When to Stop? That is the Question

Shulamit Reches[1] and **Meir Kalech**[2] and **Rami Stern**[2]
[1]Department of Applied Mathematics, Jerusalem College of Technology, Israel, shulamit.reches@gmail.com
[2]Information Systems Engineering Department, Ben-Gurion University, Israel, kalech@bgu.ac.il, sternr@gmail.com

Abstract

When to make a decision is a key question in decision making problems characterized by uncertainty. In this paper we deal with decision making in environments where the information arrives dynamically. We address the tradeoff between waiting and stopping strategies. On the one hand, waiting to obtain more information reduces the uncertainty, but it comes with a cost. On the other hand, stopping and making a decision based on an expected utility, decreases the cost of waiting, but the decision is made based on uncertain information. In this paper, we prove that computing the optimal time to make a decision that guarantees the optimal utility is NP-hard. We propose a pessimistic approximation that guarantees an optimal decision when the recommendation is to wait. We empirically evaluate our algorithm and show that the quality of the decision is near-optimal and much faster than the optimal algorithm.

Introduction

There are many real world domains in which an agent has to choose the optimal alternative among a set of candidates based on their utility. The problem becomes more complicated when the utility depends on events that occur dynamically and therefore the decision itself is based on dynamically changing, uncertain information. In such domains, the question is whether to stop at a particular point and make the best decision based on the current information or, to wait for additional information to arrive in order to make a better decision.

As an example, consider the problem of buying the best stock among several stocks in the stock market. The values of the stocks may change over time due to future events, such as publication of the company's sales report or a change in the interest rate, etc. The longer we wait, the more information becomes available and as a result, a decision is made with more certainty. However, in many real world domains there is a cost to waiting. Thus there is a tradeoff between a waiting strategy that enables one to achieve more information and decreases the uncertainty, and a stopping strategy which decreases the cost.

The goal of this paper is to determine the best time to make a decision in order to maximize the expected gain while taking into consideration the waiting cost. Previous work (Kalech and Pfeffer 2010) presents optimal and heuristic algorithms for implementing this task. But, as theoretical and empirical evaluation demonstrates, the algorithm proposed in this paper provides better results. The significance of this paper is that it (1) formally defines and proves that the problem of finding the best time to stop is a NP-hard problem; (2) proposes a polynomial approximation algorithm for solving the problem and providing a bound on its error; (3) proves that our algorithm is pessimistic, which means that its waiting decision is optimal; and (4) empirically evaluates the algorithm and shows its advantages over previous algorithms, proposed by (Kalech and Pfeffer 2010).

Our empirical evaluation on a syntactic simulation of the stock market shows that there is no significant difference between the quality of the decision made by our algorithm and an optimal algorithm. Moreover, our algorithm's runtime is polynomial rather than an exponential runtime of the optimal algorithm. Finally, our algorithm is better, in terms of runtime and quality of the decision, than the heuristic proposed by (Kalech and Pfeffer 2010).

Related Work

Our work is related in some respects to Horvitz and Rutledge's work (Horvitz and Rutledge 1991) on decision making under bounded resources. The execution of a task is associated with a utility and a cost depending on resources. When the resources are bounded, the question is: What is the best stopping point that will maximally satisfy the task? The major difference between Horvitz's approach and ours is that his goal is to execute a single task while we are seeking to select a candidate from among multiple candidates.

Another class of problems related to our work is decision making in regard to multiple observations that are informative but expensive. The challenge is to decide which variables to observe in order to maximize the expected utility. Krause and Guestrin present this problem in the domain of sensor placement. They consider a sensor network in which the utility of a sensor is determined by the certainty about the measured quantity. The task is to efficiently select the most informative subsets of observation (Krause, Singh, and Guestrin 2008). (Bilgic and Getoor 2007) address a similar problem of efficient feature acquisition for classification in domains in which there are costs associated with the ac-

quisition. The objective is to minimize the sum of the information acquisition cost. Similarly, (Radovilsky and Shimony 2008) deals with selection under uncertainty and develops an algorithm based on the value of information (VOI) with a semi-myopic approximation scheme to problems with real-valued utilities. However, in our case we do not choose a **subset** of observations/actions set, but a single candidate that maximizes the expected utility. This difference is significant since we do not focus on the main challenge of the above papers of how to reduce the complexity of the subset selection.

Finally, the tradeoff between uncertainty and cost, relates to the optimal stopping problem (OSP) (Peskir and Shiryaev 2006). In problems, the challenge is to determine when to stop the process so as to maximize the utility. However, there is a basic difference between the two. The decision we obtain is based on multiple alternatives, while in OSP, when to stop is made for only one alternative. Multiple alternatives increase the complexity exponentially.

Model Description

To describe the model clearly we use an example from the stock market. Assume a decision maker wishes to decide about what stock to purchase from a group of three stocks (c_1, c_2, c_3). The value of the stocks is influenced by future events such as the consumer price index (CPI), interest rates, etc. The decision maker cannot evaluate the influence of the future events on the stocks with certainty, but only with some degree of probability. Obviously, the sooner the decision is taken, the less is lost by not investing the money. On the other hand, the longer the waiting time, the more information that can be gathered by knowing the outcome of the expected events and consequently a more certain decision can be made.

In our model, each decision will be designated by a *candidate*, and throughout the paper we will refer to the candidate set $C = \{c_1, c_2, ..., c_m\}$. A candidate's utility depends on dynamically arriving information. We represent the dynamic information by random variables. The most fundamental entity is a *variable*.

Definition 1 (variable) *A* variable *is a discrete, finite random variable X taking values $x_1, ..., x_k$. The function $\Gamma(X_i) \in T$ represents the time stamp of variable X_i, where T is a time horizon $T = [a, b]$. Given integers i and j, $i < j \Rightarrow \Gamma(X_i) \leq \Gamma(X_j)$.*

An example of a variable is the interest decreasing in the stock domain, whose time stamp is 1 (assuming January is 0). Each variable is associated with a probability distribution over outcomes.

Definition 2 (assignment) *If X is a variable, an* assignment *to X is an outcome x_i of X. Two sets of assignments are* consistent *if they do not contain different outcomes of the same variable. A time t* global assignment*, denoted σ^t, is an assignment of values to all variables whose time stamp is less than or equal to t.*

Each candidate's utility depends on a set of variables. We represent the way the utility depends on these variables by a tree.

Definition 3 (candidate tree) *A candidate tree ct_i for candidate c_i is a tree in which the internal nodes are variables. The variable corresponding to the node n is denoted $X(n)$. The edges out of the node n are the possible assignments to $X(n)$. Each edge $X = x$ is labeled by its probability, denoted $p(X = x)$. A leaf n is labeled by its utility $\mathcal{U}(n)$. CT represents the set of candidate trees.*

The nodes along a path in the tree must be in increasing order of time. The variables appearing in ct_i are denoted by V_i. Candidate c_i's utility depends on those variables.

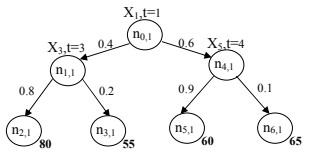

Figure 1: Candidate tree ct_1.

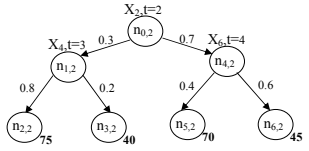

Figure 2: Candidate tree ct_2.

Figures 1 and 2 present two candidate trees. For instance, the node $n_{1,1}$ in ct_1 represents variable X_3 at time $t = 3$. The probability of its left edge is 0.8 and of its right edge is 0.2. The value 80 in the left leaf of ct_1 represents the utility of the path $\{n_{0,1}, n_{1,1}\}$ through the edged labelled by the probabilities 0.4 and 0.8. This root to leaf path represents one possibility of assignments sequence that determines the utility of the candidate. We assume that different candidates are affected by different variables.

The utility of a candidate is known for certain only at the leaves. However, the expected utility of a candidate can be calculated at any depth and will consider the subtree from that depth. The expected utility computation can be trivially implemented by a recursive function. The expected utility of a leaf is its utility; for an internal node it is the expectation of the expected utilities of its children. Formally:

Definition 4 (expected utility) *Given a node $n \in ct_i$, the function $\mathcal{EU}(n)$, returns the expected utility of n:*

$$\mathcal{EU}(n) = \begin{cases} \mathcal{U}(n) & \textit{n is a leaf} \\ \sum_i p(X(n) = x_i)\mathcal{EU}(n_i) & \textit{otherwise} \end{cases}$$

where n_i represents the successor node of n via assignment $X = x_i$.

For instance, the expected utility of the root in Figure 1 is: $0.4*(0.8*0.8+0.2*0.55)+0.6*(0.9*0.6+0.1*0.65) = 1.77$.

The expected utility is only an estimate of the real utility, based on the information known at the current time. Waiting to the next time reduces the uncertainty about the candidates's utilities and hence increases the chance to make a good decision. However, waiting incurs a cost.

Definition 5 (cost) *Each assignment a is associated with a waiting cost, denoted $\mathcal{CST}(a)$.*

Definition 6 (path) *Given a node $n \in ct_i$, the function $\mathcal{PTH}(n)$, returns the set of assignments in the root to n path: $\{X_{i_1} = x_{i_1}, X_{i_2} = x_{i_2}, ...\}$.*

Definition 7 (probability of path) *Given a node n, $Pr\mathcal{PTH}(n) = \prod_{j \in \mathcal{PTH}(n)} Pr(j)$.*

The expected gain is the difference between the expected utility and the cost:

Definition 8 (expected gain) *Given node $n \in ct_i$,*

$$\mathcal{GN}(n) = \mathcal{EU}(n) - \mathcal{CST}(\sigma^{\Gamma(X(n))})$$

There is a tradeoff between the first component of $\mathcal{GN}$, the expected utility, and the second component, the waiting cost. The challenge of this paper is to present an algorithm to find the time that maximizes the gain. Unfortunately, we could not compute the time to make the decision and the decision on the best candidate separately, since in each time the decision might be different. Therefore we define a policy to determine what to do in all situations the decision maker might face.

Definition 9 (policy) *A* policy *is a function $\pi : \mathcal{G} \rightarrow \{stop, wait\}$, where $\mathcal{G}$ is the set of all global assignments*

If the policy specifies to stop, the decision maker also needs to decide which candidate to choose. Since this decision is simple we do not include it in the definition of a policy.

Based on the above definitions, we can define the timed decision making problem (TDM):

Definition 10 (Timed Decision Making(TDM) problem) *Let $W = \{CT, \sigma, T, \mathcal{CST}\}$, where*

1. *CT is a set candidate trees $CT = \{ct_1, ct_2, ..., ct_m\}$.*
2. *$\mathcal{G}$ is a set of all global assignments.*
3. *$T = [a, b]$ is a time horizon.*
4. *$\mathcal{CST} : \mathcal{G} \rightarrow \mathbb{R}$, is a cost function.*

Let the global expected gain, $GEG(W, t, \pi)$ be a function that obtains W, a time stamp $t \in T$, and a policy π, and calculates the expected gain to the decision maker from this policy.

Given W and a time stamp $t \in T$, the TDM problem is finding a policy that maximizes $GEG(W, t, \pi)$.

We present the timed decision making problem as a decision problem in order to prove that timed decision making problem (TDM) is NP-hard, .

Given $W = \{CT, \mathcal{G}, T, \mathcal{CST}\}$, a time stamp $t \in T$, and a non-negative integer K. Answer: "Yes" if there exist a policy π such that the global expected gain $GEG(W, t, \pi) \geq K$.

Theorem 1 *TDM problem is NP-hard.*

Proof: We present a reduction from the SAT problem (Cook 1971). An instance of SAT is given by a propositional logic formula $\Phi(x_1, ..., x_n)$, the aim being to answer "yes" if there is some assignment to the Boolean variable $x_1, ..., x_n$ that satisfies the formula. Without loss of generality, we assume that $\Phi(x_1, ..., x_n)$ is presented in Conjunctive Normal Form (CNF), i.e., $\Phi(x_1, ..., x_n) = \bigwedge_i \psi_i$, where each ψ_i is a clause-a disjunction of literals. We create an instance of TDM as follows.

- We set $C = \{c_1, c_2, ..., c_n, c_{n+1}\}$.
- We set the time horizon $T = [0, t_n]$.
- For each Boolean variable x_i of the SAT formula, we create a timed variable X_i which affects the candidate c_i. We also create an additional timed variable X_{n+1} which affects the candidate c_{n+1}.
- Each candidate tree ct_i contains the variable X_i. The variable has two possible assignments: 0 or 1, with probability 0.5. $CT = \{ct_1, ct_2, ..., ct_{n+1}\}$.
- A global assignment σ is $(X_1 = a_1, ..., X_{n+1} = a_{n+1})$ where $a_i \in \{0, 1\} \forall 1 \leq i \leq n+1$.
- For each variable $X_i, 1 \leq i \leq n$ there is a time stamp $\Gamma(X_i) = t_i$
- We set $K = B + \frac{c}{2^n}$ where $B, c > 0$.
- The $\mathcal{CST}$ function calculates a real value to each assignment σ_i. Let $S = \sum_{\sigma_i} \mathcal{CST}(\sigma_i)$ where σ_i is an assignment of time stamp t_n.

The expected gain of an assignment $\sigma^{t_n} = (a_1, ..., a_n)$ is $U(a_1, ..., a_n) = max(\mathcal{GN}(n)) - cost(\sigma^{t_n})$ when n is a node in the path of the assignment σ^{t_n}. Suppose that the value of $U(a_1, ..., a_n)$ is:

$$U(a_1, ..., a_n) = \begin{cases} D & \text{if } \Phi(x_1, ..., x_n) = true \\ 0 & \text{else} \end{cases}$$

Where $a_i = 1$ iff $x_i = true$ and $D = c + B + 2^n S$.

Suppose that $\mathcal{EU}(n)\ \forall n \in tc_{n+1} = B > 0$ and that the expected gain of candidates c_{n+1} at time $t = 0$ is the highest among the candidates. Assume also that we already know the expected gain of waiting to time stamps $t_1, ..., t_{n-1}$ and it is less than K. Let $W = \{CT, \mathcal{G}, T, \mathcal{CST}\}$.

We now prove that there exists a policy π such that $GEG(W, t, \pi) \geq K$ if and only if $\Phi(x_1, ..., x_n)$ is satisfiable.

($\Rightarrow$) Assume there is a policy π such that $GEG(W, t, \pi) \geq K$. As a result, since $B < K$ and the expected gain of waiting to time stamps $t_1, ..., t_{n-1}$ are less than K, there must be at least one assignment σ^{t_n} of the random variables $X_1 = a_1, ..., X_n = a_n$ that holds $U(a_1, ..., a_n) > B$, and thus $U(a_1, ..., a_n) = D$. osing c_1 and c_2 Consequently, for $x_1, ..., x_n$ that correspond to the above assignment, $\Phi(x_1, ..., x_n) = true$, and $x_1, ..., x_n$ satisfies the formula.

($\Leftarrow$) Assume that $\Phi(x_1, ..., x_n)$ is satisfiable; then for the satisfying assignment $x_1, ..., x_n$, there is an assignment of the random variables $X_1 = a_1, ..., X_n = a_n$ that holds $U(a_1, ..., a_n) = D$. We can define a policy π as follows: $\pi(\sigma^t) = wait$ iff there exists an assignment of the random variables $X_1 = a_1, ..., X_n = a_n$ such that $U(a_1, ..., a_n) = D$. According to the above, there exists such an assignment. The probability of this assignment is $\frac{1}{2^n}$.

Consequently, based on the knowledge that the overall cost of waiting to time stamp t_n is S, the total expected gain from this policy holds: $TEG(W, t, \pi) \geq \frac{1}{2^n} D + (1 - \frac{1}{2^n})B - S = \frac{1}{2^n}(c + B + 2^n S) + B - \frac{1}{2^n} B - S = B + \frac{c}{2^n} = K.$ As a result we obtain that Timed Decision Making (TDM) is NP-hard.

It is possible to represent our problem by Markov Decision Process (MDP). In such a model, the states at time t would be the time t global assignments, and the actions would be to either select the best candidate at that time, or wait one more time step. The transition function from a time t state to a time $t + 1$ state for a wait action would be given by the product of the probabilities of the time $t + 1$ assignments. A stop action leads to a terminal state in which a reward is received equal to the gain of the winning candidate. For space limitations we refer the reader to Kalech and

Pfeffer (Kalech and Pfeffer 2010) which explain this model and its limitations in our problem.

As proved, this problem is NP-hard, thus any optimal algorithm will increase exponentially in the number of candidates (for an optimal algorithm see (Kalech and Pfeffer 2010)). In the next section, we present a polynomial approximation model, in which the expected utility from stopping is computed optimally but not that of the waiting.

An Approximate Solution

The approximated gain can be calculated by a decision tree approach. In this approach we merge candidate utility functions into a single decision tree, where each level in the tree represents a time stamp associated with a variable, i.e., level l_i in the tree represents a time point t_i where we decide whether to stop or wait.

In the decision tree, there are two nodes on each level:

Stop node, where the decision maker stops and chooses one of the candidates. A stop node, sp_i is the expected utility of stopping at level l_i.

Wait node, where the decision maker decides to wait. The wait node, wt_i is the expected utility of waiting for the next time level. This is the maximum between the stop node and the wait node of level l_{i+1}.

To describe the calculation of sp_i, we define first the following:

Definition 11 ($NODES_t^j$) *The set $NODES_t^j$ represents the nodes whose time is less or equal than t in ct_j and they have no children with time stamp less or equal to t.*

For example, in Figure 1, $NODES_3^1 = \{n_{2,1}, n_{3,1}, n_{4,1}\}$, $NODES_4^1 = \{n_{2,1}, n_{3,1}, n_{5,1}, n_{6,1}\}$.

In our approximated solution we compute, for each time stamp t_i, the expected utility by stopping (sp_i) at that level. When stopping, the optimal choice is the candidate with the highest expected utility. Thus, the expected utility of stopping (sp_i) is computed by finding the expected utility of the winner for each possible assignment and multiplying it by the probability of that assignment. Algorithm 1 describes this computation. The algorithm obtains time t and a set of candidate trees CT. For each one of the candidate trees we sort, in lines –11, the nodes whose time is less or equal to t ($NODES_t^j$) according to their expected utility. We sort in an inverse order and insert the ordered nodes into array $s_j[]$. All the arrays are added to the set S. Now we want to iterate over the arrays, and thus we initiate pointers the arrays. $indx[]$ keeps the pointers to the arrays, where $indx[i]$ keeps a pointer to array $s_j[]$. All the pointers are initiated to point to the first node in the corresponding array (lines 12–14). In the main loop (lines 15–19), we find the best node (the node with the highest expected utility) among the nodes that are currently pointed. Since each node that is currently pointed appears in a different candidate tree, we actually pick the one that currently wins (line 16). To compute its probability of winning, we multiply its probability with the probability of the nodes of the other candidates given that s_j is the winner. This means that for each one of the candidate trees we sum over the probabilities of the nodes that have a lower expected utility than the winner (line 17). Finally, in line 19, we increment the pointer of the local winner to the next node to find the winner in the next iteration. We go through this iteration until one of the candidate nodes has been scanned. In line 20 the function subtracts the cost of waiting from exp.

Algorithm 1 EXPECTED_STOPPING
(**input**: time t)
(**input**: candidate trees CT)

1: Internal variables:
2: $S \leftarrow \emptyset$
3: $indx[m]$
4: $i \leftarrow 1$
5: $j \leftarrow 1$
6: $exp \leftarrow 0$
7: $best$
8: **for all** $ct_j \in CT$ **do**
9: $s_j[] \leftarrow sort(\text{t}, ct_j)$
10: $S \leftarrow S \bigcup s_j[]$
11: **end for**
12: **for all** $j \leq m$ **do**
13: $indx[j] \leftarrow 1$
14: **end for**
15: **while** $\forall j \leq m,\ indx[j]$ did not reach to the end of $s_j[]$ **do**
16: $best \leftarrow max(S, indx[])$
17: $exp \leftarrow exp + \mathcal{EU}(s_{best}[indx[best]]) \cdot Pr\mathcal{PTH}(s_{best}[indx[best]]) \cdot \prod_{i \neq best} \sum_{k=indx[i]}^{|s_i[]|} Pr\mathcal{PTH}(s_i[k])$
18: $indx[best] \leftarrow indx[best] + 1$
19: **end while**
20: return $exp - \mathcal{CST}(\sigma^t)$

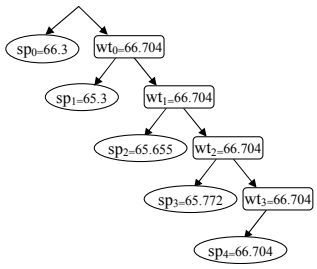

Figure 3: Decision tree based on ct_1 and ct_2.

Let us demonstrate the approximate decision tree by an example. Assume Figure 1 and Figure 2 represent two candidate trees whiele $\mathcal{CST}(\sigma^t) = t$. We generate the decision tree in a bottom-up manner, since each waiting node is actually the maximum of the nodes in the next level. In the last level l_4 there is only one node, sp_4. $NODES_4^1 = \{n_{2,1}, n_{3,1}, n_{5,1}, n_{6,1}\}$, $NODES_4^2 = \{n_{2,2}, n_{3,2}, n_{5,2}, n_{6,2}\}$. The function $sort$ sorts these sets into s_1 and s_2. For clarification, we present the expected utility of the nodes although in the algorithm we use the nodes. $s_1 = [80, 65, 60, 55]$, $s_2 = [75, 70, 45, 40]$. In the first iteration (lines 15–19), $best = 1$ since $s_1[1] = 80 > s_2[1] = 75$. Thus, $exp = s_1[1] \cdot Pr\mathcal{PTH}(n_{2,1}) \cdot (Pr\mathcal{PTH}(n_{2,2}) + Pr\mathcal{PTH}(n_{3,2}) + Pr\mathcal{PTH}(n_{5,2}) + Pr\mathcal{PTH}(n_{6,2})) = 80 \cdot (0.4 \cdot 0.8) \cdot (0.3 \cdot 0.8 + 0.7 \cdot 0.4 + 0.7 \cdot 0.6 + 0.3 \cdot 0.2) = 25.6$. Then the pointer to s_1 is incremented to point on $s_1[2]$. In the next iteration, $best = 2$ since $s_2[1] = 75 > s_1[2] = 65$. Thus, $exp = exp + s_2[1] \cdot Pr\mathcal{PTH}(n_{2,2}) \cdot (Pr\mathcal{PTH}(n_{6,1}) + Pr\mathcal{PTH}(n_{5,1}) + Pr\mathcal{PTH}(n_{3,1})) = 75 \cdot (0.8 \cdot 0.3) \cdot (0.6 \cdot 0.1 + 0.6 \cdot 0.9 + 0.4 \cdot 0.2) = exp + 12.24 = 37.84$.

Finally, $exp = 70.704$ and the expected utility of stopping is $sp_4 = 70.704 - 4 = 66.704$. $wt_3 = sp_4$ since there is no wait node in time t_4. In the same way we calculate sp_3 based on $NODES_3^1 = \{n_{2,1}, n_{3,1}, n_{4,1}\}$, $NODES_3^2 = \{n_{2,2}, n_{3,2}, n_{4,2}\}$: $sp_3 = 65.772$. $wt_2 = max(st_3, wt_3) = 66.704$. The complete decision tree is presented in Figure 3.

At runtime, the decision maker decides to wait or stop according to sp_0 and wt_0. The agent decides to stop if $sp_0 > wt_0$ and then it chooses the candidate with the highest expected utility. If the agent decides to wait, several assignments will occur. At this point the decision tree needs to be recomputed, because some of the nodes become irrelevant.

Analysis

A policy that uses Algorithm 1, implements a pessimistic approach. If the algorithm decides to wait, then we are guaranteed that an optimal algorithm would decide the same. In case that the algorithm decides to stop, it says that the expected gain from stopping is greater than the expected gain from waiting, although by comparing it to the optimal waiting expectation, it could be lower:

Proposition 1 *Given time t and global assignment σ^t, a policy that uses Algorithm 1, maximizes $GEG(W, t, \pi)$ for $\pi(\sigma^t) = wait$. For $\pi(\sigma^t) = stop$, $GEG(W, t, \pi) \leq GEG(W, t, \pi^*)$, where π^* is an optimal policy.*

Proof: By definition, for each time t, Algorithm 1 finds the optimal expected gain of stopping, considering the candidate with the maximal gain for each assignment. The waiting node is the maximum between the wait node and the stop node at the next time level. As a result, it does not take into consideration the combination of waiting and stopping for different assignments. Thus, the wait node's value, wt_0, is less or equal than the optimal expected gain from waiting. If $\pi(\sigma^t) = wait$ it means that $wt_0 > st_0$, and thus the optimal expected gain from waiting is also higher than st_0. That means that our algorithm guarantees the optimal expected gain from waiting. If $\pi(\sigma^t) = stop$ it means that $sp_0 > wt_0$, and since the wait node's value, wt_0, is less or equal than the optimal expected gain from waiting, it may be that the optimal expected gain from waiting is greater than sp_0.

We now prove the approximated error. The approximated error depends on the time the policy decides to stop. It is actually the cost of waiting from the current time stamp to level l_{n-1}, while l_n is the last level.

Theorem 2 *Given time horizon $T = [a, b]$, and $t' \in T$, where $\pi(\sigma^{t'}) = stop$, the absolute approximation error ϵ of the policy that uses Algorithm 1, is $\epsilon < \mathcal{CST}(\sigma^{t_n}) - \mathcal{CST}(\sigma^{t'+1})$, where n is the height of the decision tree.*

Proof: In the last level l_n, there is only one stop node. Since the stop nodes are optimal and since the wait node $wt_{n-1} = sp_n$, then wt_{n-1} is also optimal. At time level t', the optimal value of the expected gain from waiting is $wt_{n-1} + (\mathcal{CST}(\sigma^{t_n}) - \mathcal{CST}(\sigma^{t'+1}))$. In that case, this will be the value of the stop node at time t'. The obtained value of the expected gain from waiting, wt_0 at time t', is the maximal value between sp_1 and wt_1. This value is less than the optimal value and greater or equal than wt_{n-1}. If $\pi(\sigma^t) = stop$, then $sp_0 > wt_0$ and thus $sp_0 > wt_{n-1}$. As a result, the absolute approximation error ϵ, is $\epsilon < \mathcal{CST}(\sigma^{t_n}) - \mathcal{CST}(\sigma^{t'+1})$.

Note that this approximated error will be obtained in rare cases. Let us examine the upper bound case in which the error is $\mathcal{CST}(\sigma^{t_n}) - \mathcal{CST}(\sigma^{t'+1})$. Note that this case could not actually be obtained, since in this case the sp_0 would obtained this value. The meaning of this hypothesized case is that there is no assignment that justifies waiting. This will happen only in case that there is an absolute winner for every possible assignment.

Finally, we prove the complexity of Algorithm 1.

Proposition 2 *The complexity of Algorithm 1 is $O(m^2 \cdot M + m \cdot M \log M)$, where m is the number of candidates and M is the maximal size among the candidate trees.*

Proof: In time stamp t_i the algorithm sorts the nodes in the set $NODES_{t_i}^j$ for each candidate tree ct_j. Since the maximum number of nodes in $NODES_{t_i}^j$ is M, the worst case complexity of this sort is $M \log M$. We perform this sort for each candidate tree, thus the complexity is $O(m \cdot M \log M)$. To compare between the sorted sets in set S, the algorithm goes over the candidates and finds the maximum among the pointed nodes of the candidates. This computation is m^2. The algorithm stop once it reaches to the end of one candidate's array (line 15). The worst case is M. Finding the $Pr\mathcal{PTH}$ of each node could be calculated once before the loop with complexity of $m \cdot M \log M$. Thus the worst case complexity of Algorithm 1 is $O(m^2 \cdot M + m \cdot M \log M)$.

Empirical Evaluation

We experimentally validated our algorithm within a systematic artificial framework inspired by the stock market. We extensively varied the number of candidate stocks (2-30) and the time horizon of the economic events (1-5) (i.e., the timed variables). The waiting cost of all the events was fixed to a constant value of $\$2.8K$ for each event. We ran each combination 625 times. In each test the distribution of the economic events outcome as well as the possible profits of the stocks (the utility) were randomly selected from a range of $[\$10K \ldots \$100K]$.

We compared our algorithm (APPROX) to: (1) an optimal solution (OPT); (2) the heuristic presented in (Kalech and Pfeffer 2010) (HEURIST), and to two baseline algorithms; (3) a stopping strategy: determining the winning candidate at the beginning based only on the expected utility (STOP); and (4) a waiting strategy: determining the winning candidate at the end based on full information (WAIT).

We compared the above algorithms using two metrics: (1) runtime, and (2) the outcome utility. To normalize the utility, we divided it by the utility obtained by an omniscient decision maker with no cost.

Due to space limitations, we only present a subset of the results with a time horizon of 5 levels. Figure 4 presents the utility for a test setting of up to six candidates. Due to memory limitations the optimal algorithm failed for larger candidate sets. The utility obtained by our algorithm is very close to the optimum and actually the difference between them is not significant. This result is much better than the results of the baseline algorithms and even those of the heuristic algorithm. The runtime of all the algorithms is polynomial,

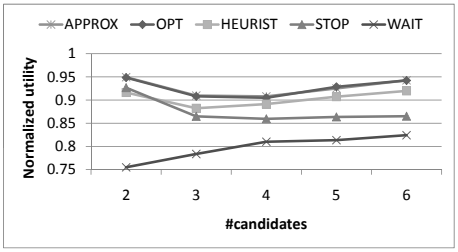

Figure 4: Normalized utility over 6 candidates.

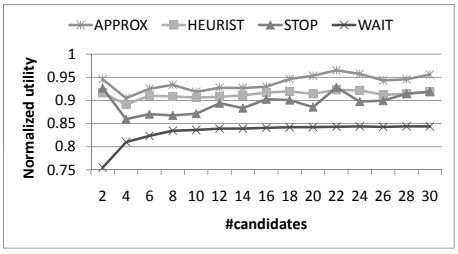

Figure 5: Normalized utility over 30 candidates.

except for the optimal algorithm which is exponential. For instance, the average runtime of the optimal algorithm for six candidates is 5836 milliseconds, while that of the other algorithms is less than two milliseconds.

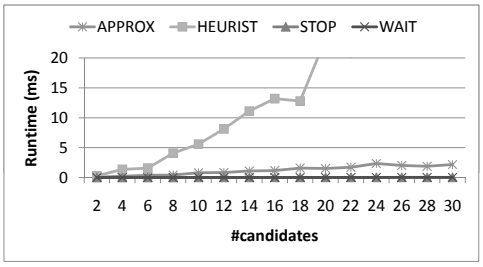

Figure 6: Runtime over 30 candidates.

Other than for the optimal algorithm, we further compared algorithms for larger sets of up to 30 candidates. The utility of our algorithm was always significantly better than the others, as shown in Figure 5. Surprisingly, although both our algorithm as well as the heuristic, are polynomial, our algorithm is also better than the heuristic in terms of runtime, as shown in Figure 6.

To illustrate the significance of these results, consider the stock market and five candidate stocks. Based on our experiments, the average profit of the optimal algorithm is $\$92.8K$, which is 92.8% of the optimum. Our algorithm's profit is, on average, less than the optimal only in $\$300$, while the heuristic reduces the profit in $\$2800$. Obviously the baseline algorithms reduce the profit drastically. The wait strategy, for instance, produces a profit of $\$11,300$.

Finally, Figure 7 shows the pessimistic attribute of our algorithm. The y-axis represents the depth (in percentage relative to the maximal horizon) in which the algorithms stop and decide. As analyzed, our algorithm always stops at most at the depth of the optimal algorithm, since the waiting decision of our algorithm is optimal.

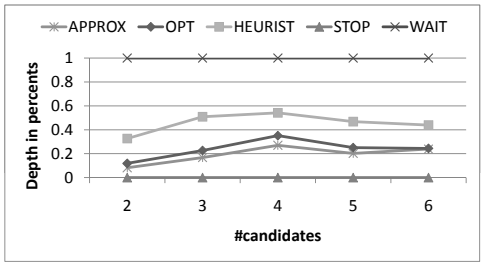

Figure 7: Normalized depth over 6 candidates.

Summary and Future Work

In this paper we presented the problem of decision making with dynamic information. We focused on the question of when to stop and make a decision that maximizes the utility, where there is a cost for waiting. We proved that this problem is NP-hard, and thus presented an approximation algorithm that is pessimistic and polynomial in the number of candidates. We proved that our approximation computes the optimal expected utility when stopping. Empirical evaluation of our algorithm showed no significant difference between the outcome utility of the optimal algorithm and ours, yet both are significantly better than a previous algorithm (Kalech and Pfeffer 2010).

In the future we plan to further investigate this problem in domains involving multi-agent decision making, where multiple agents should share the same decision, based on different variables and utilities. Basically, a multi-agent version of our approximation grows exponentially in the number of agents and thus we plan to reduce this complexity.

References

Bilgic, M., and Getoor, L. 2007. Voila: efficient feature-value acquisition for classification. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, 1225–1230. AAAI Press.

Cook, S. A. 1971. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, 151–158. New York, NY, USA: ACM.

Horvitz, E., and Rutledge, G. 1991. Time-dependent utility and action under uncertainty. In *In Proceedings of Seventh Conference on Uncertainty in Artificial Intelligence*, 151–158. Morgan Kaufmann.

Kalech, M., and Pfeffer, A. 2010. Decision making with dynamically arriving information. In *Proceedings of the 9th international joint conference on Autonomous agents and multiagent systems (AAMAS-10)*.

Krause, A.; Singh, A. P.; and Guestrin, C. 2008. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research* 9:235–284.

Peskir, G., and Shiryaev, A. 2006. *Optimal Stopping and Free-Boundary Problems*. Birkhäuser Basel.

Radovilsky, Y., and Shimony, S. E. 2008. Observation subset selection as local compilation of performance profiles. In *UAI*, 460–467.