State (1, (0, 1, 1)) Best Action 0
State (0, (1, 0, 0)) Best Action 5
State (5, (1, 1, 1)) Best Action End
State (5, (1, 0, 0)) Best Action 3
State (2, (1, 0, 1)) Best Action 1
State (1, (0, 0, 0)) Best Action 5
State (5, (1, 1, 0)) Best Action 3
State (4, (1, 1, 1)) Best Action End
State (2, (0, 0, 0)) Best Action 1
State (5, (0, 1, 0)) Best Action 0
State (3, (0, 1, 1)) Best Action 0
State (4, (0, 0, 0)) Best Action 0
State (5, (0, 0, 0)) Best Action 2
State (1, (1, 0, 1)) Best Action 5
State (1, (1, 0, 0)) Best Action 2
State (2, (1, 1, 0)) Best Action 5
```

```
State (4, (0, 0, 1)) Best Action 0
State (3, (1, 1, 0)) Best Action 2

Rewards
```

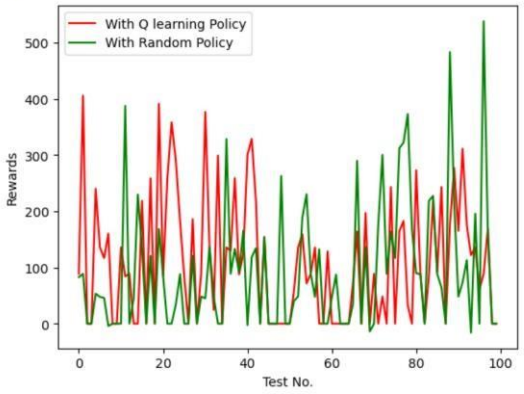



```
Rewards_Qpolicy=[]
Rewards_RandomPolicy=[]
for test_no in range(100):
    print ("Test no.",test_no)
    S=random.choice(state_space)
    S_random=S
    step=0
    r=0
    r_random=0

    while(S[1]!=tuple(np.ones(no_items)) and step<200):
        A=best_policy[S]
        S_,rew=takeaction(S,A)
        S=S_
        rew=R[(S,A,S_)]
        #print S,A,S_,rew
        if rew==None:
            break
        r+=rew
        step+=1
    Rewards_Qpolicy.append(r)

    step=0
    while(S_random[1]!=tuple(np.ones(no_items)) and step<200):
        A_random=random.choice(actions)
        S_random_,rew=takeaction(S_random,A_random)
        S_random=S_random_
        rew=R[(S_random,A_random,S_random_)]
        #print S_random,A_random,S_random_,rew

        if rew==None:
            break
        r_random+=rew
        step+1
    Rewards_RandomPolicy.append(r_random)

print ("")
print ("Rewards  ")
plt.plot(Rewards_Qpolicy,color='r',label='With Q learning Policy')
plt.plot(Rewards_RandomPolicy,color='g',label='With Random Policy')
plt.xlabel('Test No.')
plt.ylabel('Rewards')
plt.legend()
plt.show()
```

```
print ("")
print ("Rewards ")
plt.plot(Rewards_Qpolicy,color='r',label='With Q learning Policy')
plt.plot(Rewards_RandomPolicy,color='g',label='With Random Policy')
plt.xlabel('Test No.')
plt.ylabel('Rewards')
plt.legend()
plt.show()
```

```
Test no. 0
Test no. 1
Test no. 2
Test no. 3
Test no. 4
Test no. 5
Test no. 6
Test no. 7
Test no. 8
Test no. 9
Test no. 10
Test no. 11
Test no. 12
Test no. 13
Test no. 14
Test no. 15
Test no. 16
Test no. 17
Test no. 18
Test no. 19
Test no. 20
Test no. 21
```

Rewards



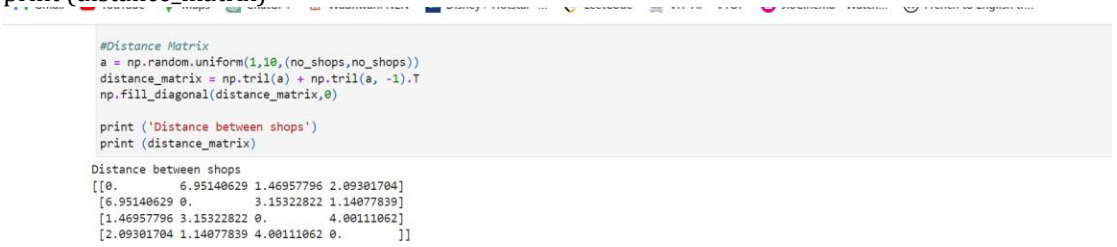
Source code for value iteration:

```
import itertools
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import sys
import copy
import random
```

In [2]:

```
#Data
MRPs=dict()
MRPs[0]=20
MRPs[1]=100
MRPs[2]=50
MRPs[3]=50
MRPs[4]=100
MRPs[5]=60
MRPs[6]=35
MRPs[7]=216
MRPs[8]=27
MRPs[9]=130
MRPs[10]=160
MRPs[11]=89
MRPs[12]=73
MRPs[13]=27
MRPs[14]=185
MRPs[15]=249
MRPs[16]=199
MRPs[17]=46
MRPs[18]=55
```

```
MRPs[19]=99
'''
MRPs['chocolate']=20
MRPs['shampoo']=100
MRPs['chips']=50
MRPs['pulses']=50
MRPs['apples']=100
MRPs['grapes']=60
MRPs['pepsi']=35
MRPs['bournvita']=216
MRPs['bread']=27
MRPs['jam']=130
MRPs['buscuits']=160
MRPs['butter']=89
MRPs['toothpaste']=73
MRPs['eggs']=27
MRPs['flour']=185
MRPs['peanut_butter']=249
MRPs['cheese']=199
MRPs['milk']=46
MRPs['yoghurt']=55
MRPs['muffins']=99
'''

no_shops=4
no_items=2

actions=range(no_shops)

reward_buying=50

#Bernoulli variable for each shop
bernoulli=np.random.rand(no_shops,no_items)

#Price bias for each shop
bias=np.random.normal(0,5,no_shops)

#Distance Matrix
a = np.random.uniform(1,10,(no_shops,no_shops))
distance_matrix = np.tril(a) + np.tril(a, -1).T
np.fill_diagonal(distance_matrix,0)

print ('Distance between shops')
print (distance_matrix)
```




```
def price_penalty(next_state):
    scaling=0.1
    shop=next_state[0]
    next_status=next_state[1]
    price=0
    for item_no in range(len(next_status)):
        if next_state[1][item_no]:
            price=price+np.random.normal(MRPs[item_no]+bias[shop],1)

    return -price*scaling
```

In [4]:

```
def distance_penalty(distance):
    return -distance
```

In [5]:

```
def availability_in_shop(current_state,next_state):
    old_status=current_state[1]
    new_status=next_state[1]
    next_shop=next_state[0]

    prob=1

    for item_no in range(len(old_status)):
        if old_status[item_no]==0:
            if new_status[item_no]==0:
                prob=prob*(1-bernoulli[next_shop][item_no])
            else:
                prob=prob*bernoulli[next_shop][item_no]

    return prob
```

In [6]:

```
def PVF(num_states,samples,k,state_space_enumeration):
    G = np.zeros((num_states,num_states))
    for sample in samples:
        G[state_space_enumeration[sample[0]]][state_space_enumeration[sample[3]]]+=1
    denominator = np.sum(G, axis = 1)
    P = np.zeros((num_states,num_states))
    for i in range(num_states):
        for j in range(num_states):
            if(denominator[i]!=0):
                P[i,j]=G[i,j]/denominator[i]
    evals, evecs = np.linalg.eigh(P)
    idx  = evals.argsort()[::-1]
    evals = evals[idx]
    evecs = evecs[idx]
    perron_vec = evecs[-1]
    D = np.zeros((num_states,num_states))
    for i in range(num_states):
        D[i][i] = perron_vec[i]
    L = D - (D.dot(P)+P.T.dot(D))/2.0
    evals,evecs = np.linalg.eigh(L)
    idx = evals.argsort()[::-1]
    evals = evals[idx]
    evecs = evecs[idx]
    phi = np.array([evecs[i] for i in range(k)]).T
    print( phi[0,:][np.newaxis].T.shape)
    return phi
```

In [7]:

```
def  LSPE(num_states,samples,phi,state_space_enumeration,  alpha):
    r = np.zeros(phi.shape[1])[np.newaxis].T
    for k in range(len(samples)):
        C = np.zeros((phi.shape[1],phi.shape[1]))
        d = np.zeros(phi.shape[1])[np.newaxis].T
        for i in range(k):
            C+=
phi[state_space_enumeration[samples[i][0]],:][np.newaxis].T.dot(phi[state_space_enumeration[s
amples[i][0]],:][np.newaxis])
            q = phi[state_space_enumeration[samples[i][0]],:][np.newaxis].dot(r) -
alpha*phi[state_space_enumeration[samples[i][3]],:][np.newaxis].dot(r) + samples[i][2]/60.0
            d+= phi[state_space_enumeration[samples[i][0]],:][np.newaxis].T*q
        r_next = r - np.linalg.pinv(C).dot(d)
        if(all(i<0.1 for i in np.abs(r_next-r))):
            break
    return r
```

In [8]:

```
def sampling(policy, k):
    global state_space
    samples = []
    current_state = random.choice(state_space)
    for i in range(k):
        sample=[]
        next_state,reward=takeaction(current_state,policy[current_state])
        sample.append(current_state)
        sample.append(policy[current_state])
        sample.append(reward)
        samples.append(next_state)
        if(not all(i==0 for i in (next_state[1]-np.ones(no_items)))):
            current_state=next_state
        else:
            current_state=random.choice(state_space)
    return samples
```

In [9]:

```
def RPI():
    global state_space
    state_space_enumeration =dict()
    i=0
    for state in state_space:
        state_space_enumeration[state]=i
        i+=1
    nextpolicy = dict()
    policy=dict()
    for state in state_space:
        nextpolicy[state] = random.randint(0,no_shops)
        nextpolicy[state] = random.randint(0,no_shops)
    flag=True
    while(flag):
        flag = False
        policy = copy.deepcopy(nextpolicy)
        samples = sampling(policy,10000)
        phi = PVF(len(state_space),samples,20,state_space_enumeration)
        r = LSPE(len(state_space),samples,phi,state_space_enumeration,0.7)
        V = phi.dot(r)
        for state in state_space:
            P, nextpolicy[state] = T(V,state)
        for state in state_space:
            if(policy[state]!=nextpolicy[state]):
                flag=True
```

In [10]:

```
def M(shop_b,shop_a):
    temp=sum(sum(np.triu(distance_matrix)))
    temp2=(temp-distance_matrix[shop_b,shop_a])/((no_shops-1)*temp)
    return temp2
```

In [11]:

```
def T(V,current_state):
    alpha = 0.9
    min_action = sys.maxsize
    min_V=float('inf')
    for action in actions:
        summ=0
        for next_state in state_space:
            if(P[(current_state,action,next_state)]!=0):
                summ += (-R[(current_state,action,next_state)]/500.0 +
alpha*V[next_state])*P[(current_state,action,next_state)]
        # print summ
        if (summ<min_V):
            min_V=summ
            min_action=action
    return min_V,min_action
```

In [12]:

```
#Creating State Space
state_space=[]
all_possible_buying_statuses=list(itertools.product([0, 1], repeat=no_items))

for shop_no in range(no_shops):
  for buying_status in all_possible_buying_statuses:
    state=(shop_no,buying_status)
    state_space.append(state)

print ('State Space')
print (len(state_space))
```

```
    for buying_status in all_possible_buying_statuses:
        state=(shop_no,buying_status)
        state_space.append(state)

print ('State Space')
print (len(state_space))
```

```
State Space
16
```

```
#Defining Transition Probabilities and Rewards
P=dict()
R=dict()

actions=range(no_shops)
for current_state, action, next_state in list(itertools.product(state_space,actions,state_space)):
  if current_state[1]==tuple(np.ones(no_items)):
    P[(current_state,action,next_state)]=0
    R[(current_state,action,next_state)]=None
    continue

  if action == current_state[0]: #action==current_shop
    if next_state[0]!=current_state[0]:
      P[(current_state,action,next_state)]=0
      R[(current_state,action,next_state)]=None
    else:
      P[(current_state,action,next_state)]=availability_in_shop(current_state,next_state)
      bought_items=sum(next_state[1])
      if bought_items>0:

R[(current_state,action,next_state)]=bought_items*reward_buying+distance_penalty(distance_m
atrix[current_state[0]][next_state[0]])+price_penalty(next_state)
      else:

R[(current_state,action,next_state)]=distance_penalty(distance_matrix[current_state[0]][next_sta
te[0]])
  else:
    if next_state[0]==action:
      P[(current_state,action,next_state)]=0.9*availability_in_shop(current_state,next_state)
      bought_items=sum(next_state[1])
      if bought_items>0:

R[(current_state,action,next_state)]=bought_items*reward_buying+distance_penalty(distance_m
atrix[current_state[0]][next_state[0]])+price_penalty(next_state)
      else:

R[(current_state,action,next_state)]=distance_penalty(distance_matrix[current_state[0]][next_sta
te[0]])
    else:

P[(current_state,action,next_state)]=0.1*availability_in_shop(current_state,next_state)*M(next_st
ate[0],current_state[0])
      bought_items=sum(next_state[1])
      if bought_items>0:

R[(current_state,action,next_state)]=bought_items*reward_buying+distance_penalty(distance_m
```

```
atrix[current_state[0]][action]+distance_matrix[action][next_state[0]])+price_penalty(next_state
)
        else:

R[(current_state,action,next_state)]=distance_penalty(distance_matrix[current_state[0]][action]+
distance_matrix[action][next_state[0]])

    #print "Next transition:"
    #print "Current State, Action, Next State, P, R"
    #print current_state, action, next_state,
P[(current_state,action,next_state)],R[(current_state,action,next_state)]
```

In [14]:

```
def value_iteration(state_space):
    threshold = 1
    V = dict()
    next_V = dict()
    policy = dict()
    for state in state_space:
        V[state] = 0.0
        next_V[state] = 0.0
        policy[state] = -1
    flag = True
    while(flag):
        flag = False
        for current_state in state_space:
            next_V[current_state], policy[current_state] = T(V, current_state)
            if next_V[current_state] - V[current_state] > threshold:
                flag = True
            V[current_state] = next_V[current_state]
    return V, policy
```

In [15]:

```
def takeaction(current_state, action, state_space):
    global P
    global R
    r = random.random()
    next_state = None
    reward = None

    for next_state_candidate in state_space:
        if r <= 0:
            break
        r -= P[(current_state, action, next_state_candidate)]
        next_state = next_state_candidate

    if next_state is not None:
        reward = R[(current_state, action, next_state)]

    return next_state, reward
```

In [19]:

```
ran = []
vi = []
c1 = 0
c2 = 0

for i in range(1000):
    try:
        J, policy = value_iteration(state_space)
        print(J)
        print(policy)

        summ1 = 0
        summ2 = 0
```

```
        current_state = (0, (0, 0))

        while current_state[1] != (1, 1):
            action = random.randint(0, no_shops - 1)
            next_state, reward = takeaction(current_state, action, state_space)
            print(current_state, action, next_state)
            summ1 += reward  # Accumulate reward directly
            current_state = next_state

        current_state = (0, (0, 0))

        while current_state[1] != (1, 1):
            next_state, reward = takeaction(current_state, policy[current_state], state_space)
            summ2 += reward # Accumulate reward directly
            current_state = next_state

        c1 += summ1
        c2 += summ2

        ran.append(summ1)
        vi.append(summ2)

    except Exception as e:
        print("Exception:", e)

plt.plot(ran, label='Random', color='green')
plt.plot(vi, label='Value Iteration', color='red')
plt.legend()
plt.show()
```