Decision Making with Dynamically Arriving Information

Meir Kalech[1]
Avi Pfeffer[2]
[1]Information Systems Engineering Department, Ben-Gurion University, Israel, kalech@bgu.ac.il
[2]Charles River Analytics, USA, apfeffer@cra.com

ABSTRACT

Decision making is the ability to decide on the best alternative among a set of candidates based on their value. In many real-world domains the value depends on events that occur dynamically, so that the decision is based on dynamically changing uncertain information. When there is a cost to waiting for more information, the question is when to make the decision. Do you stop and make the best decision you can, given the information you have so far, or do you wait until more information arrives so you can make a better decision? We propose a model that characterizes the influence of dynamic information on the utility of the decision. Based on this model, we present an optimal algorithm that guarantees the best time to stop. Unfortunately, its complexity is exponential in the number of candidates. We present an alternative framework in which the different candidates are solved separately. We formally analyze the alternative framework, and show how it leads to a range of specific heuristic algorithms. We evaluate the optimal and the simplest heuristic algorithms through experiments, and show that the heuristic algorithm is much faster than the optimal algorithm, and the utility of the winner it finds is close to the optimum.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents*

General Terms

Algorithms, Theory

Keywords

Agent Reasoning::Reasoning (single and multi-agent), Economic paradigms::Electronic markets

1. INTRODUCTION

How to make decisions when faced with dynamically changing information is an important problem. Do you stop at a particular point and make the best decision you can, given the information you have so far, or do you wait until more information arrives so you can make a better decision? When there is a cost to waiting, this problem becomes nontrivial. As an example, consider a meeting scheduling system. Determining the best time for a meeting could depend on many factors like other meetings, location and attendees. Typically, these factors may change dynamically. The longer one waits, the more information becomes available, and the higher the probability of choosing the best time. However, waiting to make the decision could be associated with a cost, for instance because the chosen time slot might no longer be available. The challenge of this research is to determine the best time to make the decision, i.e., the time that maximizes the utility (which increases in the certainty of the information) and minimizes the cost.

In this paper, we develop a model for representing the arrival of dynamic information and its influence on the utilities of the candidates. We present an optimal algorithm that guarantees the best decision, trading off certainty for waiting cost. We show that, unfortunately, the complexity of this algorithm is exponential in the number of candidates. We therefore develop an alternative heuristic framework for solving the problem in which the different candidates are solved separately. We formally analyze this framework and show that the calculations it performs are exactly correct, but that it approximates the true solution by considering different policies from the optimal algorithm. We then discuss how this framework translates into a suite of heuristic algorithms, depending on how a key probability is computed. We show that the complexity of the simplest heuristic algorithm is only polynomial in the number of candidates.

We compare the simplest heuristic algorithm to the optimal algorithm and two more baseline algorithms, one that makes the decision in the beginning and the other that makes the decision after obtaining the whole information. We evaluate the algorithms in terms of quality of the utility of the decision and runtime. We show that the heuristic runs much faster than the optimal algorithm, and the utility of the winner it finds is close to the optimum.

Cite as: Decision Making with Dynamically Arriving Information, Meir Kalech, Avi Pfeffer, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 267-274

2. RELATED WORK

Our work is related in some aspects to Horvitz's work [8, 7] 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 to mostly satisfy the task? Horvitz presents the use of an expected value of computation to determine the best time to stop. The major difference is that whereas in that work the goal is to execute a single task, in our work we are seeking to select a candidate out of multiple candidates. In particular, we want to do this in a way that scales reasonably with the number of candidates.

Another class of work addresses monitoring anytime algorithms [17], which search for the best possible answer under the constraint of limited time and/or resources. One question for this class of algorithms is how to optimally decide the time to stop. For instance, Finkelstein and Markovitch [4] developed algorithms that design an optimal query schedule to detect satisfaction of a given goal. Their aim is to minimize the number of queries (which are time

consuming) to reach the goal. We, too, propose to minimize the amount of information. However, in our problem we do not satisfy a *known* goal or task but we search for the best candidate that maximizes the utility. This difference is significant, since in anytime algorithms there is only a single decision about whether to stop or whether to continue a single algorithm, rather than to attempt to select a candidate among multiple candidates.

Hansen and Zilberstein [6] developed a framework for run-time monitoring of anytime algorithms. In this framework, they considered the question of when to stop a deliberation based on various factors like time, cost of delay and quality of solution. In our work, on the other hand, we do not propose to investigate the right time for a computation task, but rather we plan to find the right time to increase the likelihood of the selection of the best candidate.

Another series of papers that address decision making under uncertainty relate to multi-attribute decision making (MADM) [16, 3]. In MADM, the decision maker evaluates the candidates' outcome based on given attributes. There are several works relating to MADM under uncertainty, but most of them do not deal with attributes that change dynamically. For instance, Keeney and Raiffa [11] address uncertainty of the utility function. Lahdelma and Salminen [12, 13] focus on measurement of uncertainty attributes. Many works ([15, 10, 9]) deal with incomplete information about the attributes' weight. The challenge in this paper is, however, to determine the best candidate when the attributes (in our terms information) are changing dynamically.

Finally, the tradeoff between uncertainty and cost relates to the optimal stopping problem (OSP) [2, 14]. In both 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 in our problem is based on multiple alternatives, while in OSP, when to stop is made only for one alternative. Multiple alternatives increase the complexity exponentially.

3. MODEL DESCRIPTION

To motivate the model we use an example of a simple stock market. Assume an agent's goal is to buy the best stock among three stocks (c_1, c_2, c_3). The current value of each stock is obviously known but will change over time. For instance, stocks c_1 and c_2 may be affected by the interest decreasing in February. c_1 may also be affected by the publication of the company's balance sheet in April; the same for c_3 in March. Finally, c_2's value may be affected by a launching of a new technology in May and at the same time c_3's company is expected to sell real estate that may influence the stock value.

The agent cannot evaluate the influence of the future events on the stocks for certain, but with some probability. Obviously, the sooner the agent makes the decision the less it loses by not investing its money. On the other hand, the more time it waits the more information it gathers by knowing the outcome of the expected events, and the more certain decision it could make.

In our model, each decision will be designated by a *candidate*, and throughout the paper we will refer to the candidate set $C = \{c_1, c_2, ..., c_n\}$. 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 (TIMED VARIABLE). *A* variable *consists of a discrete, finite random variable X taking values $x_1, ..., x_n$, and a time stamp $\Gamma(X)$.*

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 timed 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 $\mathcal{U}(n)$.*

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. The variable space V is the union of all the candidates' sets: $V = \bigcup_i V_i$. Two candidates can be affected by the same timed variable; we call such a variable a *common variable*. If a variable is a common variable, it must have the same time stamp and probability distribution over assignments in all trees in which it appears. The interest decreasing, for example, is a common variable since both c_1 and c_2 are affected by it.

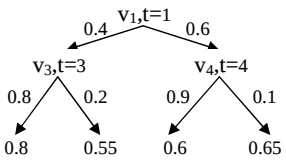

Figure 1: Candidate tree ct_1.

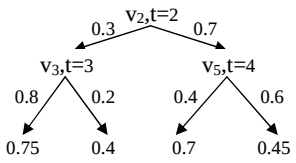

Figure 2: Candidate tree ct_2.

Figures 1 and 2 present two candidate trees. For instance, the variable labeled $v_3, t = 3$ in ct_1 represents variable v_3 at time $t = 3$. The probability of its left edge is 0.8 and of its right edge is 0.2. The value 0.8 in the left leaf of ct_1 represents the utility of the path $\{v_1, v_3\}$ through probabilities 0.4 and 0.8. This root to leaf path represents one possibility of assignments sequence that determines the utility of the candidate. Variable v_3 represents a common variable for c_1 and c_2 For this reason, their outgoing edges have the same probability distribution. We assume that different variables are independent, and the only interaction between different candidates is via common 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:*

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

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 $\mathcal{CST}(a)$.*

The cost of a node is the sum of the waiting costs of assignments in the path to the node.

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.*

The cost of node n is $\sum_j \mathcal{CST}(x_j \in \mathcal{PTH}(n))$. The expected gain is the difference between the expected utility and the cost:

DEFINITION 7 (EXPECTED GAIN). *Given node $n \in ct_i$,*

$$\mathcal{GN}(n) = \mathcal{EU}(n) - \sum_j \mathcal{CST}(x_j \in \mathcal{PTH}(n))$$

There is a tradeoff between 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 algorithms to find the time that maximizes the gain. One might be tempted to define the optimal decision problem as finding in advance the best time to stop, and when that time is reached choosing the candidate with the highest expected utility at that time. However, this is incorrect. Such a definition amounts to determining the stopping time in advance, before any assignments have happened, and using the same stopping time no matter what assignments happen. Instead, the decision of whether to stop at time 2 may depend on assignments that happen at time 1. Therefore we define a policy to determine what to do in all situations the decision maker might face.

DEFINITION 8 (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.

One option to represent our problem would be to use Markov Decision Processes (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.

The usual advantage of an MDP formulation is to be able to use dynamic programming methods like value iteration and policy iteration. However, in our problem dynamic programming has no benefit because the same state cannot be reached by different paths, and the number of states is exponential in the total number of variables in all the trees. One of the methods to address large MDPs is factored MDPs [1, 5]. This approach is not viable in our domain because the utility of stopping is a maximization over the utilities in all the trees, which depends on all the variables. As we will show in Section 5, there is a special structure in our problem that is not readily apparent in the MDP or factored MDP formulation.

4. OPTIMAL ALGORITHM

The optimal gain can be calculated by a decision tree approach. The optimal decision tree merges the candidate trees into a single decision tree whose depth is the maximal time of the variables in the candidate trees. In this decision tree, there are three kinds of nodes:

Decision nodes, in which the decision maker must decide whether to stop or to wait, and if to stop, which candidate to choose.

Stop nodes, where the decision maker has stopped and chosen one of the candidates.

Wait nodes, where the decision maker has decided to wait.

Each node is marked with a time stamp. Edges leading out of wait nodes are labeled by conjunctions of assignments. Every node in the tree is marked by a set of assignments, which are the assignments on the path leading up to the node.

The tree is constructed as follows:

1. The root is a decision node with time stamp 0.
2. The children of decision nodes with time stamp t are the wait node with time stamp t, and all the stop nodes for all candidates with time stamp t. If t is the final time step, no wait node child is included.
3. Stop nodes are leaves of the tree. The label of a stop node is the gain of stopping and choosing that candidate, given the assignments that have happened so far, and taking into account the cost of waiting until that time. However, the cost is not equal to the sum of costs of all assignments in the path leading to the node in the optimal tree, because assignments that are not part of the candidate tree of the winning candidate are not considered. Instead, if the stop node corresponds to node $n \in ct_i$, the value of the node is $\mathcal{GN}(n)$ (Definition 7).
4. The children of a wait node with time stamp t are determined as follows:
   (a) Let A be the assignments on the path leading to the node.
   (b) Let V_i be the timed variables in candidate tree ct_i with time stamp $t+1$ such that all assignments on the path to V_i in ct_i are in A.
   (c) Let $V = \bigcup_i V_i$.
   (d) For each *joint* outcome of the variables in V, the wait node has a child, labeled by the set of assignments comprising that outcome. The child is a decision node with time stamp $t+1$. The branch to the child is labeled by the product of the probabilities of those assignments (since we assume the variables are independent).

Once the tree has been constructed, it can be evaluated using a simple bottom-up process. The gains at the leaves, i.e. the stop nodes, have already been calculated. The gain of a wait node is the expectation of the utilities of its children. The gain of a decision node is the maximum gain of its children, and the optimal decision is the one that leads to maximal gain. The decision tree is generated and evaluated in advance, before any assignments have happened. Its solution represents a policy (Definition 8).

Figure 3 presents the optimal decision tree for a decision problem with candidate trees ct_1 (Figure 1) and ct_2 (Figure 2). The

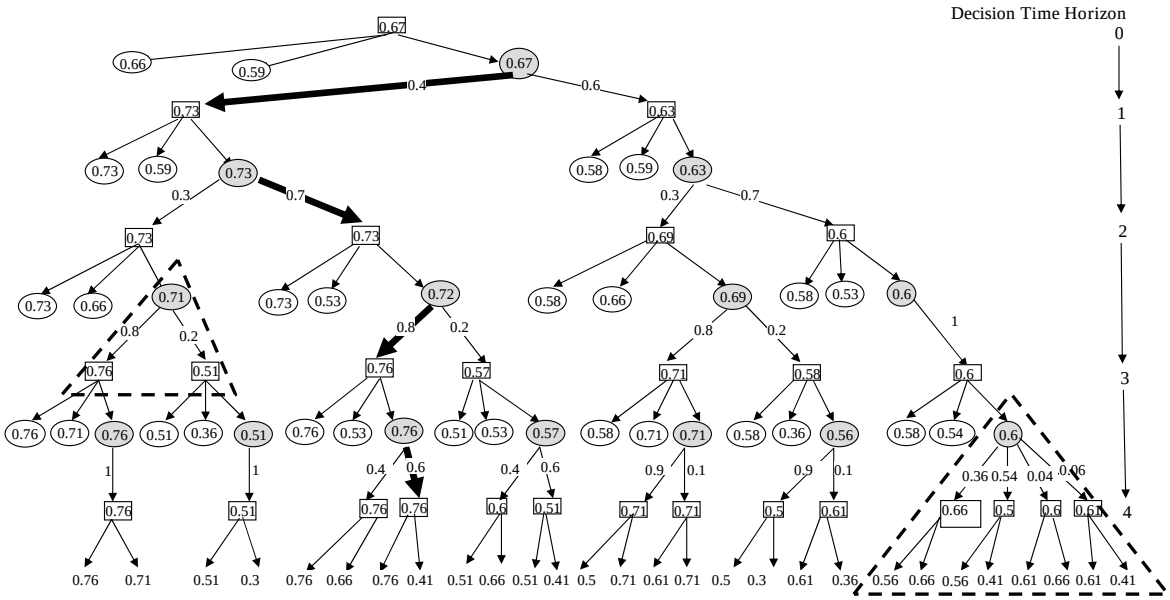

Figure 3: Optimal decision tree for ct_1 and ct_2 candidate trees.

axis on the right of the graph represents the time horizon of the decision. The rectangle nodes represent decision nodes, the shaded ellipse nodes represent wait nodes and the empty ellipse nodes represent stop nodes.[1] The stop nodes come in pairs, one for each candidate; the node for c_1 is on the left. The numbers in the nodes represent expected gains, computed using the bottom up algorithm. We used the cost function in which the cost of every event is 0.02, so the cost of reaching a leaf is 0.04 (since two events happen on the path to every leaf).

For example, consider the dashed triangle on the right hand side of the figure. The root of the subtree shown in this triangle is a wait node with time stamp 3. In determining the children of this node, we consider all variables in the candidate trees with time stamp 4. There are two such variables, v_4 for candidate c_1 and v_5 for candidate c_2. We need to split on all joint outcomes of these two candidates, so the wait node has four children. Each of these children is a decision node with time stamp 4. Since this is the last time stamp, these decision nodes only have stop nodes as their children.

For a second example, consider the dashed triangle on the left hand side of the figure. v_3 is a common variable; although both candidates split on a variable at this point, since it is a common variable we only need to split on the outcomes of a single variable in the optimal tree (the only two consistent paths). Thus common variables can help make the optimal tree smaller.

In bold we have shown one particular course of events. At the decision node at the root, the gain of the wait node child (0.67) is higher than that of the best stop node child (0.66) so the agent waits. Then, in this course of events, the assignment $v_1 =$ left happens. At the next decision node child, the gain of stopping with candidate c_1 (0.73) is as high as that of waiting (0.73), so the agent may stop and chooses c_1.

Assume the sequence of assignments happened as shown in bold in the figure. This leads eventually to a leaf node whose gain is shown as 0.76. Note, however, that this gain incorporates the cost of waiting two assignments to make the decision. Now, if the agent had been omniscient and known the outcomes of variables in advance, it would have obtained gain 0.8, since it would not have to wait at all. For a realistic agent who stops in time 1, the gain for choosing c_1 is $0.8 - 0.02 = 0.78$ (since one event has happened).

To determine the complexity of the optimal algorithm, consider first the case where there are no common variables. Since there are no common variables, every path in one candidate is consistent with every path in another candidate. Thus the optimal tree will contain a path corresponding to every combination of paths in all candidates. Let the maximum size of a candidate tree be M and the number of candidates be n. Then the worst-case size of the optimal tree is $O(M^n)$.

Now, consider the opposite extreme, where all candidates share the same variables. One might think that in this case the size of the optimal decision tree is polynomial in the number of candidates. Unfortunately this is not the case. The problem is that even though the candidates share variables, the variables may appear at different places in the trees. To illustrate, suppose we have two candidates ct_1 and ct_2 with three variables v_1, v_2 and v_3. v_1 is the root in both candidates. In ct_1, v_2 is the left child of v_1 and v_3 is the right child, while in ct_2 it is the other way around. The paths $[v_1 =$ left, $v_2 =$ left$]$ and $[v_1 =$ left, $v_2 =$ right$]$ in ct_1 are both consistent with $[v_1 =$ left, $v_3 =$ left$]$ and $[v_1 =$ left, $v_3 =$ right$]$ in ct_2, so we have to consider all their combinations in the optimal tree and the complexity remains exponential in the number of candidates. It is clear that this example can be extended to arbitrarily many candidates by widening the trees.

The best-case scenario is where all candidates share the same variables in the same order. In this case the size of the optimal tree grows polynomially in the size of the candidates' trees.

5. SOLVING EACH CANDIDATE SEPARATELY

The optimal algorithm presented in Section 4 considers all candidates simultaneously. In this section, we develop an alternative

[1]To keep the figure readable, the stop nodes at the final time do not have ellipses. Also, we have omitted assignment labels on the edges.

algorithmic framework, in which the candidates are considered separately. This alternative viewpoint will lead us to more efficient heuristic algorithms in Section 6.

The main idea behind the alternative framework is that each candidate makes a separate contribution to the overall utility. Specifically, a candidate only contributes to the overall utility when the candidate actually wins. Thus we can estimate the value of reaching a node in a candidate tree in the future by the expected gain of the node times the probability that the candidate will win given that the node is reached.

DEFINITION 9 (PROBABILITY OF WINNING). *Let n be a node in candidate tree ct_i with time t. The probability of winning for c_i at n, denoted $Pr(c_i \text{ wins}|n)$ is the probability that the expected utility of n is greater than that of every other candidate at time t, given the assignments on the path leading to n.*[2]

DEFINITION 10 (RELATIVE EXPECTED GAIN). *The relative expected gain of node n in candidate tree ct_i is $\mathcal{GN}(v)Pr(c_i \text{ wins}|v)$.*

In the alternative framework this value is estimated by constructing a separate decision tree for each candidate as follows. The individual candidate decision trees are generated in a manner similar to the optimal decision tree, except that **relative expected gain** is used instead of **expected gain**. Each of the decision trees for each of the candidates is solved in a manner similar to the optimal tree. At this point, if we consider the stop values at time 0, exactly one candidate will have probability 1 of winning and all others will have probability 0, since there is no uncertainty about which candidate will be chosen if the decision is taken immediately. Therefore the agent can compare the expected gain of the immediately winning candidate with the sum of the values of the wait node children of the roots of all the trees. Intuitively, the value of a wait node for a candidate estimates that candidate's contribution to the benefit of waiting, so if the sum of the wait values is greater than the maximum immediate expected gain, that means that the total expected gain of waiting is greater than the expected gain of stopping, in this case the decision will be to wait. Otherwise, the decision will be to stop and choose the candidate with the highest expected gain.

If the agent decides to wait, some assignments will happen. At this point the decision trees need to be recomputed. Some parts of the candidate trees will be inconsistent (see Definition 2) with the assignments. Therefore the candidate trees can be pruned only to include the consistent assignments. This may change the probability of winning, because the event of winning at some future node is conditioned on the assignments that have already happened.

Before analyzing the algorithm we present some definitions. The first definition describes the relationship between a global assignment and a path through an individual candidate tree. Intuitively, the projection of a global assignment onto a candidate is the path in the individual candidate tree resulting from that assignment.

DEFINITION 11 (PROJECTION). *The* projection *of a time t global assignment σ^t onto candidate c_i, denoted σ_i^t, is the maximal path of assignments through ct_i such that each assignment is in σ^t. Such a path is called a* local sequence. *The set of all global assignments whose projection is σ_i^t is denoted by $C(\sigma_i^t)$.*

The next definitions describe the relationship between a global policy and a set of policies for individual candidates.

[2]We assume that ties between candidates are broken in a consistent manner.

DEFINITION 12 (LOCAL POLICY). *A* local policy *for candidate c_i is a rule that dictates either stopping or waiting for each local sequence σ_i^t. For contrast, we will sometimes use the term* global policy *to mean a policy.*

DEFINITION 13 (COMPATIBLE). *A global policy π is* compatible *with a set of local policies $\pi_1, ..., \pi_n$ if, for every global assignment σ^t, and the projection σ_i^t for each candidate c_i, $\pi(\sigma^t) = \pi_i(\sigma_i^t)$.*

Some global policies are not compatible with any set of local policies, and some sets of local policies are not compatible with any global policy. For a set of local policies to be compatible with a global policy, if the same sequence appears in more than one of the candidate trees, all the local policies for those candidates must make the same decision for that sequence. If this does not hold, there can be no global policy that agrees with all the local policies. For a global policy to be compatible with a local policy, it must make the same decision for all global assignments that are consistent with a local sequence appearing in one of the candidates. If this does not hold, the global policy will specify multiple decisions for the sequence. In the case that we have a global policy and compatible local policies, we show that computing the expected utility in the optimal and candidate-based algorithms produces the same result (see the proof in the Appendix):

THEOREM 1. *Let π be a policy and $\pi_1, ..., \pi_n$ a compatible set of local policies. Let $\mathcal{GN}(\pi)$ denote the expected gain of policy π as computed in the optimal decision tree, and let $\mathcal{GN}_i(\pi_i)$ denote the expected gain of local policy π_i computed in the decision tree constructed for the candidate c_i by the alternative algorithm. Then $\mathcal{GN}(\pi) = \sum_i \mathcal{GN}_i(\pi_i)$.*

This result yields insight into the operation of the approximate algorithm. Splitting up the calculations into the individual candidate trees is in fact correct. What makes this algorithm approximate is that by focusing on local policies, and then extracting a global policy from those, it forces a different set of policies to be considered when considering the expected gain of waiting until a node is reached in the future. On the one hand, it does not allow a decision for an individual candidate to depend on assignments that only affect other candidates, which may cause the algorithm to miss possibilities that would give it higher utility and underestimate the expected gain. On the other hand, it allows different candidates to make incompatible decisions in the future, which might cause it to overestimate the expected gain.

6. COMPUTING PROBABILITY OF WINNING

A key step in the candidate-based algorithm is computing the probability that a candidate wins, given that a certain node in the candidate tree is reached. To compute the probability of winning, we note that: $Pr(c_i \text{ wins}|n) = Pr(\bigwedge_{j \neq i} c_i \text{ beats } c_j|n)$.

In principle, the events on the right hand side of this equation are not independent, since the different candidates may share common variables. Therefore, if were to compute this probability exactly, the complexity of the algorithm might still be exponential in the number of candidates. Therefore we need to approximately compute this probability. The simplest approximation is to assume the events of beating different candidates are independent. So we get

$$Pr(c_i \text{ wins}|n) \approx \prod_{j \neq i} Pr(c_i \text{ beats } c_j|n)$$

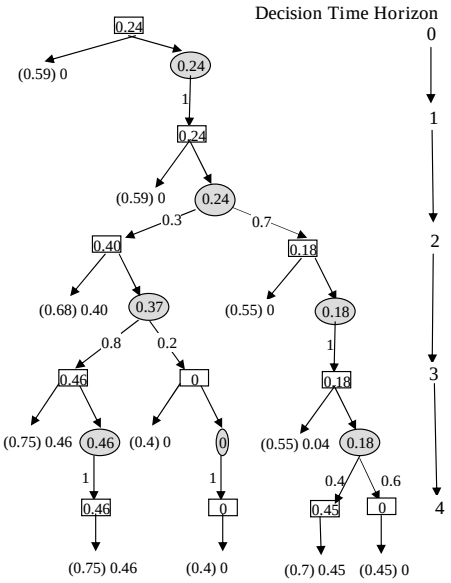

Figure 4: Candidate decision tree for candidate c_2.

Now, to compute $Pr(c_i \text{ beats } c_j|n)$, we consider nodes n_j in ct_j that are consistent (see Definition 2) with n. Not all such nodes are considered; they must have the right time. Let the time stamp of the variable at node n be t. A node has the right time if its variable's time stamp is at least as great as t, and its parent's time stamp is not. This means that the node's variable is the first thing to be decided after time t. Therefore, if the decision maker had to make the decision about candidate c_j at time t, it would use the expected gain of the subtree rooted at that node. We will use $ok(n_j|n)$ to indicate that n_j is consistent with n and has the right time. For each such node n_j, if the expected utility of n is greater than n_j, we add n_j's probability to the probability that c_i beats c_j. Formally,

$$Pr(c_i \text{ beats } c_j|n) = \frac{\sum_{(n_j \in ct_j : ok(n_j|n) \text{ and } \varepsilon u(n) > \varepsilon u(n_j))} P(n_j|n)}{\sum_{(n_j \in ct_j : ok(n_j|n))} P(n_j|n)}$$

It remains to compute $P(n_j|n)$. This is the product of the probabilities of assignments along the path to n_j that do not appear along the path to n.

Figure 4 presents the decision tree of ct_2 (Figure 2). The leaves contain two numbers, the first represents the expected utility and the second (in bold) represents the relative expected gain.

For example, let us compute the relative expected gain of the leftmost bottom-level node. This node is reached after v_2 and v_3 both came out left, and waiting till $t = 4$. At this point, there are three nodes in ct_1 consistent with this node, with utilities 0.8, 0.6 and 0.65. This node, with utility 0.75, beats the latter two nodes, whose total probability is 0.6. The total probability of all nodes in ct_1 consistent with this node is $0.6 + 0.4 * 0.8 = 0.92$, so the probability of winning is $0.6/0.92 = 0.65$. To compute the relative expected gain of this node (see Definition 10), we multiply the gain, which is $0.75 - 0.04 = 0.71$, by the probability of winning, to get 0.46.

If we compute the decision tree of candidate c_1 too, we find that the optimal choice at time 0 is to wait. Suppose that variable v_1, determined at $t = 1$, came out left. The decision trees are now updated. Candidate c_1's tree is pruned so that it only includes the subtree rooted at v_3. Consider, for example, the third leaf of ct_2 in Figure 2, whose utility is 0.7. In estimating its probability of winning, we see that candidate c_1 will have, at this time, utility 0.8 with probability 0.8 and utility 0.55 with probability 0.2. Thus c_2's probability of winning is 0.2 (in which $0.7 > 0.2$). Now consider the leftmost leaf in Figure 2, with utility 0.75. Because v_3 is a common variable, the only node in ct_1 consistent with this leaf is the leftmost leaf in Figure 1, with utility 0.8. Thus c_2's probability of winning is now 0.

If we use this approximation of the probability of winning, the complexity of our heuristic algorithm is only polynomial in the number of candidates since we build a decision tree for every candidate separately. To be precise, again let the maximum size of a candidate tree be M and the number of candidates be n. When evaluating each candidate tree, we must compute probability of winning at $O(M)$ nodes. For each such node, we perform a summation over $O(M)$ nodes in each of the other candidate trees. We must do this for all n candidate trees. Thus the total cost of the algorithm is $O(M^2n^2)$. This compares favorably to $O(M^n)$ for the optimal algorithm if the number of candidates is large.

Assuming complete independence between the candidates is only the simplest possibility that results in the fastest runtime. An alternative is to compute the probabilities exactly, perhaps using an algorithm such as variable elimination. In the worst case this computation is exponential in the number of common variables. One might therefore try alternative ways of approximating the probability. We do not go into these alternatives in this paper, but note that our framework is general enough to accommodate many such approaches. Finally, we remark that in the case where there are no common variables, assuming independence is exactly correct, but the algorithm is still a heuristic for the reasons described in Section 5.

7. EMPIRICAL EVALUATION

We evaluated our algorithms considering two metrics: runtime and the utility of the candidate. To normalize the utility, we divided it by the utility obtained by an omniscient agent with no cost. We compared the optimal algorithm to the heuristic algorithm in which the probability of winning is computed assuming the events of beating all candidates are independent. We also compared them to two baseline algorithms: (1) stopping strategy: the agent determines the winning candidate at the beginning based only on the expected utility; (2) waiting strategy: the agent determines the winning candidate at the end based on full information.

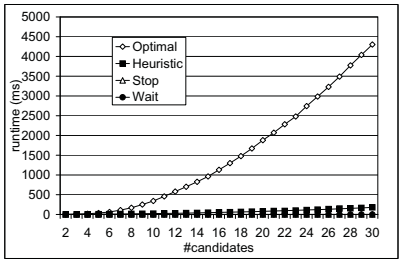

Figure 5: One third common variables: Runtime over the number of candidates.

We conducted a simulation inspired by the stock market with a wide variety of conditions. The utility of each stock was evaluated

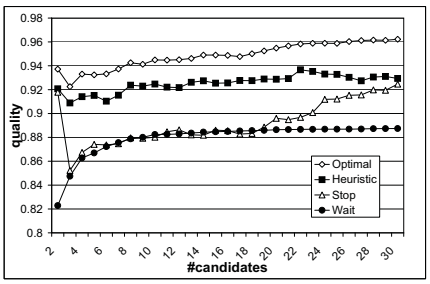

Figure 6: One third common variables: Quality of the decision over the number of candidates.

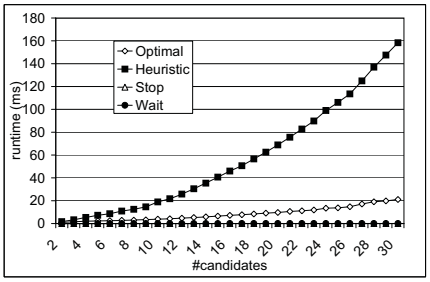

Figure 7: All common variables: Runtime over the number of candidates.

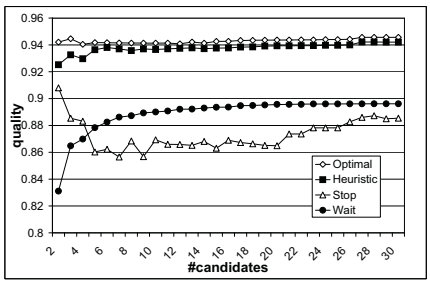

Figure 8: All common variables: Quality of the decision over the number of candidates.

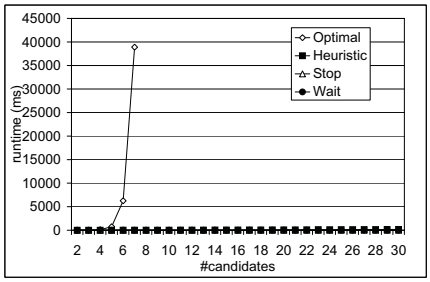

Figure 9: No common variables: Runtime over the number of candidates.

by its profit. The timed variables were represented by economic events like interest decreasing that influence the stocks' value. We ran simulations with both common variables and independent variables. The waiting cost of all assignments was fixed to a constant K, representing the cost of waiting one time unit. In the simulation we fixed K to be two orders less than the expected value of the stocks.

We simulated varying stock market settings, in particular we varied the number of stock candidates (2-30), the time horizon of the economic events and the constant cost K. For space limitation, we present only a subset of the results. In these results, we fixed the time horizon to four time units, and the constant cost to $K = 2.8$. In the context of the stock market simulation the meaning is that for each time unit of one month the cost of waiting is a loss of \$2.8K, where the maximum profit is \$100K.

We set experiments in which a third of the economic events are common variables. We measured the runtime to compute when to make the decision and the quality of the decision. Figure 5 presents the runtime in milliseconds over the number of candidates. Each data point is an average of 900 tests. As analyzed in Section 4 the runtime of the optimal algorithm grows exponentially in the number of candidates but only polynomially for the heuristic algorithm. Figure 6 presents the quality of the decision over the number of candidates. The heuristic algorithm is better than the baseline algorithms and close to the optimal algorithm.

To illustrate the results in the context of the stock market, the maximum profit is $\$100K$ from the point of view of an omniscient agent. The optimal algorithm for 14-18 candidates achieves 95% of the maximum. For the same number of candidates the heuristic achieves 92.5% which is less by $\$2500$ than the optimal algorithm. However, the stopping and the waiting baseline algorithms achieve only 88.5% which is less by $\$6500$ than the optimal.

We have further run our algorithms in two extreme environments with (1) worst case: no common variables and (2) best case: all variables are common and in the same locations in the candidate trees. In the last environment, the runtime of the optimal algorithm decreases significantly by two orders compared to the environment with one third common variables and is even faster than the heuristic (Figure 7)[3]. However, its utility slightly decreases and the utility of the heuristic slightly increases (Figure 8). On the other hand, the runtime of the optimal algorithm significantly increases in environments with **no** common variables, much more even than the environment with third common variables (Figure 9),[4] and in fact we could not run it with more than seven candidates. In Figure 10 we can see the same tendency as in the other environments that the quality of the heuristic is better than the baseline algorithms and close to the optimal. We have obtained similar results in other conditions of costs and time horizons.

8. SUMMARY AND FUTURE WORK

In this paper we presented the problem of finding the best time to determine the winning candidate in an uncertain environment where information about the candidates arrives dynamically. We proposed a model to represent the arrival of dynamic information and its influence on the utilities of the candidates. We presented an optimal algorithm and a framework of heuristic algorithms to find the best time and candidate. We analyzed the algorithmic framework to thoroughly understand the source of its approximation. We empirically showed that although the simplest heuristic algorithm does not guarantee the optimal time and winner, it presents high

[3] At the end of Section 4 we showed that the complexity of the optimal algorithm is exponential even in case of all common variables. However, here we run experiments where the locations of the variables are the same in all candidate trees. That fact explains the non-exponential runtime.

[4] All the results are statistically significant.

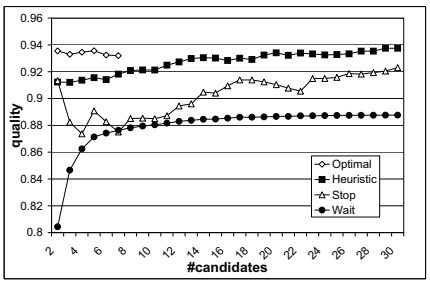

Figure 10: No common variables: Quality of the decision over the number of candidates.

quality results close to the optimum. On the other hand, it is much faster than the optimal algorithm in terms of runtime.

In future we will study the performance of heuristic algorithms that use other methods for computing the probability of winning, such as variable elimination. We will also consider representations in which candidates interact not only through common variables but also through dependencies between variables. In addition, we plan to extend the problem to multi-agent systems, where a set of agents should make a joint decision over multiple candidates. In such a situation, each agent will receive different information, and therefore they will differently estimate the probabilities of future information and the expected value of different candidates.

9. REFERENCES

[1] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121:49–107, 2000.

[2] T. Ferguson. Who solved the secretary problem? *Statistical Science*, 4:282–289, 1989.

[3] J. Figueira, S. Greco, and M. Ehrgott. *Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer Verlag, Boston, Dordrecht, London, 2005.

[4] L. Finkelstein and S. Markovitch. Optimal schedules for monitoring anytime algorithms. *Artificial Intelligence*, 126:63–108, 2001.

[5] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research (JAIR)*, 19:399–468, 2003.

[6] E. A. Hansen and S. Zilberstein. Monitoring and control of anytime algorithms: a dynamic programming approach. *Artificial Intelligence*, 126(1-2):139–157, 2001.

[7] E. Horvitz. Principles and applications of continual computation. *Artificial Intelligence*, 126:126–1, 2001.

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

[9] V.-N. Huynh, Y. Nakamori, T. B. Ho, and T. Murai. Multiple-attribute decision making under uncertainty: the evidential reasoning approach revisited. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 36(4):804–822, 2006.

[10] A. Kangas. The risk of decision making with incomplete criteria weight information. *Canadian Journal of Forest Research*, 36:195–205, 2006.

[11] R. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Cambridge University Press, Cambridge, 1993.

[12] R. Lahdelma and P. Salminen. Pseudo-criteria versus linear utility function in stochastic multi-criteria acceptability analysis. *European Journal of Operational Research*, 141:454–469, 2002.

[13] R. Lahdelma and P. Salminen. Stochastic multicriteria acceptability analysis using the data envelopment model. *European Journal of Operational Research*, 170(1):241–252, April 2006.

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

[15] J. B. Yang and D. L. Xu. On the evidential reasoning algorithm for multiattribute decision analysis under uncertainty. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 32(3):289–304, May 2002.

[16] K. Yoon and C. Hwang. *Multiple Attribute Decision Making: An Introduction*. Sage Publications, 1995.

[17] W. Zhang. Iterative state-space reduction for flexible computation. *Artificial Intelligence*, 126(1-2):109–138, 2001.

Appendix: Proof of Theorem 1

The proof uses the concept of terminal assignments, which result in stopping immediately. Only terminal assignments actually contribute to the expected utility of the policy.

DEFINITION 14 (TERMINAL). *Let π be a policy and π_i a local policy. A global assignment σ^t is* terminal *for π if $\pi(\sigma^t) =$ stop. Similarly a local sequence σ_i^t is* terminal *for π_i if $\pi_i(\sigma_i^t) =$ stop. The set of terminal assignments (resp. sequences) for π (resp. π_i) is denoted $T(\pi)$ (resp. $T(\pi_i)$).*

Proof of Theorem 1: On the one hand, we have

$$\begin{array}{l}\mathcal{GN}(\pi) = \\ \sum_t \sum_{\sigma^t \in T(\pi)} (\max U(c_i|\sigma^t) - \mathcal{CST}(\sigma^t_{\arg\max_{U(c_i|\sigma^t)}}))P(\sigma^t) = \\ \sum_t \sum_{\sigma^t \in T(\pi)} \sum_i (U(c_i|\sigma^t) - \mathcal{CST}(\sigma_i^t))[c_i \text{ wins } |\sigma^t]P(\sigma^t)\end{array}$$

where $[.]$ denotes the indicator function. Here we have used the fact that c_i wins if and only if $c_i = \arg\max_{U(c_i|\sigma^t)}$, so $\max U(c_i|\sigma^t) = \sum_i U(c_i|\sigma^t)[c_i \text{ wins } |\sigma^t]$. On the other hand, we have

$$\begin{array}{l}\mathcal{GN}_i(\pi_i) = \\ \sum_t \sum_{\sigma_i^t \in T(\pi_i)} (U(c_i|\sigma_i^t) - \mathcal{CST}\sigma_i^t)P(c_i \text{ wins } |\sigma_i^t)P(\sigma_i^t) = \\ \quad \sum_t \sum_{\sigma_i^t \in T(\pi_i)} \sum_{\sigma^t \in C(\sigma_i^t)} \begin{array}{l}(U(c_i|\sigma_i^t) - \mathcal{CST}(\sigma_i^t)) \\ [c_i \text{ wins } |\sigma^t]P(\sigma^t)\end{array} = \\ \sum_t \sum_{\sigma^t \in T(\pi)} (U(c_i|\sigma^t) - \mathcal{CST}(\sigma_i^t))[c_i \text{ wins } |\sigma^t]P(\sigma^t)\end{array}$$

The second line holds because the probability c_i wins for a given local sequence σ_i^t is the sum, over all possible completions of σ_i^t into a global assignment σ^t, of the probability c_i wins for σ^t. Those global assignment are precisely those whose projection is σ_i^t.

the probability that c_i wins is precisely the sum of the probabilities of global assignments such that it wins. The third line uses the fact that for compatible policies, a global assignment σ^t consistent with σ_i^t is in $T(\pi)$ if and only if σ_i^t is in $T(\pi_i)$. It also uses the fact that the utility of c_i is determined only by ct_i, so $U(c_i|\sigma_i^t) = U(c_i|\sigma^t)$. From the last line, we have

$$\begin{array}{l}\sum_i \mathcal{GN}_\backslash(\pi_i) = \\ \sum_i \sum_t \sum_{\sigma^t \in T(\pi)} (U(c_i|\sigma^t) - \mathcal{CST}(\sigma_i^t))[c_i \text{ wins } |\sigma^t]P(\sigma^t) = \\ \sum_t \sum_{\sigma^t \in T(\pi)} \sum_i (U(c_i|\sigma^t) - \mathcal{CST}(\sigma_i^t))[c_i \text{ wins } |\sigma^t]P(\sigma^t) = \\ \mathcal{GN}(\pi_i) \quad \blacksquare\end{array}$$