

Modeling decision-making with intelligent agents to aid rural commuters in developing nations.

Patricio Julián Gerpe¹, Evangelos Markopoulos²

¹ Enpov, Silk Lafto, Kebele 02, New, 5600, Addis Ababa, Ethiopia.

² HULT International Business School, 35 Commercial Road, Whitechapel, E1 1LD London United Kingdom.

patricio@enpov.com, evangelos.markopoulos@faculty.hult.edu

Abstract. More than a billion rural merchants in the developing world depend on hiring on-demand transportation services to commute people or goods to markets. Selecting the optimal fare involves decision-making characterized by multiple alternatives and competing criteria. Decision support systems are used to solve this. However, those systems are based on object-based approaches which lack the high-level abstractions needed to effectively model and scale human-machine communication. This paper introduces AopifyJS, a novel agent-based decision-support tool. We developed a two-agent simulation. One agent makes a request, then another takes a dataset of a stratified sample of 104 Ethiopian commuter criteria preferences and a dataset of fare alternatives. The second agent computes HPA and TOPSIS algorithms to weight, score, rank those alternatives. Once we run the simulation, it returns an interpretable prescription to the first agent, storing all interactions in an architecture that allows developers to program further customization as interactions scale.

Keywords: Agent-based modeling · Agent-oriented programming · Multi-criteria decision-making · TOPSIS · Social innovation · Interpretable Artificial Intelligence

1 Introduction

Rural transportation has been frequently regarded as a critical infrastructure for economic development. [1], [2]. According to the latest Rural Access Index of the World Bank, over a billion people has not access to all-weather roads in a range of fewer than two kilometres [3]. In Ethiopia, rural commuting has its own challenges. One critical job for rural merchants is selecting the most convenient fare alternative to get them to the market. Several factors influence the merchant's decision making (DM), such as price, time and the trust they have in the driver. This type of problem can be classified as a multi-criteria decision making (MCDC) problem. MCDC is a subject that is a common concern for researchers in the field of decision science [4], [5].

This concept refers to a type of problem where an agent is required to make a choice based on different conflicting criteria. Different techniques have been used to address

these problems such as AHP, TOPSIS, ELECTRE, and PROMETREE [6], [7], [8], [9]. Those techniques are indeed useful to build the so-called ‘decision-support systems’. We claim that traditional object-oriented paradigms, due to the lack of high-level abstractions, are quite limited and unscalable when it comes to model human communication. That is why we propose to introduce a novel approach to tackle DM problems combining agent-oriented software engineering (AOSE) with MCDC methods.

Our approach, which we named AopifyJS, is the first AOP open-source framework written in Javascript language to fully integrate methods widely used in MCDM. For the sake of clarity, this paper is structured as follows: we will first detail the problem that rural merchants face in Ethiopia; then we will provide a literature review of MCDC and AOSE scientific literature; after that, we will detail our agent-based tool; and finally we will run our two-agent simulations based on the exemplary case given in this paper.

2 Field Study: Inefficiencies in Commuting in Rural Areas

Ethiopia has a total population of around 104 million people [10], where 70% of the population relies on agriculture. 36.7% of GDP comes from agriculture [11]. Yet, the country deals with critical challenges in transportation infrastructure.

Smallholder farmers commonly sell their produce to the internal market. To do that, rural merchants are required to wait on rural roads for small motorized vehicles called ‘Bajajs’ or sometimes simply horse-drawn carts, which regularly pass down rural roads seeking customers. We first conducted preliminary field observations with the purpose of acquiring a first-hand understanding of the commuting process of farmers in Ethiopia.

During 2017 and 2018, we spent two days in Wacho Belisso, Ethiopia, and seven days in the Duken Region where we interviewed a sample of 28 rural commuters asking about their commuting routine.

The key insights from those observations were that (A) on average, transportation seems to cost 30% of the farmers’ income and, (B) commuters usually go to main roads, waiting near KM markers for vehicles to pass by.

Apart from the field interviews, we visited different institutions in Ethiopia to contrast information from the ATA (Agricultural Transformation Agency), the Transportation Bureau, and the national agricultural ministry in the city of Addis Ababa. In summary, rural commuters are often immersed in complex decision-making environments constrained by scarce resources and multiple alternative courses of actions wherein information is often incomplete and outcomes are difficult to predict.

Figure 1 illustrates the whole process. As already outlined, in this paper we will be focusing on solving the second step of this flow chart.

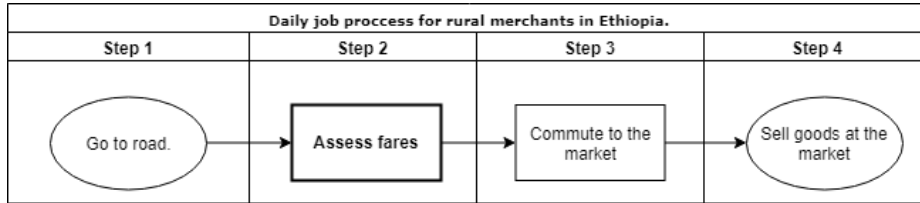


Fig. 1. Commuting process flowchart.

3 Literature Review

3.1 Overview of Agent-oriented Software Engineering Research

The term ‘agent-oriented programming’ was formerly coined by computer scientist Soham in 1993 [13]. This concept refers to the developing paradigm in which the center of applications are agent entities. Those agent entities provide to the developer high-level abstractions which include but are not limited to: beliefs, motives, skills, and policies.

Table 1 describes Shoham AOP definition, as an extension of Object-Oriented programming OOP [14].

Table 1. Table of comparison. OOP and AOP.

| Framework | OOP | AOP |
|------------------------------------------|----------------------------------------|----------------------------------------------|
| Basic unit: | Object | Agent |
| Parameters defining state of basic unit. | Unconstrained | Beliefs, commitments, capabilities, choices. |
| Process of computation: | Messages passing and response methods. | Message passing and response methods. |
| Types of message: | Unconstrained | Inform, request, offer, promise, decline. |
| Constraints on methods: | None | Honesty, consistency. |

Agents have the following properties: autonomy, social, reactivity, and proactivity. This means that an agent’s memory is self-contained; they have a language in which they can share information with other agents; they are proactive in that they pursue goals and act to achieve them; and they are reactive in that they produce new actions in response to stimuli in a given environment.

In Javascript, some *general-purpose* attempts to create frameworks have already been made. For instance, SieborgJS [15], is a proposed framework for multi-agent simulations in Javascript. Nonetheless, when reviewing the available open code repositories, the code does not appear to be open-source. AgentSimJS [16] is a comprehensive framework for 3D spatial agent simulations and EveJS [17] is a framework to build multi-agent systems in web ecosystems, integrated with websockets, https, http and JSON RPC protocols.

These last two frameworks do have an accessible open-source code. However, based on its commit history, we deduce that those projects were merely academic and are no longer actively maintained or tested. Unlike those attempts, we seek to build a scalable toolkit that not only solves problems such as the one outlined in this paper but can also turn into a standard in the IT industry, shifting object and function-oriented programming into an approach specifically designed to consider human factors when aiding decision problems. Apart from that, our proposed framework is the only one in this language to provide integration with MCDC algorithms.

3.2 Overview of Multi-criteria Decision Making Literature

Multi-criteria decision making (MCDM) is an area that has caught the attention of scholars from fields such as operation research, ergonomics, management, and decision science. Stanley Zionts popularized the acronym MCDM back in 1979 [18].

Throughout recent decades, several methods have been proposed. A number of researchers have compared these methods [19]. The most common methods include but are not limited to: AHP, ANP, TOPSIS, ELECTRE, PROMETHEE, and VIKOR. In recent years, with the increase in computing power and the availability of data, machine learning techniques have been widely incorporated into the creation of decision support systems, especially optimization algorithms such as Markov chains, naive Bayes and decision trees that had been used in the so-called recommender systems [20]. To tackle our problem we will use AHP and TOPSIS algorithms.

The Analytic Hierarchy Process (AHP) was proposed by Saaty in 1979. Essentially, the algorithm takes a matrix of comparative judgment between competing criteria, it normalizes that matrix and then computes an eigenvector considering each column data. That eigenvector represents the weighting of preference that each criterion has in the decision-maker.

The Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) was proposed by Hwang and Yoon in 1981. The algorithm is based on the premise that the most desirable solution out of a set of alternatives is the one with the closest Euclidean distance to the hypothetical ideal solution and the farthest Euclidean distance to the hypothetical anti-ideal solution.

4 AopifyJS: Our Agent-based Tool

Unlike previous attempts, our tool integrates MCDC methods for prescriptive analytics with agent-based modeling (ABM). Due to its high-level abstractions, we argue that this ABM design allows the early incorporation of ergonomics and human factors when it comes to deploying decision-support systems.

We decided to build our system in Javascript because (A) it has extensive and on-growing support from the developer community, (B) it allows developers with cross-platform developments and (C) its built-in asynchronous methods can be reutilized for our event-based communication engine.

To tackle our fare-selection problem, our datasets include: (A) one matrix containing fictional data on six available drivers and its core attributes (time, price and trust – where trust is an integer number from 1 to 5 representing a rating system such as those commonly used in on-demand reputation systems) and (B) a dataset of 104 matrixes that contains an assessment of those three criteria based on a comparative subjective judgment from real rural merchants we interviewed in Ethiopia. In other words, each farmer we interviewed compared each criterion against the others. To collect that data, we used a simplified version of the Saaty Scale [21].

Table 2. Fare alternative matrix.

| Alternative | Time | Price | Trust |
|-------------|------|-------|-------|
| Driver 1 | 2 | 5 | 5 |
| Driver 2 | 60 | 26 | 4 |
| Driver 3 | 20 | 20 | 4 |
| Driver 4 | 500 | 2 | 4 |
| Driver 5 | 50 | 23 | 3 |
| Driver 6 | 25 | 10 | 1 |

Table 2 illustrates our alternative matrix. To build our dataset of criteria assessment we asked 104 rural merchants in the Dukem region, through direct field interviews, the three questions below:

- Is price more important for you than time?
- Is price more important for you than trust?
- Is time more important for you than trust?

Considering cognitive and educative aspects of our respondents, we used a simplified version of a Saaty questionnaire to ensure greater enthusiasm and comprehension by them.

For each interview we built a matrix and we inserted that matrix into a new row of a csv file.

| OBSERVATIONS | | | | | | | | | |
|--------------|--------------|---------------|---------------|---------------|----------------|----------------|---------------|----------------|----------------|
| # | Time vs Time | Time vs Price | Time vs Trust | Price vs Time | Price vs Price | Price vs Trust | Trust vs Time | Trust vs Price | Trust vs Trust |
| 1 | 1,00 | 3,00 | 3,00 | 0,33 | 1,00 | 3,00 | 0,33 | 0,33 | 1,00 |
| 2 | 1,00 | 3,00 | 0,33 | 0,33 | 1,00 | 0,20 | 3,00 | 5,00 | 1,00 |
| 3 | 1,00 | 3,00 | 3,00 | 0,33 | 1,00 | 0,20 | 0,33 | 5,00 | 1,00 |
| 4 | 1,00 | 3,00 | 0,33 | 0,33 | 1,00 | 5,00 | 3,00 | 0,20 | 1,00 |
| 5 | 1,00 | 3,00 | 0,33 | 0,33 | 1,00 | 1,00 | 3,00 | 1,00 | 1,00 |
| 6 | 1,00 | 3,00 | 1,00 | 0,33 | 1,00 | 5,00 | 1,00 | 0,20 | 1,00 |
| 7 | 1,00 | 0,33 | 3,00 | 3,00 | 1,00 | 5,00 | 0,33 | 0,20 | 1,00 |
| 8 | 1,00 | 3,00 | 3,00 | 0,33 | 1,00 | 5,00 | 0,33 | 0,20 | 1,00 |
| 9 | 1,00 | 1,00 | 3,00 | 1,00 | 1,00 | 0,20 | 0,33 | 5,00 | 1,00 |
| 10 | 1,00 | 3,00 | 3,00 | 0,33 | 1,00 | 0,20 | 0,33 | 5,00 | 1,00 |

Fig. 2. Dataset: Structure of the criteria assessment dataset.

Therefore, our csv dataset file looks as Figure 2 illustrates. The full dataset is available in our open-source repository in Github. If the value is one, that means the criteria was perceived to have equal preference against the other.

A number higher than one means higher preference and, a lower one means lower preference. Afterwards, we created an agent entity that incorporated both AHP and TOPSIS algorithms in its methods in order to aid fare requests from rural merchants. Our agent entities and methods are mainly inspired in EveJS, yet, tailored to facilitate decision-making like the one dealt by merchants.

```
class Agent extends EventEmitter {
  constructor(name) {
    super();
    this.id = uuid();
    this.name = name;
    this.isAlive = false;
    this.interactions = new Set();
  }
}
```

Our agents communicate with each other via an event-based architecture. That is why we are utilizing the built-in module of NodeJS: ‘EventEmitter’. Each agent has a name (string) and an id (uuid). The ‘isAlive’ property indicates whether the agent has been initialized. If the agent is not initialized then it cannot receive/send any message. Each time our agent interacts with another agent, we will store that information in a Javascript Set. This architecture, unlike traditional non-agent approaches it is self-contained in each agent, allowing developers to provide further user customization and user experience optimization as we scale.

The Agent class has two elemental methods to initialize or deinitialize the agent (start and kill) and four methods that allow it to communicate and capture data from its environment: (tell, on, store and decide) where ‘on’ is already inherent in the EventEmitter class. Table 3 illustrates all these methods:

Table 3. Agent class methods.

| Method | Purpose | Use case |
|----------|---------------------------------------------------------------|------------------------------------------------|
| decide() | To enable agents to resolve complex decision-making problems. | Agent aiding another agent to make a decision. |
| tell() | To enable communication between agents. | Agent sending its insights to another agent. |
| on() | To enable agents to capture stimuli from environment. | Agent waiting for another agent's information. |
| store() | To enable agents to store data about their interactions | Agent storing the information it received. |

All these methods are written in the most declarative way possible, allowing human interpretability not just from the user-side but also from the developer side. Start() is the method that initializes our agent so it starts listening to the events to which it is subscribed. Kill() is the method that deletes our agent from its environment. On() is the method of subscribing to one particular event. Tell() is the method to send a particular message to another agent. Store() is used to tell the agent to store information from a message it received, and Decide() is the method that imports a prescriptive analytics function to solve a problem of MCDC.

Our agent system is prepared to compute AHP algorithm for criteria weighting and TOPSIS for alternative selection. Both came from a self-coded repository (Ahp-lite and Topsis named respectively) available in NPM package management system of the NodeJS server-side javascript framework.

Consequently, we can proceed to set-up our simulation. We initialize two agent entities. One represents a rural merchant in Ethiopia that needs to go to market and the other one represents a computer agent that handles his or her request.

```
const ai = new Agent('ai');  
const human = new Agent('human');
```

Prior to initializing the communication between these two agents, we will make the assistant agent pre-process our criteria assessment data collected through field interviews using the AHP algorithm.

Each assessment has the form illustrated in equation 1. Where M stands for criteria matrix and c represents each criterion. Therefore, c_1 is time, c_2 is price and c_3 is trust. And, as previously outlined, each merchant was asked to compare all criteria against each other.

$$M = \begin{bmatrix} c_1/c_1 & c_1/c_2 & c_1/c_3 \\ c_2/c_1 & c_2/c_2 & c_2/c_3 \\ c_3/c_1 & c_3/c_2 & c_3/c_3 \end{bmatrix}. \quad (1)$$

Moving onward, for each assessment the AHP algorithm returns us an eigenvector (ev) with the weighting of each criterion for a particular rural merchant that we interviewed.

The eigenvector formula is indicated in equation 2. Where w_1 , w_2 and w_3 represents the weighting of each criterion. Saaty recommends that we check the consistency of the judgment by computing the consistency ratio [29]. To maintain the quality of the data we will only include in our analysis the judgements that were consistent. The lamda max operator is used to that end.

$$ev = [w_1 \quad w_2 \quad w_3]. \quad (2)$$

Once we process all judgements, we will proceed in synthesizing all judgements into one eigenvector. As Saaty also recommends, we will use the geometric mean to that end.

Equation 3 indicates the results. This means that for an average merchant in the Dukem region of Ethiopia, the time criterion is estimated to have a weighting of 37% importance, price a 29% weighting and trust a 25% weighting.

$$ev = [0.37 \quad 0.29 \quad 0.25]. \quad (3)$$

That being processed, we are now ready to program our communication engine.


We will program our assistant agent ('ai') to subscribe to our human agent 'request' events.

Every time the agent receives a request from its human partner it will take as arguments the alternative matrix of available drivers and the eigenvector resulting from the processing the dataset of judgements, to compute the TOPSIS algorithm and return a prescription:

```
ai.on('request', (msg) => {
  const res = ai.decide('topsis', data);
  ai.store(msg, 'request', human.id, ai.id);
  msg = `AGENT: The best fare for you is this one. The
rating is ${res[2]} stars. You will reach location in
around ${res[1]} minutes and the cost is ${res[0]}
birrs.`;
ai.tell({ name: 'response', msg }, human);
});
msg = 'HUMAN: I need to find a ride to market!';
human.tell({ name: 'request', msg }, ai);
```

5 Results

Once we run our programme and we inspect our agent instance in our console we will see this Javascript class:



```
Agent {
  _events: [Object: null prototype] { request: [Function] },
  _eventsCount: 1,
  _maxListeners: undefined,
  id: 'dc29ebb3-d602-4976-96c8-7aa1116cc16b',
  name: 'ai',
  isAlive: true,
  interactions:
    Set {
      Interaction {
        from: 'b52347b0-2de1-4ab8-bfe4-c97bb188cd96',
        to: 'dc29ebb3-d602-4976-96c8-7aa1116cc16b',
        evtnt: 'request',
        msg: 'HUMAN: I need to find a ride to market!',
        time: '2019-02-09T10:21:05+00:00' },
      Interaction {
        from: 'dc29ebb3-d602-4976-96c8-7aa1116cc16b',
        to: 'b52347b0-2de1-4ab8-bfe4-c97bb188cd96',
        evtnt: 'response',
        msg:
          'AGENT: The best fare for you is this one. The rating is 5 stars. You will reach location in around 5 minutes
and the cost is 2 birrs.',
        time: '2019-02-09T10:21:05+00:00' } },
  birthday: '2019-02-09T10:21:05+00:00' }
```

Fig. 3. Console results after running our simulation.

Where the full ‘msg’ text is: 'AGENT: The best fare for you is this one. The rating is 5 stars. You will reach location in around 5 minutes and the cost is 2 birrs.' Therefore, that means, our most desirable solution was Driver 1.

As can be seen, almost instantly we computed our response from the agent and all interactions remained self-contained within the agent entity. Driver 1 evidently provides the optimal solution as it has a high trust score (benefit criteria to maximize),

extremely low price (cost criterion to minimize) and it is extremely quick (time was a cost criterion to minimize).

The self-contained storage architecture of our machine-human communication engine allows developers to collect valuable data in real time to further customize interactions between agents.

To conclude, the major contributions of this paper can be summarized as follows:

- A literature review of MCDC and agent systems in Javascript.
- An open-source tool that sets the basis for interpretable intelligent agents from both the user and developer side that can be used for research simulations or industrial applications.
- A novel integration of high-level abstractions from AOSE and MCDC methods in Javascript that enables developers to program incremental interactions between computer agents and both computer and human agents.
- Proof of concept with a concrete use case to improve a social outcome.

6 Discussion and Further Research

This research was designed with a view to providing easy-to-deploy standardized methods using agents system architectures for general purpose MCDC problems in a scalable and ergonomic manner. We believe that this integration may have a wide range of potential use cases in management science and business analytics.

A matter for further study is the implementation of more human-like abstractions for machine argumentation and communication between human and intelligent agents, especially assessing this architecture in simulations with a much larger number of agents and events. Further improvements will include integrations with machine learning algorithms, reward and utility functions, three-valued logic inference systems and other general-purpose AI capabilities. All those features are still in progress. So the results will be published soon.

References

1. Gollin, D., & Rogerson, R. Agriculture, roads, and economic development in Uganda (No. w15863). National Bureau of Economic Research. (2010)
2. Wondemu, K. A., & Weiss, J. Rural roads and development: evidence from Ethiopia. *European journal of transport and infrastructure research*, 12(4), 417--439. (2012)
3. Roberts, P., Kc, S., & Rastogi, C. Rural access index: a key development indicator. (2006)
4. Belton, V. A comparison of the analytic hierarchy process and a simple multi-attribute value function. *European Journal of Operational Research*, 26(1), 7--21. (1986)
5. Pawlak, Z., & Sowinski, R. Rough set approach to multi-attribute decision analysis. *European Journal of Operational Research*, 72(3), 443--459. (1994)
6. Saaty, T. L., & Erdener, E. R. E. N. A new approach to performance measurement the analytic hierarchy process. *Design Methods and Theories*, 13(2), 62--68. (1979)
7. Hwang, C. L., & Yoon, K. Methods for multiple attribute decision making. In *Multiple attribute decision making* (pp. 58--191). Springer, Berlin, Heidelberg. (1981)

8. Roy, Bernard. "Classement et choix en présence de points de vue multiples (la méthode ELECTRE)". *La Revue d'Informatique et de Recherche Opérationnelle (RIRO)* (8): 57--75. (1968)
9. J.P. Brans & P. Vincke. "A preference ranking organisation method: The PROMETHEE method for MCDM". *Management Science*. (1985)
10. Ethiopia Population, total. World Bank Group. <https://data.worldbank.org/country/ethiopia> (2017)
11. Annual Report 2016/17. Agricultural Transformation Agency. (2017)
12. NBE Annual Report 2016/2017. National Bank of Ethiopia. (2017)
13. Shoham, Y. Agent-Oriented Programming. *Artificial Intelligence*. pp. 51--92 (1993)
14. Shoham, Y. Agent oriented programming: An overview of the framework and summary of recent research. In *International Conference on Logic at Work* (123--129). Springer, Berlin, Heidelberg. (1992)
15. Lukic, A., Luburi, N., Vidakovi, M., & Holbl, M. Development of multi-agent framework in JavaScript. (2017)
16. Calenda, T., De Benedetti, M., Messina, F., Pappalardo, G., & Santoro, C. AgentSimJS: A Web-based Multi-Agent Simulator with 3D Capabilities. In *WOA* (117--123). (2016)
17. De Jong, J., Stellingwerff, L., & Pazienza, G. E. Eve: a novel open-source web-based agent platform. In *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on* (1537--1541). IEEE. (2013)
18. Zionts, S. A survey of multiple criteria integer programming methods. In *Annals of Discrete Mathematics* (Vol. 5, 389--398). Elsevier. (1979)
19. Zavadskas, E. K., Turskis, Z., & Kildienė, S. State of art surveys of overviews on MCDM/MADM methods. *Technological and economic development of economy*, 20(1), 165--179. (2014)
20. Portugal, I., Alencar, P., & Cowan, D. The use of machine learning algorithms in recommender systems: a systematic review. *Expert Systems with Applications*. (2017)
21. Saaty, T. L. Exploring the interface between hierarchies, multiple objectives and fuzzy sets. *Fuzzy Sets and Systems*, 1(1), 57--68. doi:10.1016/0165-0114(78)90032-5 (1978)
22. Aczél, J., & Saaty, T. L. Procedures for synthesizing ratio judgements. *Journal of Mathematical Psychology*, 27(1), 97. doi:10.1016/0022-2496(83)90028-7 (1983)