

Application of Utility Maximization to Artificial Intelligence

Matthew Coe, Soumya Mitra, and Kyle Lierer

I. ABSTRACT

Within artificial intelligence (AI), robotics, and game design there are often situations where an agent, an autonomous entity that makes decisions, modeled by either a finite state machine [1] or hierarchical state machine like a behavior tree [2] needs to make a decision from a finite set of certain decisions (decisions that lead to a specific outcome). For instance, say an autonomous car for a rideshare company is picking up and dropping off clientele at a fixed price. Their goal is to maximize their profit. However, they are just picking up and dropping off clients in order. This method does not take into account the utility [3] of each of their actions. Such as, the driver could be dropping off nearby clients first and then picking up all nearby clients to cut down on drive time. In this case, the action with the highest utility would be dropping off nearby clients followed by picking up nearby clients. However, the agent must also consider the amount of time each client is waiting to be picked up, since if a client waits too long, they might not use the service again. We would like to suggest an algorithmic approach to the above problem (not choosing an action without considering the utility of the action) that is based upon the economics concept of utility maximization. This algorithmic approach would determine which action an agent should take from a set of actions assigned with a given utility. In other words, an agent should choose an interaction that would produce the highest utility for their circumstances.

II. PROBLEM DEFINITION

Given a list of actions, it is not a trivial task for a state-driven agent to decide on which action to perform. As a result of this, our goal is to design a generalized algorithmic approach to this problem that is based on the concept of utility maximization.

In this section we will formally define key terminologies used and the problem definition for our algorithmic approach. Before we can formally define the problem, it is important that we first define several key terms.

Table 1: Key Terminology

Name	Description
Utility (u)	The utility gained such that $u \in \mathbb{R}$. Each u is identified by a unique string.
Utilities (U)	A set of u such that $U \subset \mathbb{R}^n$ where $n \in \mathbb{Z}$.
Actions (A)	The set of all actions that can be taken.
Action (a)	A single action that can be taken such that $a \in A$.
State (s)	Represents a and the U gained from completing that specific action.

Agent (s_a)	A s instance which represents the beginning state of the agent making the decision.
Decision Tree (T)	A tree of s from which utility can be maximized. Each node in the tree can have 0 to ∞ children. This tree indicates all of the future decisions the agent can make in a finite amount of time.
Path (P)	An ordered set of s for paths such that $P \subset T$. A path consists of one chosen state from each level of T .
Utility Function $w(P)$	$w(P) = \sum_{s \in P} \sum_{U \in s} \sum_{u \in U} u$

With the key terminology defined, we can confidently outline our problem. Given an **Agent** (s_a) and a **Decision Tree** (T), what **Decision Path** (P) can be made such that the utility of the agent is maximized based on the **Utility Function** $f(P)$? Our goal is to create an algorithm that will find the path from the root of a tree to a leaf that maximizes the Utility of the Agent at the end of the decision tree.

In order to help visualize this problem, we have provided a visualization of example input in Table 2. In this example, the goal is to find the path that maximizes the utility of the agent, where the utility function is the summation of all U for a given P . In this example each node in the tree can have multiple parents

Table 2: Tree visualization

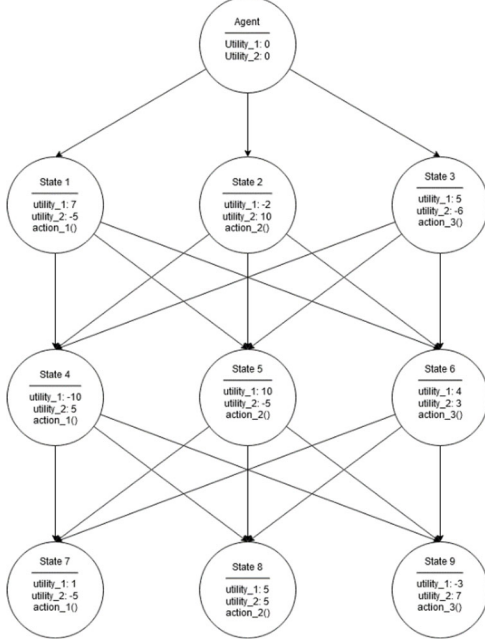
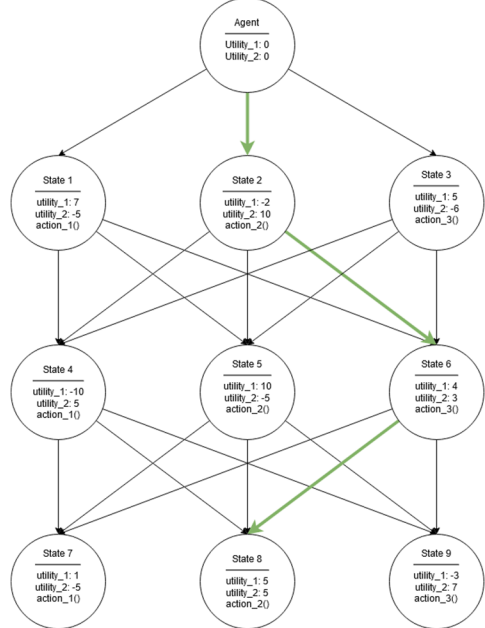


Table 3: Tree visualization with completed path



Based on table 3 the path in which the summation of utility is greatest is {"State 2", "State 3", "State 4"} where $f(\{"State 2", "State 3", "State 4"\})$ returns 20.

III. LITERATURE REVIEW

Coming up with a qualitative algorithm for decision making is not a new concept. This is one of the areas studied by decision theory, an interdisciplinary topic related with game theory [4] that is the study of an agent's choice(s) [5]. In order for us to establish a generalized algorithmic approach to qualitative decision making for a state driven AI, it is important that we take into account theories in both classical and contemporary decision theory.

Classical decision theory was proposed by mathematicians John von Neumann and Oskar Morgenstern in their 1944 book *Theory of Games and Economic Behavior* in which they defined the Von Neumann-Morgenstern (VNM) utility theorem [6]. In the VNM utility theorem, von Neumann and Morgenstern showed that when an agent is faced with a set of outcomes of different choices, they will behave in such a way such as to maximize the expected value of some function defined over potential outcomes at some point in the future [7]. Moreover, if the conditions for the VNM utility theorem are met, the agent is said to be rational and thus abide by the VNM utility theorem. The VNM utility theorem was the first basis to the expected utility hypothesis. Considered to first be introduced by Daniel Bernoulli in 1738 as a resolution to the St. Petersburg paradox [8], expected utility hypothesis is concerned with choosing an action when the decision maker is uncertain of the consequence from a chosen action. The VNM utility theorem was pivotal in establishing the classical decision theory. Leonard Savage shortly after proposed an alternate framework for looking at utility. Savage argued that expected utility is subjective to the agent [6]. As a result of this, an agent acts in their own interest whereas the VNM utility theorem does not treat an agent as being capable of making subjective decisions.

For our algorithm, we will pull logic and approaches from a combination of both the VNM utility theorem and Savage's subjective expected utility theorem to calculate the utility at each state within the decision tree. However, we will not be taking into account the probability that an action is successful: like both the VNM utility theorem [7] and Savage's subjective expected utility theorem [9]. We will also pull concepts from John von Neumann's minimax theorem. The most popular implementation of the minimax theorem is the minimax search algorithm which is a backtracking algorithm used to find a path in a tree, typically in two player board games [10]. Finally, we will also pull inspiration from both Dijkstra's shortest path algorithm and Bellman-Fords shortest path algorithm so that we have a perspective on popular optimization algorithms [11]. By building off of the VNM utility theorem, Savage's subjective expected utility theorem, von Neumann's minimax theorem, and popular optimization algorithms we have a solid theoretical foundation in classical decision theory from which to base our algorithm.

Contemporary decision theory is similar to classical decision. However, the introduction of more

advanced physical hardware has made decision theory more complicated in recent years. One of the primary differences between classical decision theory and contemporary decision theory is the introduction of machine learning and neural networks.

Machine learning takes a different route than classical decision theory when it comes to creating a model that represents an agent's decisions. In a 2014 article titled "On combining machine learning with decision making" published in Machine Learning, the authors created a machine learning framework that took into account utility through operational costs [12]. This allowed them to create a predictive model that better represented human decision-making when it came to modeling decisions making. However, in this instance decision-making was only applied to a single decision instead of a sequential set of decisions. Moreover, the goal was to both reflect human decision-making behavior as well as maximize utility. Moreover, their approach had to be retrained to deal with new decisions. Ultimately, since our approach for our project is for certain decisions, we do need not take into account probabilistic models and thus an approach through machine learning is excessive.

Neural networks, like machine learning, also take a different approach than classical decision theory. Like machine learning, neural networks are being used to determine the probabilistic outcome of a decision [13]. Through Bayesian statistics, neural networks are capable of determining sequential decisions that maximize utility [13]. Again, since our approach deals with certain decisions instead of uncertain decisions, the need to take into account the probability is unnecessary. Machine learning and neural networks have the power to model what decision should be taken based upon utility and human behavior incredibly

accurately [12], [13], however they are specifically geared towards modeling decisions that take into account probability of uncertain decisions. As a result, our approach will not be dealing with contemporary decision theory, as it is excessive to utilize machine learning or a neural network where it is not necessary.

It's important to look at Classical and Contemporary Decision Theory in order to determine a generalized approach to determining the best way to implement our specific algorithm. In the case of the Classical decision theories, we have the VNM utility theorem and Savage's expected utility theorem. We will be referencing the VNM utility theorem because it provides insight on how an agent can maximize a specific value, given a set of outcomes. Additionally, we will reference Savage's subjective expected utility theorem since it deals with the addition of the agent's own interests with the expected outcome based on maximizing utility.

It's important to look at both classical and contemporary decision theory in order to establish a generalized approach to determining the best way to implement our specific algorithm. In the case of the classical decision theories, we have the VNM utility theorem and Savage's expected utility theorem. We will be referencing the VNM utility theorem because it provides insight on how an agent can maximize a specific value, given a set of outcomes. Additionally, we will reference Savage's subjective expected utility theorem since it deals with the addition of the agent's own interests with the expected outcome based on maximizing utility. As for contemporary decision theory, we have decided that we will not be referencing any as the complexity is beyond the scope of our specific algorithm and what we intend to accomplish.

IV. OUR ALGORITHM

1. Our Proposed Algorithm

The problem we are trying to solve is an optimization problem. Because of the complexity of decision making, utilizing a brute force or similar approach would be an incredibly inefficient method to our specified problem. Given that the number of potential paths that could result from an agent making decisions, checking every single combination would result in an incredible amount of overhead for even a simple decision tree. As a result of this, we have opted to utilize a greedy approach to determine the path that maximizes an agent's utility. In order to do this, we are looking at two very famous greedy algorithms, Dijkstra's shortest path algorithm and the Bellman-Ford shortest path algorithm, as a starting point. Most importantly, we are taking inspiration from the technique of *relaxation* that both Dijkstra's and Bellman-Ford algorithms utilize [11].

2. Algorithm Pseudo-Code

Before, we formally define our proposed algorithm we will define the relaxation principle. In our algorithm we will maintain an attribute $s.w$ which is lower bound on the weight of the maximum utility path from s_a to a leaf s in T and we will call the *maximum utility estimate*. We will initialize the maximum utility estimates using the following procedure:

INITIALIZE-SINGLE-SOURCE (T, s_a)

- 1 **for** each $s \in T.S$
- 2 $s.w = -\infty$

```

3       $s.\pi = NIL$ 
4       $s_a.w = 0$ 

```

Once initialization is complete, for all $s \in S$, $s_a.w = 0$, $s.\pi = NIL$, and for all $s \in S - \{s_a\}$, $s.w = -\infty$. With initialization defined, we can define the procedure for relaxation where $u \in T.S$. In the relaxation procedure we will update the maximum utility estimates for a State m . Furthermore, we are using our previously defined utility function $w(P)$ in our relaxation.

RELAX (m, s)

```

1  if  $s.w < m.w + w(m, s)$ 
2       $s.w = m.w + w(m, s)$ 
3       $s.\pi = m$ 

```

With relaxation and initialization defined, we can move onto the core algorithm. Similar to Dijkstra's algorithm we will maintain a set of nodes and repeatedly select nodes to relax. However, unlike Dijkstra's algorithm, our algorithm can handle negative estimates. This is a result of our algorithm using a tree instead of graph making repeating cycles impossible. Our algorithm will select nodes from a maximum priority queue Q , relax the node's maximum utility estimate using the utility function, and then add those to set F . Once Q is empty, we find the maximum path in F and then the algorithm is done.

PROPOSED-ALGORITHM (T, s_a)

```

1  INITIALIZE-SINGLE-SOURCE ( $T, s_a$ )
2   $F \leftarrow \emptyset$ 
3   $Q \leftarrow T.S$ 
4  while  $Q \neq \emptyset$ 
5      do  $m \leftarrow \text{EXTRACT-MAX}(Q)$ 
6       $F \leftarrow F \cup \{m\}$ 
7      for each  $s \in \text{Children}[m]$ 
8          RELAX( $m, s$ )
9  return FIND-MAX-PATH ( $F$ )

```

Finally, we need to define the procedure FIND-MAX-PATH (F). This procedure will find the leaf State with the highest maximum utility estimate, such that $m \in F$. Then we will back track up the tree using the temporary state p such that $p \in F$. As a we backtrack up the tree, we will put States into the path variable P such that $P \in T$. Finally, we return P .

FIND-MAX-PATH (F)

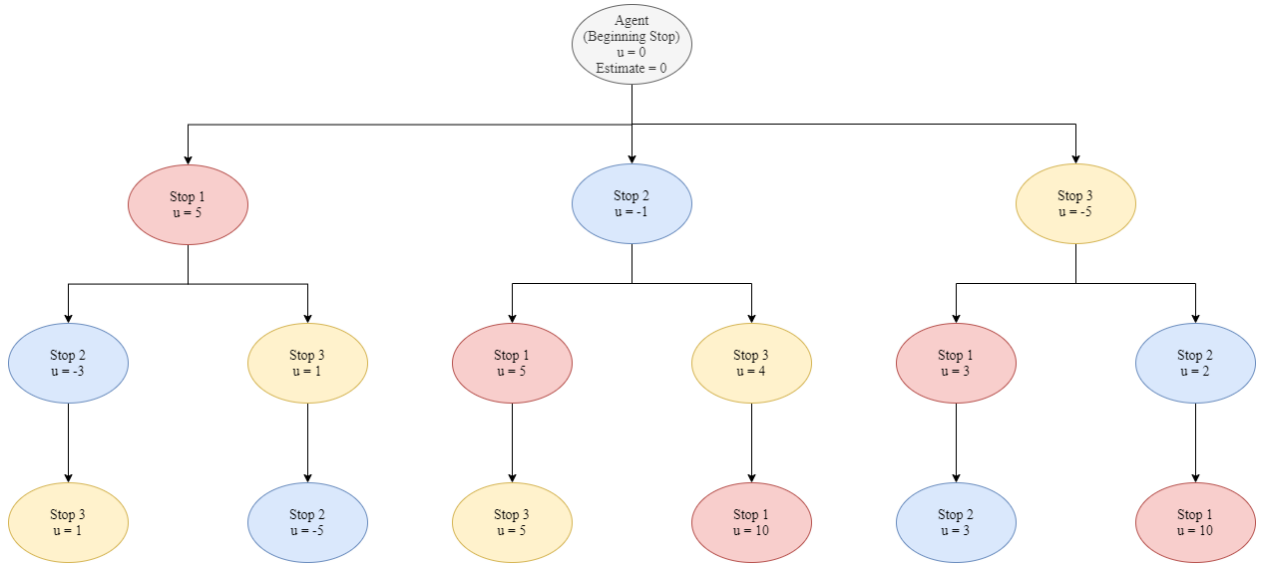
```

1   $m \leftarrow F[0]$ 
2  for each  $s \in F$  with no children
3      if  $m.w > s.w$ ;  $m = s$ 
4   $P \leftarrow NIL$ 
5   $p = m$ 
6  while  $p.\pi \neq NIL$ 
7       $P \leftarrow p$ 
8       $p = p.\pi$ 
9  return  $P$ 

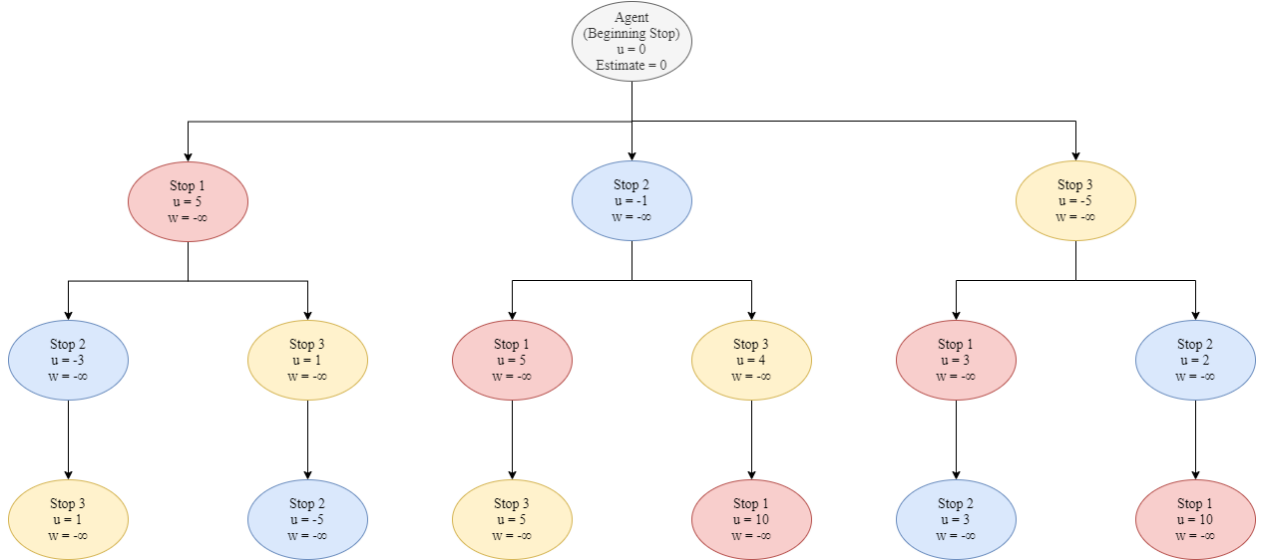
```

3. Use Case

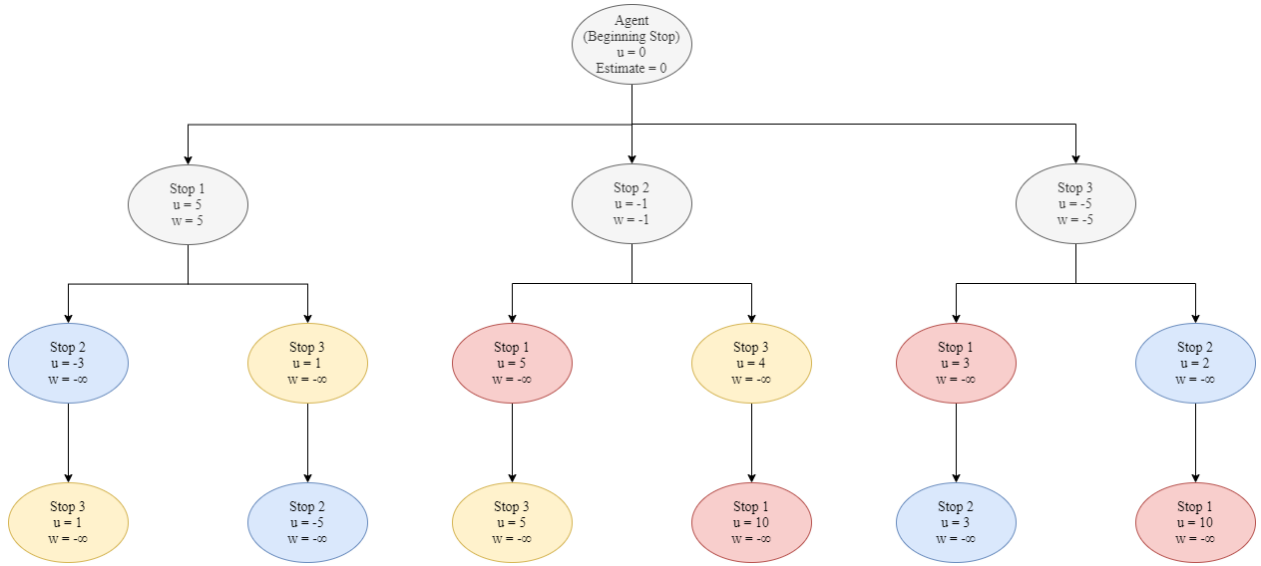
For this algorithm, we will use an example of ride sharing to demonstrate how the driver can choose the order of each of their stops to yield the highest utility. For this example, we will use this tree to depict all the possible actions and the respective utility gained from taking those actions. The utility in this example is found based around the time it takes to reach each stop. Thus, less time results in higher utility. Moreover, in this example the set of utilities is only made up of one singular utility.



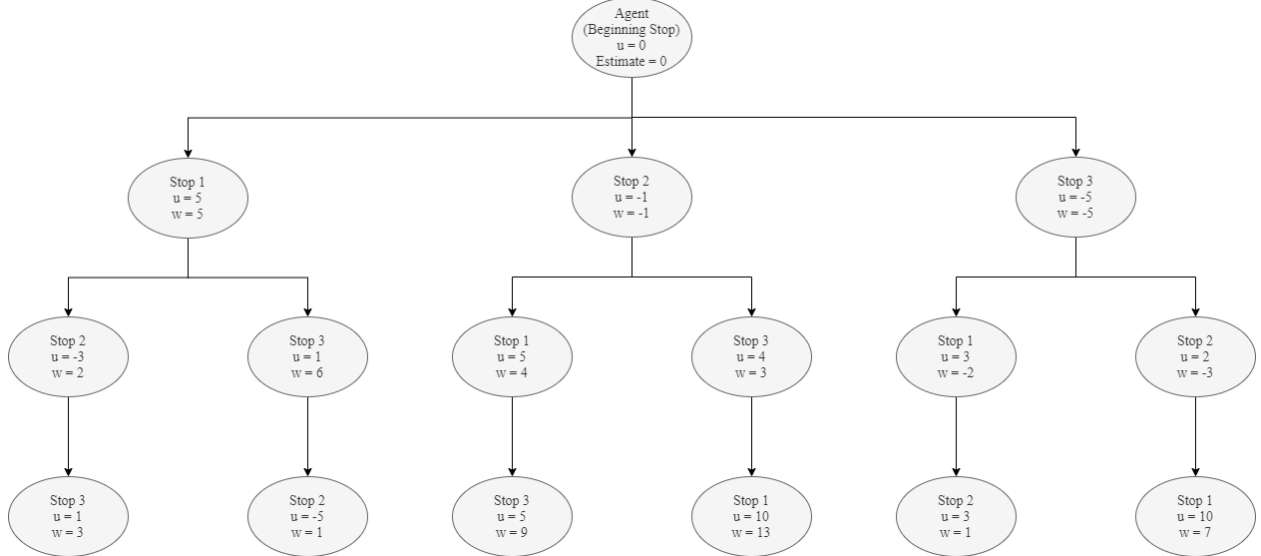
We first set the weight of all the state's, excluding the agent (s_a), as $-\infty$ with the INITIALIZE-SINGLE-SOURCE (T, s_a) function.



Then, we “relax” the first three instances of stop 1, stop 2, and stop 3 by setting their weights equal to the sum of the stop's utility and the utility of all that stop's parent states. This is done through the utilization of RELAX(m, s). Thus, we have:



This step will repeat until all possible states have been visited. Hence, we get:



We then will utilize FIND-MAX-PATH (F) to determine the path with the highest utility. With this, we will find all the last stops and compare their weights to determine which stop has the maximum weight (this stop being p). Then, we will find all the parent stops of p and store them into the maximum path P . Then, we shall receive the path of stops that garner the maximum utility.

4. Cost Analysis

There are several cases where our algorithm may operate in the best case, but we have concentrated on the worst-case analysis, primarily. The time-cost complexity analysis for our algorithm closely mirrors that of Dijkstra's shortest path algorithm. The final path consists of a single State from each level and back-traces from the bottom of the tree to the top. To complete our cost analysis, we have considered the cost for each of the major methods which we have for algorithm. The schema is such that the algorithm steps are on the left side and the complexity is on the right side for individual relevant steps.

Cost for the Initialization Step:

INITIALIZE-SINGLE-SOURCE (T, s_a)

1 **for each** $s \in T.S$

$\theta(|S|)$

2	$s.w = -\infty$	$\theta(S)$
3	$s.\pi = NIL$	$\theta(S)$
4	$s_a.w = 0$	$\theta(1)$

Explanation: Line 1 i.e. grabbing each state from the set of states – has complexity $\theta(|S|)$, where $|S|$ is the number of states or nodes in our graph. We initialize all the state's weights to minimum i.e. $-\infty$, and initializes the parent of each node to NIL – both having time complexity $\theta(|S|)$. Finally, the weight of the root node i.e. Agent is initialized to 0 which has a complexity of $\theta(1)$.

Cost for Relaxation Step:

RELAX (m, s)

1	if $s.w < m.w + w(m, s)$	0 to $\theta(1)$
2	$s.w = m.w + w(m, s)$	0 to $\theta(1)$
3	$s.\pi = m$	0 to $\theta(1)$

Explanation: The relax step checks whether the accumulated utility of an extracted vertex(node) plus the additional utility attainable by going to state s will be greater than the accumulated utility up until s . This is an $\theta(1)$ operation and it will do so for all the edges.

Cost for Proposed Algorithm:

PROPOSED-ALGORITHM (T, s_a)

1	INITIALIZE-SINGLE-SOURCE (T, s_a)	$\theta(S)$
2	$F \leftarrow \emptyset$	$\theta(1)$
3	$Q \leftarrow T.S$	$\theta(S)$
4	while $Q \neq \emptyset$	$\theta(S)$
5	do $m \leftarrow \text{EXTRACT-MAX}(Q)$	$\theta(S) \cdot \text{time complexity of EXTRACT-MAX}$
6	$F \leftarrow F \cup \{m\}$	$\theta(S)$
7	for each $s \in \text{Children}[m]$	$\theta(S)$
8	RELAX(m, s)	$\theta(\text{Children in } T \cdot \text{time complexity of INCREASE-KEY})$
9	return FIND-MAX-PATH (F)	$\theta(h)$

Explanation: The first step is the initialization step which we have described earlier. F is the queue which will hold all the vertices or states as we extract each one from Q . The priority Q is used to store as a max-heap all the states. Initially it will take $\theta(|S|)$ time, since all the states are initially $-\infty$ (there needn't be any max-heapify operation involved at this stage). Line 4 is the while loop that runs $|S|$ times i.e. as long as Q is not empty. Line 5 is the extract maximum utility state until now, which runs $|S|$ times. The complexity for extracting and adjusting the priority queue to preserve its property of priority is $\theta(1)$ for array, $\theta(\log|S|)$ for max-heap and $\theta(\log|S|)$ for a Fibonacci heap implementations of the priority queue Q .

Line 6 just unionizes the extracted vertex(state) m into the set F which runs $\theta(|S|)$. In line 7, it runs $\text{Children in } T$ times i.e. it will run the number of times equal to the number of outgoing children of m . The RELAX(m, s) method will take $\theta(|R|)$ to relax each edge. Finally, we return the FIND-MAX-PATH(F) which takes $\theta(h)$ time.

Cost for Find Max Path:

FIND-MAX-PATH (F)

1	$m \leftarrow F[0]$	$\theta(1)$
2	for each $s \in F$ with no children	
3	if $m.w > s.w; m = s$	
4	$P \leftarrow NIL$	$\theta(1)$
5	$p = m$	$\theta(1)$
6	while $p.\pi \neq NIL$	$\theta(h)$
7	$P \leftarrow p$	$\theta(h)$
8	$p = p.\pi$	$\theta(h)$
9	return P	$\theta(1)$

Explanation: To find maximum utility path, we first assign in line 1 the first state in F to m . Now, in line 2 and 3, we try to find the maximum utility path-weight amongst the leaf states. The time complexity for those two steps will depend on the number of leaf states we have. In line 4, we create the path P and assign it a null value which takes $\theta(1)$ time. Line 5 initially has the first highest utility vertex from the leaf nodes. Line 6 starts to iteratively back-trace the path by tracing back the parent of each current child until it reaches the root state. This should have a time-complexity of $\theta(h)$, where h is the height of the tree.

Therefore, the algorithm has a time complexity of our proposed algorithm is $\theta(|S| + 1 + |S| + |S| + |S| \cdot \text{time complexity of EXTRACT-MAX} + |S| + |S| \cdot |\text{Children}[m]| + |\text{Children in } T| * \text{time complexity of INCREASE-KEY} - \text{KEY} + h \cdot \text{time complexity of EXTRACT-MAX} + |\text{Children in } T| * \text{time complexity of INCREASE-KEY})$.

This can be simplified to $\theta(|S| \cdot \text{EXTRACT-MAX} + |\text{Children in } T| \cdot \text{INCREASE-KEY})$.

5. Proof of Correctness

Here we prove that our algorithm is correct. However, before we do that let us declare the state attribute $s.\delta$ to be the maximum utility path summation from s_a to m . It is also to be noted that $s.w$ continues to be the maximum utility path summation estimate.

Lemma: For each $s \in F$, $s.w = s.\delta$; this is similar to Lemma 24.17 in *Introduction to Algorithms* by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein [11].

Proof by Induction: *Base case*, $|F| = 1$: $|F| = 1$ only when $F = \{s_a\}$ and $s.w = s.\delta = 0$.

Inductive Hypothesis (IH): Let m be the last state added to F . Let $F' = F \cup \{m\}$. Thus, $m \notin F$ and for each $s \in F'$, $s.w = s.\delta$.

By the IH, for every state in F' that is not m , the correct utility summation value is present. In order to prove by induction, we need to show that $m.w = m.\delta$.

Suppose there exists a maximum utility path Q from s_a to m such that $w(Q) < m.w$.

Q starts in F' and at some leaves F' . Let xy be the first edge in Q such that it leaves F' . Let F_x be the s_a to x path of Q . Thus, $w(Q_x) + w(xy) \leq w(Q)$.

Since $x.w$ is the length of the maximum path from s_a to x by IH, $x.w \leq w(Q_x)$ which gives us $x.w + w(xy) \leq w(Q_x)$.

Since y is adjacent to x , $y.w$ must have been updated by our algorithm, so $y.w \leq x.w + w(xy)$.

Finally, since m was picked by our algorithm, m must have the greatest utility summation value such that $m.w \leq y.w$. Combining these inequalities in reverse order gives us $x.w < x.w$ which is a contradiction so there is no path Q with greater maximum utility thus $m.w = m.\delta$.

Thus, through induction, there exists a path within the tree that maximizes the utility.

CONTRIBUTION CLARIFICATIONS

1. Abstract

All members (Matthew Coe, Soumya Mitra, and Kyle Lierer) contributed equally.

2. Literature Review

All members equally contributed to the literature review including reading and writing all of the referenced material.

3. Problem Definition

All members equally contributed to the problem definition.

4. Our Algorithm

Soumya Mitra wrote 33% of section 4.1, 5% of 4.2, 5% of 4.3, and 90% of 4.4.

Matthew Coe wrote 33% of section 4.1, 5% of 4.2, 90% of 4.3, and 5% of 4.4.

Kyle Lierer wrote 33% of section 4.1, 90% of 4.2, 5% of 4.3, 5% of 4.4, and 100% of 4.5.

- [10] P. Norvig, "Search and the Game of Othello," in *Paradigms of Artificial Intelligence Programming*, Elsevier, 1992, pp. 596–654.
- [11] R. L. Cormen, Thomas H;Leiserson, Charles E;Rivest, "Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein Introduction to algorithms 2001." 2001.
- [12] T. Tulabandhula and C. Rudin, "On combining machine learning with decision making," *Mach. Learn.*, vol. 97, no. 1–2, pp. 33–64, Oct. 2014.
- [13] Z. Ghahramani, "Probabilistic machine learning and artificial intelligence," *Nature*, vol. 521, no. 7553. Nature Publishing Group, pp. 452–459, 27-May-2015.

REFERENCES

- [1] "Finite state machine | Definition of Finite state machine at Dictionary.com." [Online]. Available: <https://www.dictionary.com/browse/finite-state-machine>. [Accessed: 28-Feb-2020].
- [2] M. Colledanchise and P. Ögren, "Behavior Trees in Robotics and AI: An Introduction," *CoRR*, Aug. 2017.
- [3] R. M. Adelson and P. C. Fishburn, "Utility Theory for Decision Making," *Oper. Res. Q.*, vol. 22, no. 3, p. 308, Sep. 1971.
- [4] E. Kalai, "Game theory: Analysis of conflict," *Games Econ. Behav.*, vol. 3, no. 3, pp. 387–391, Aug. 1991.
- [5] C. Allen, U. Nodelman, and E. N. Zalta, "The Stanford Encyclopedia of Philosophy : A Developed Dynamic Reference Work," *Metaphilosophy*, vol. 33, no. 1-2, pp. 210–228, 2002.
- [6] R. Graboś, "Qualitative model of decision making," in *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, 2004, vol. 3192, pp. 480–489.
- [7] J. von Neumann and O. Morgenstern, *Theory of games and economic behavior*. Princeton, NJ, US: Princeton University Press, 1944.
- [8] D. Schmeidler and P. Wakker, "Expected Utility and Mathematical Expectation," in *Utility and Probability*, London: Palgrave Macmillan UK, 1990, pp. 70–78.
- [9] I. Gilboa, "Expected utility with purely subjective non-additive probabilities," *J. Math. Econ.*, vol. 16, no. 1, pp. 65–88, Jan. 1987.