Event-Based Decompositions for Reasoning about External Change in Planners

Jim Blythe
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
jblythe@cs.cmu.edu

Abstract

An increasing number of planners can handle uncertainty in the domain or in action outcomes. However, less work has addressed building plans when the planner's world can change independently of the planning agent in an uncertain manner. In this paper, I model this change with external events that concisely represent some aspects of structure in the planner's domain. This event model is given a formal semantics in terms of a Markov chain, but probabilistic computations from this chain would be intractable in real-world domains. I describe a technique, based on a reachability analysis of a graph built from the events, that allows abstractions of the Markov chain to be built to answer specific queries efficiently. I prove that the technique is correct. I have implemented a planner that uses this technique, and I show an example from a large planning domain.

Introduction[1]

An increasing number of subgoaling planners can handle uncertainty in the domain or in action outcomes, including (Kushmerick, Hanks, & Weld 1994) and (Haddawy, Doan, & Goodwin 1995). However relatively little work has addressed building plans when the world can change independently of the planning agent in an uncertain manner, although (Blythe 1994) and (Hanks, Madigan, & Gavrin 1995) are examples of work in this area. In this paper I describe a method for representing external change by events that can capture some aspects of structure in the possible changes in the planner's domain. The events have a similar form to STRIPS actions, allowing a planner to create subgoals whose achievement will affect the probability of occurrence of events beyond its direct control. This is discussed further in (Blythe 1994), while the main focus of this paper is to provide a formal description of the model and demonstrate a technique that allows probabilistic expressions to be computed efficiently.

The event model is given a formal semantics in terms of a Markov chain with the same state space as the planning domain, called the "full model". However, probabilistic computations made from this chain would be very inefficient for real-world domains. In calculating the probability of a particular plan succeeding, one typically only needs to know the values of small subsets of the literals that describe the domain. This is because each individual action usually depends on only a small number of literals. It is therefore very inefficient to compute the probability directly from a model that considers the complete state of the domains and so specifies all the fluents. A crucial issue is then to find ways to decompose the full model into small, abstract models that can be used to answer queries about these subsets of the domain. This problem has been discussed by several groups of researchers, including (Boutilier, Dean, & Hanks 1995) and (Dean & Lin 1995).

In this paper I describe an algorithm that takes a subset of the domain literals and produces an abstraction of the full model that can provably be used to compute the same probabilities as the full model for queries involving the subset of literals. I introduce the *event graph*, which is designed to represent the dependencies among events. Given a subset of literals of interest in the domain, I show how a reachability analysis of the event graph can be used to build abstractions of the full model that will answer queries about the subset efficiently. I sketch a proof, contained in full in (Blythe 1996), that computations in the abstract chains will produce the same answers as in the full model. The event graph can be computed efficiently from the domain description before the planning problem is encountered. I show an example of an implemented planner that uses this technique in a large planning domain, where computation with the full Markov chain that models the events is intractable.

The next section provides a formal model of external events for a planner in terms of an algorithm that uses them to build a Markov chain describing change over time in the domain. Then I describe the event graph and the technique for building abstractions of the Markov chain useful for answering specific queries. A proof that this technique is sound is sketched in the appendix to this paper. After describing the event graph I demonstrate an implementation of the technique in a large planning domain.

[1]

A formal model for actions and events

In this section I give a formal description of a representation for actions and events, used to motivate and describe the event graph in the next section. The model is based on (Kushmerick, Hanks, & Weld 1994). The planner in which the event graph approach has been implemented uses a more powerful representation than is described here, but this model is adequate to describe event graphs, and the proof that the technique for building abstractions is correct holds in the more complex case.

A planning domain is defined by a set of literals L and a set of actions $\mathcal{A}$. Each literal $l \in L$ is a binary-valued proposition, and the state space Ω is the cross product of L. Expressions can be formed using the literals and the logical connectives $\wedge$, $\vee$ and $\neg$ to denote sets of states. For example, if $L = \{\mathsf{sunny}, \mathsf{cold}\}$, then Ω has four states and the expression $\neg\mathsf{sunny}$ denotes two states. I denote the set of literals that are true in state $\omega \in \Omega$ by $\mathrm{lit}(\omega)$.

Actions are defined in terms of their effects on literals, forming an extension to the action definition in STRIPS. An action a in STRIPS is specified by a precondition written $\mathrm{pre}(a)$, which is a logical expression over the literals, and two lists of literals called the add and delete lists, written $\mathrm{add}(a)$ and $\mathrm{del}(a)$ respectively. If the action is performed in the state ω, in which the precondition is satisfied, the resulting state ω' is specified by its literals:

$$\mathrm{lit}(\omega') = (\mathrm{lit}(\omega) - \mathrm{del}(a)) \cup \mathrm{add}(a)$$

This state is written as $\mathrm{result}(\omega, \mathrm{add}(a), \mathrm{del}(a))$.

An action in a probabilistic planner maps a state in Ω to a probability distribution over Ω. First we generalise the add and delete lists to a finite probability distribution over add and delete lists, called an *effect distribution*. This is a list of triples, $(p_i, \mathrm{add}_i, \mathrm{del}_i)$, with $1 \leq i \leq n$ for some finite n, where the add_i and del_i are add and delete lists of literals as above, $p_i > 0$ for $1 \leq i \leq n$ and $\sum_{i=1}^{n} p_i = 1$. An action a is specified by a binary tree whose leaves correspond to effect distributions and whose internal nodes correspond to literals. When the action is applied in a state ω, the tree is first traversed to find the appropriate effect distribution. At each internal node of the tree, the "true" branch is taken if the associated literal is in $\mathrm{lit}(\omega)$, and otherwise the "false" branch is taken. The effect distribution at the end of this path, written $\mathrm{eff}(a, \omega)$, gives a conditional probability distribution $P(.|\omega, a)$ over Ω defined by

$$\begin{aligned} P(\omega'|\omega, a) &= p_i, \quad \text{if } \mathrm{lit}(\omega') = \mathrm{result}(\omega, \mathrm{add}_i, \mathrm{del}_i) \\ & \qquad\quad \text{for some effect in } \mathrm{eff}(a, \omega) \\ P(\omega'|\omega, a) &= 0 \qquad \text{otherwise} \end{aligned}$$

Syntactically, external events are represented identically to actions. However, while actions can be selected by the planner to control its environment, events specify the way the environment will change in the absence of control. A literal may be contained in the effects of at most one event, so that if an event can change a literal, that event specifies all the ways the literal can change. The semantics of the set E of domain events are given in terms of a Markov process M_f that models the evolution of the state when no actions take place. The state space of M_f is the state space Ω of the planning domain. The set of transitions from the state $\omega \in M_f$ are effectively determined by allowing all applicable events to take place in parallel independently of each other. Since each event $e \in E$ can have any effect triple in the distribution $\mathrm{eff}(e, \omega) = \{(p_{j(e,\omega)}, \mathrm{add}_{j(e,\omega)}, \mathrm{del}_{j(e,\omega)}) \mid 1 \leq j(e, \omega) \leq m(e, \omega)\}$, the probability of the transition in M_f from ω to the state given by

$$(\mathrm{lit}(\omega) - \bigcup_{e \in E} \mathrm{del}_{j(e,\omega)}) \cup \bigcup_{e \in E} \mathrm{add}_{j(e,\omega)}$$

is $\prod_{e \in E} p_{j(e,\omega)}$. I refer to M_f as the *full model* of change in the domain.

I use a highly simplified domain concerned with cleaning up spilled oil to illustrate events, the full model and the event graph. Oil spilled from a tanker at sea is initially concentrated around the tanker, but after some amount of time becomes spread out, described by the literal spread. The weather at the scene can be fair or foul, described by the literal fair and a local fishing boat may be at sea or in the harbour, described by the literal sea. The three events in Figure 1 describe how these aspects of the domain can change over time, independently of any actions taken by the planner. These events can be interpreted by the full model of the domain, the Markov chain shown in Figure 2. For example, the probability associated with the one-step transition from state $\omega 1$, $(\mathsf{fair} \wedge \neg\mathsf{spread} \wedge \mathsf{sea})$, to $\omega 7$, $(\mathsf{fair} \wedge \mathsf{spread} \wedge \neg\mathsf{sea})$, is $0.9 \times 0.1 \times 0.2 = 0.018$, where the value 0.9 is the probability that the weather remains fair, from the event "weather-change", the value 0.1 is the probability that the oil changes from not spread to spread, given that the weather is fair, and the value 0.2 is the probability that the boat leaves the sea for the harbour, also given that the weather is fair.

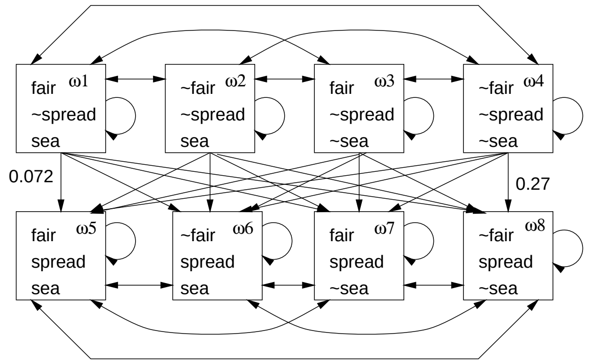

Figure 2: The full Markov chain describing the environment, showing all transitions and some sample probabilities.

Since the semantics of events are defined in terms of a Markov chain, one may ask why use an event-based representation of external change in a planning domain rather than specify a Markov chain directly? The events provide

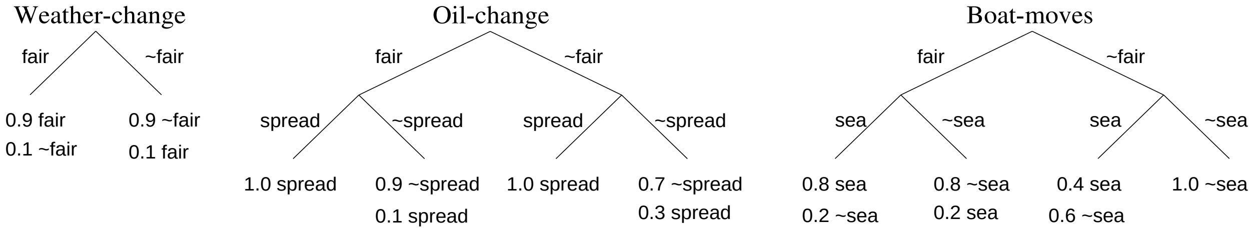

Figure 1: Three events from the simplified oil spill scenario, describing how the weather can change, how the spilled oil may spread over time and how a nearby fishing boat may move. An abbreviated form for effects is used, where a positive literal stands for adding the literal and a negated literal stands for deleting it. Thus, the effect distribution at the leaf reached when fair is true in the weather-change event is $((0.9, (\mathsf{fair}), ()),\ (0.1, (), (\mathsf{fair})))$

an abstraction of the domain's structure of change that can be useful to subgoaling planners. Events can be used to predict change with only a partial specification of a state, and the preconditions of events can lead to planning subgoals whose achievement will reduce the probability of events interfering negatively with a plan, or increase the probability of events having useful effects.

Decompositions based on the event graph

When plans are constructed in domains where operators take a significant time to execute, the probability of plan success depends on the way the world can change while the plan is executed. For example consider the oil-spill domain with an initial state of ($\mathsf{fair} \wedge \neg\mathsf{spread} \wedge \mathsf{sea}$), and a goal to remove the threat of the oil. Suppose a candidate plan is to use a boat to transport some chemical dispersant to the oil and apply it, which requires that the weather is fair and the oil has not spread. If it will take the boat three hours to arrive on the scene, we need to know the probability distributions governing the weather and the oil at that time in order to compute the plan's probability of success, even though the initial state is known.

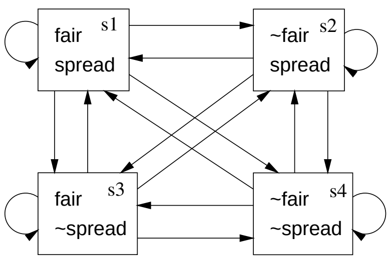

Figure 3: A reduced model of the domain, abstracting away the boat's position.

One way to compute this probability is to find the three-step transition probabilities in the full model of the domain in Figure 2. It has eight states, two of which ($\omega 1$ and $\omega 3$) will lead to the plan succeeding if they hold when the barge arrives, since we don't care about the fishing boat. Figure 3 shows a reduced model that ignores the boat. We can find the probability of success by computing with the reduced model, since the boat has no effect on the way the weather and the oil change. The computational savings are small in this example, but grow exponentially with the number of boats and other objects in the planning domain with associated events, since the size of the full model's state space will grow exponentially while the reduced model will not be affected.

A naive way to use reduced models might be to create a number of minimal models such as the one in Figure 3 and use them to answer queries about the future state of the world. However, care must be taken about joint queries if the events are not independent. For example, suppose the chemical dispersant can only be used when the boat is not at sea, so we need to know the probability of $\neg\mathsf{sea} \wedge \mathsf{fair} \wedge \neg\mathsf{spread}$. We cannot use two minimal models, one for sea and one for spread, because these literals do not change independently but both depend on the weather. We must use the full model to get the correct answer.

The event graph

Given a set of literals of interest, we would like to build a submodel that is small enough to answer queries about these literals efficiently, but includes all the literals necessary to ensure the result is correct. Event graphs enable us to do this by capturing the dependencies between the events in the domain.

The *event graph* for a set of events E is a directed graph whose nodes are either events or literals in the domain. There is an arc from each event $e \in E$ to every literal in its effects, and from every literal in any of the branches of e to e (so the graph may have cycles). Figure 4 shows the event graph for the example domain. A simple algorithm that adds arcs for each event in turn can produce the event graph in time linear in the number of events, the maximum number of branches in an event and the maximum number of effects in an effect distribution.

Given a query, such as $\neg\mathsf{spread} \wedge \mathsf{fair}$, that contains a set of literals, V, a set of Markov chains is created in two steps. First, create the subgraph of the event graph, restricted to the nodes in V and all their ancestors in the graph. Second, create one Markov chain for each component of the resulting graph. The state space for each chain is the cross product of the literals that appear in the corresponding component, and

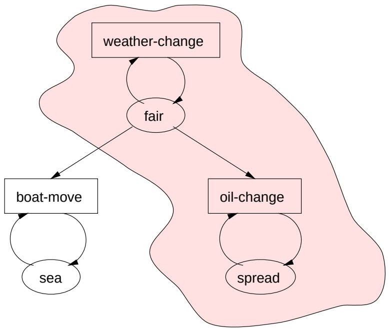

Figure 4: The event graph for the example domain, with boxes for events and ovals for literals. Inside the shaded area is the subgraph for a query on the literal spread

the transitions are formed from the events in the component in the same way as the full-size Markov chain is created from all the events, described above. The probability of the query over literals in V after n stages can then be computed by multiplying together the probability from each chain of the projection of V on that chain after n stages. I refer to this set of Markov chains as the *submodel of the full model induced by* V. For example, the reduced model in Figure 3 is the submodel of the full model for the domain induced by {spread}.

Intuitively, no literal that is not contained in the subgraph of the event graph created for V can affect the way any literal in V changes over time, since if it could affect any event that could affect any of the literals in V, either directly or indirectly, it would be contained in the subgraph by its definition. Similarly the literals in different components of the subgraph change independently of each other, since the value of one can never affect the way the value of the other changes. This intuition underlies the proof of the following theorem. A sketch of the proof is given in the appendix.

Theorem: *All queries about the probability of a logical expression over a set of literals $V \subset L$, after some fixed number of stages n given an initial state probability distribution over the full state space Ω will have the same value in the submodel induced by V as in the full model of the domain.*

Using this theorem we can compute the answers to queries about small subsets of the domain variables without consulting the full model of the domain. In fact, the full model is typically never constructed.

An example

Weaver is an implemented planner that uses event graphs to reason about change in the environment. I present an example of Weaver reasoning about a plan constructed in the oil-spill domain, a large planning domain developed by SRI for the US Coast Guard[2] (Desimone & Agosta 1994).

Weaver is a backchaining planner built on top of Prodigy (Veloso *et al.* 1995) that includes a representation for actions and events similar to that described in the last section. The algorithm for Weaver is presented in (Blythe 1996) but for this section it is sufficient to know that Weaver decides whether a plan produced by Prodigy exceeds some threshold probability of success, taking events into account. Prodigy represents actions and events in schemas that contain predicates with typed variables. Thus it is not possible to build the full event graph before the planning problem is specified, since the number of literals in the domain depends on the number of objects of various types. Weaver therefore calculates the event graph in two stages. Before a particular set of objects is specified, it creates an *event graph schema* whose nodes are either event schemas or predicates whose arguments may be variables. At planning time, given a set of literals of interest, Weaver builds the relevant subgraph of the event graph by instantiating the event graph schema in all ways that can match the literals, recursively instantiating all ancestor literals of interest. In this way we avoid building the entire event graph, which would grow linearly in the number of literals in the planning problem.

The oil-spill domain was developed to help allocate resources for disaster aid in the event of an accident involving an oil tanker, by aiding a human simulate spill scenarios with various available resources. In the context of a spill from a tanker, goals are set to control the spread of oil, to clean it up and to protect sensitive areas of coastline. In its translation into Prodigy, the domain has 57 operators and inference rules. An example scenario constructed at SRI for the domain and used in this section contains 302 different objects, including 30 vessels and 183 physical locations. This paper is too short to describe the domain in much detail, but I will briefly describe a plan constructed by Prodigy for the example scenario.

The top-level goal in this domain is usually *respond-to-spill*, which is satisfied when (1) most of the oil spilled in the water has been removed and (2) the vessel leaking oil has been stabilized to reduce the rate of discharge[3]. The plan shown in Figure 5 contains steps to remove the oil on the water with a chemical dispersant and to stabilize the leaking vessel by pumping the remaining oil into a barge. This requires moving the equipment to the required locations by sea.

Each vessel in the domain places a limit on the sea conditions under which it may be used. In order for the plan to succeed, the sea conditions at the points where each vessel is used must not exceed this worst level at any time. The sea conditions of each sea sector in the domain vary over time, according to an event schema "sea-change" that takes into account the current sea conditions of the sector and the weather. The planning scenario includes 37 distinct sea sectors, so if this were the only relevant factor of change, and if the sea condition and weather took on only two values, the full model of the domain would require 2^{38} states. This is

[2] I am grateful to Roberto Desimone and John Mark Agosta of SRI for making this domain available.

[3] In general we also require steps to be taken to prevent damage to any sensitive shoreline areas, but this is omitted here for brevity.

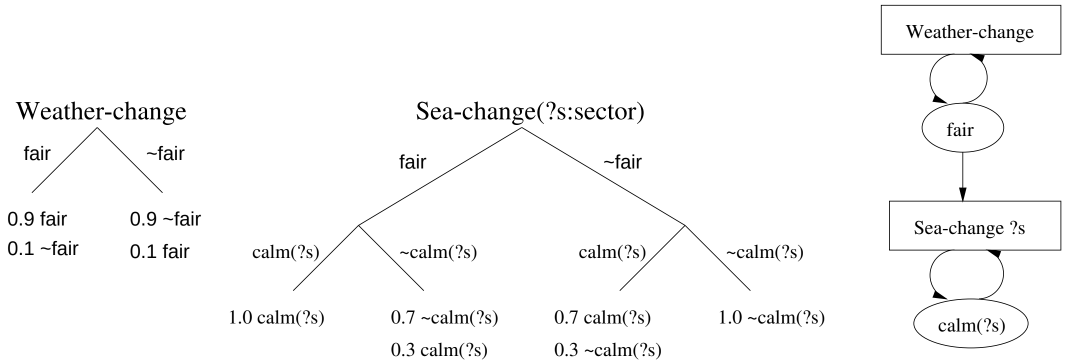

Figure 6: Two events in the Oil-spill domain and the corresponding event graph schema.

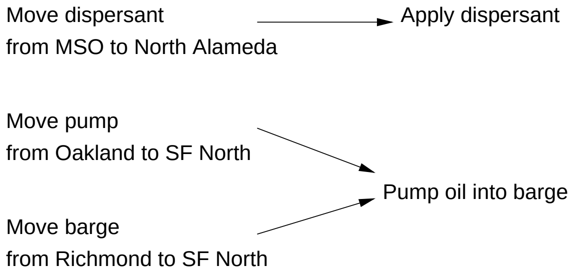

Figure 5: Operator tree for a plan to respond to an oil spill. In this plan, two operators can be applied in parallel unless one is a descendant of the other in the tree.

clearly intractable even without considering other relevant events, for example modelling the traffic conditions, for transportation of other equipment by trucks, and the spread of the oil itself.

Weaver computes the submodel of the domain induced by the literals used in the plan. Computing plan success requires us to keep track of the sea state in 5 sea sectors. Because they are all dependent on the weather, the event graph technique leads to a reduced Markov model consisting of 64 states which is adequate to compute the probability of success of this plan, assuming the change in sea condition is the only uncertain event. Weaver builds this model by instantiating the event graph schema in Figure 6 for all known sea-sectors, then finding the relevant subgraph for the plan, containing the 5 sea sectors[4]. The instantiated sea-change events are all connected in the event graph through the literal `fair`, so there is only one component to the event subgraph.

Once the submodel induced by the literals in the plan is created, it is incorporated into a Bayesian belief net that Weaver builds to reason about the probability of plan success, described in (Blythe 1994). Nodes in the belief net represent steps in the plan, and the values of fluents at time points that are relevant to the plan. In addition, nodes are added for each component of the submodel, at each time point that a literal from the component is relevant to plan success. The state space of the node is the cross product of the literals in the component, and so has 64 states in this example. Link probabilities in the belief net that represent persistence properties of the component are computed by evaluating the Markov chain. This hybrid model allows Weaver to take account of both uncertain operator outcomes and exogenous events efficiently.

The belief net created in the example plan is shown in Figure 7. In this case, the Markov chains are only run for one step since the actions are set to have unit duration. Since there is just one component to the reduced model in this example, there is no computational advantage visible in the Bayes net diagram to the reduced model. In this case the advantage comes from the smaller state space of the Markov chain. In the next section I briefly mention some ways to exploit a finer-grained independence among the events than the current implementation.

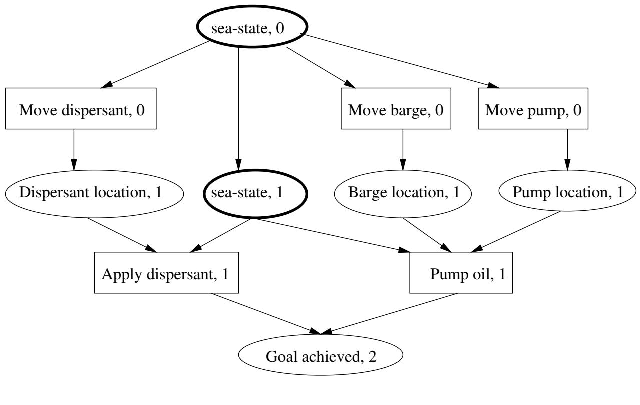

Figure 7: The belief net created to model the example plan. Steps are represented with squares and fluent values with ellipses. The heavy ellipses represent nodes that are computed from the reduced markov model. The numbers represent the time at which the step is applied, or the fluent is queried.

[4]In fact the instantiation process is merged with finding the subgraph for efficiency, but the result is the same.

Discussion

I have described a model for external change in the environment and have shown an algorithm that, given a subset of the literals in the domain, produces a reduced Markov chain that can be used to compute the probabilities of changes in those literals efficiently. The appendix sketches a proof that the technique is correct. I have demonstrated a planner using the technique in a domain where reasoning about change in the full environment would be intractable.

Decompositions of Markov processes have been discussed in several places in the context of probabilistic planning. For example, (Boutilier, Dean, & Hanks 1995) and (Dean & Lin 1995) consider decompositions of Markov decision processes. Boutilier and Dearden (Boutilier & Dearden 1994) use abstractions based on subsets of the domain literals to produce policies that come within a given bound of optimal for a Markov decision process (MDP). Dean and Lin (Dean & Lin 1995) partition the state space of an MDP and solve each partition using a different abstraction, combining the results to provide a solution to the original problem.

This paper describes a technique for exact decompositions of Markov chains, which has no computational overhead in re-combining the results from the abstract chains but is not as broadly applicable as the techniques mentioned above. The event graph can be used to find abstractions for the other methods, even when the graph does not split into true components but into groups connected by small cutsets. Note that the underlying planning algorithm used in the implementation described here is a search-based AI planner rather than policy iteration. The commitment made in the algorithm to a specific plan is partly responsible for the small size of the abstraction produced. This focus of attention is not achieved in standard policy iteration algorithms, although it is used in structured policy iteration (Boutilier, Dearden, & Goldszmidt 1995).

There are also many instances in the literature where the operator dependency graph is analysed. The event graph used here is similar to the operator graph used by Knoblock to create abstractions for classical planning (Knoblock 1993), as well as the problem space graph introduced by Etzioni for learning search control from action representations (Etzioni 1990), and the directed cyclic graphical representations of Spirtes (Spirtes 1995).

Although this paper has focussed on marginal independence, a commitment to a particular policy allows us to exploit conditional independence in the events to get a finer-grained abstraction. For example in the plan produced in the previous section, success is conditionally independent of the sea-state in Richmond, given the weather and the success of the step moving the barge from Richmond port to San Francisco. In Weaver this independence could be captured in the Bayes net representing the plan. This would lead the sea-state in Richmond to be modelled for a shorter time, which has two advantages. First, the shorter period leads to reduced uncertainty. Second, events may be independent over a limited time that are dependent in the long run. Suppose, for example, that at each time period the sea-state depends on the weather and the adjacent sea-sectors. Then the event graph component for one sea-sector must include all connected sea-sectors, despite the fact that over two time periods the sector is independent of sea-sectors more than two steps away in the adjacency graph. A future direction for this work is to make precise the conditions under which limited-time independence and conditional indpendence of events can be exploited.

Another direction is the extension of the results to semi-Markov networks. In order to represent the world using regular Markov networks with reasonable accuracy, all the events must take identical amounts of time, or have a stationary (exponential) probability distribution on their duration as well as their probability of initiation. More complex problems are more accurately modelled using semi-Markov networks that allow arbitrary probability distributions to model the time before a transition occurs in a state. This model is more expensive to solve than the pure Markov case, but in some cases the event graph technique may lead to a tractable model of change.

Acknowledgements

Discussions with Jaime Carbonell and the Prodigy group at CMU have been very helpful in developing these ideas. I am also grateful for Rich Goodwin's comments on an earlier draft. This research is sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. Views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of Wright Laboratory or the United States Government.

Appendix: Sketched proof of the event graph theorem

Theorem: *All queries about the probability of a logical expression over a set of literals $V \subset L$, after some fixed number of transitions n given an initial state probability distribution over the full state space Ω will have the same value in the submodel induced by V as in the full model of the domain.*

It is sufficient to show that probabilities of all conjunctions involving each literal in V or its negation will be the same in each model, since any expression can be expressed as the disjunction of such expressions, whose probabilities can be summed since they are mutually exclusive. I first show that a chain M_r built of the subgraph of the event graph containing the variables in V and their ancestors in the event graph will compute the correct value, and then showing that the same value can be computed by creating independent chains for the separate components of the subgraph.

Each full conjunctive expression corresponds to a unique state of M_r. For each such state i, let $F(i)$ denote the set of states in the full model M_f that agree with i on the literals in V. Let $p^n_{r,(i,j)}$ be the probability that the reduced chain M_r is in state j after n steps given that it began in state i, and let $p^n_{f,(x,y)}$ be the probability that the full model M_f is in state y after n steps given that it began in state x.

A situation in which M_r is in state i may correspond to any probability distribution $P(.)$ over states in $F(i)$ for M_f, where $\sum_{x \in F(i)} P(x) = 1$. Thus given states i and j in M_r, we need to show that, for all n and for all $P(.)$,

$$p^n_{r,(i,j)} = \sum_{x \in F(i)} P(x) \sum_{y \in F(j)} p^n_{f,(x,y)} \qquad (1)$$

By choosing different probability distributions it can be seen that this is equivalent to

$$p^n_{r,(i,j)} = \sum_{y \in F(j)} p^n_{f,(x,y)} \qquad \forall x \in F(i) \quad (2)$$

since we can choose $P(x) = 1$ for any $x \in F(i)$. Conversely if equation (2) is true, any linear combination given by a probability distribution will also satisfy the equality, so equation (1) is true also.

We use induction on n. The base case to prove is for $n = 1$:

Let E_r be the set of events in the subgraph of the event graph used to build M_r, and let $p(e, i, j)$ be the probability of the effect in eff(e, i) that changes the literals affected by e to match j, if such an effect exists, and let $p(e, i, j) = 0$ otherwise. By definition, $p^1_{r,(i,j)} = \prod_{e \in E_r} p(e, i, j)$

Similary for any $x \in F(i)$ and for any $y \in M_f$, $p^1_{f,(x,y)} = \prod_{e \in E} p(e, x, y)$. We can arrange this product in terms of events in E_r and the rest:

$$p^1_{f,(x,y)} = \prod_{e \in E_r} p(e, x, y) \times \prod_{e' \in E - E_r} p(e', x, y)$$

Now the events in E_r only effect the literals in M_r and are completely determined by them, by the construction of the event subgraph. So if $y \in F(j)$, $p(e, x, y) = p(e, i, j)$ and the first term is just $p^1_{r,(i,j)}$. So we have

$$\sum_{y \in F(j)} p^1_{f,(x,y)} = p^1_{r,(i,j)} \sum_{y \in F(j)} \prod_{e' \in E - E_r} p(e', x, y) \forall x \in F(i)$$

Also by the construction of the event subgraph, the events in $E - E_r$ do not alter any of the literals in M_r. So for $x \in F(i)$, if we fix the outcomes of events in E_r to lead to j, all transitions with non-zero probability in M_f must lead to a state in $F(j)$. Then under these circumstances $\sum_{y \in F(j)} \prod_{e' \in E - E_r} = 1$, and the induction base case is proved.

Now suppose the result is true for $n \leq m - 1$, so that for any $x \in F(i)$ and $z' \in F(k)$,

$$\begin{aligned} p^m_{r,(i,j)} &= \sum_{k \in M_r} p^{m-1}_{r,(i,k)} p^1_{r,(k,j)} \\ &= \sum_{k \in M_r} \Big(\sum_{z \in F(k)} p^{m-1}_{f,(x,z)} \Big) \Big(\sum_{y \in F(j)} p^1_{f,(z',y)} \Big) \end{aligned}$$

We can re-arrange the summands on the right hand side, and choose $z' = z$ separately for each $z \in F(k)$ to get

$$p^m_{r,(i,j)} = \sum_{k \in M_r} \sum_{z \in F(k)} \sum_{y \in F(j)} p^{m-1}_{f,(x,z)} p^1_{f,(z,y)}$$

since each state in M_f is in $F(k)$ for some $k \in M_r$,

$$\begin{aligned} p^m_{r,(i,j)} &= \sum_{z \in M_f} \sum_{y \in F(j)} p^{m-1}_{f,(x,z)} p^1_{f,(z,y)} \\ &= \sum_{y \in F(j)} \sum_{z \in M_f} p^{m-1}_{f,(x,z)} p^1_{f,(z,y)} \\ &= \sum_{y \in F(j)} p^m_{f,(x,y)} \end{aligned}$$

which is the desired result.

For the second stage of the proof we need to show that separate Markov chains derived from the components of the event graph's subgraph and treated independently will yield the same result as a calculation in M_r. The proof uses the same technique as in the first stage, noting that the events in each component effect different literals and are determined by different literals from all the other components, so the transition probabilities can be split in the same way. This part of the proof is omitted here but can be found in (Blythe 1996).

References

Blythe, J. 1994. Planning with external events. In de Mantaras, R. L., and Poole, D., eds., *Proc. Tenth Conference on Uncertainty in Artificial Intelligence*, 94–101. Seattle, WA: Morgan Kaufmann.

Blythe, J. 1996. Planning under uncertainty. Technical report, School of Computer Science, Carnegie Mellon University.

Boutilier, C., and Dearden, R. 1994. Using abstractions for decision-theoretic planning with time constraints. In *Proc. Twelfth National Conference on Artificial Intelligence*, 1016–1022. AAAI Press.

Boutilier, C.; Dean, T.; and Hanks, S. 1995. Planning under uncertainty: structural assumptions and computational leverage. In *Proc. European Workshop on Planning*. Assissi, Italy: IOS Press.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Proc. 14th International Joint Conference on Artificial Intelligence*, 1104–1111. Montréal, Quebec: Morgan Kaufmann.

Dean, T., and Lin, S.-H. 1995. Decomposition techniques for planning in stochastic domains. In *Proc. 14th International Joint Conference on Artificial Intelligence*, 1121 – 1127. Montréal, Quebec: Morgan Kaufmann.

Desimone, R. V., and Agosta, J. M. 1994. Oil spill response simulation: the application of artificial intelligence planning technology. In *Simulation Multiconference*.

Etzioni, O. 1990. *A Structural Theory of Explanation-Based Learning*. Ph.D. Dissertation, School of Computer

Science, Carnegie Mellon University. Available as technical report CMU-CS-90-185.

Haddawy, P.; Doan, A.; and Goodwin, R. 1995. Efficient decision-theoretic planning: Techniques and empirical analysis. In Besnard, P., and Hanks, S., eds., *Proc. Eleventh Conference on Uncertainty in Artificial Intelligence*, 229–326. Montreal, Quebec: Morgan Kaufmann.

Hanks, S.; Madigan, D.; and Gavrin, J. 1995. Probabilistic temporal reasoning with endogenous change. In Besnard, P., and Hanks, S., eds., *Proc. Eleventh Conference on Uncertainty in Artificial Intelligence*, 245–254. Montreal, Quebec: Morgan Kaufmann.

Knoblock, C. A. 1993. *Generating Abstraction Hierarchies: An Automated Approach to Reducing Search in Planning*. Boston: Kluwer.

Kushmerick, N.; Hanks, S.; and Weld, D. 1994. An algorithm for probabilistic least-commitment planning. In *Proc. Twelfth National Conference on Artificial Intelligence*, 1073–1078. AAAI Press.

Spirtes, P. 1995. Directed cyclic graphical representations of feedback models. In Besnard, P., and Hanks, S., eds., *Proc. Eleventh Conference on Uncertainty in Artificial Intelligence*, 491–498. Montreal, Quebec: Morgan Kaufmann.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The prodigy architecture. *Journal of Experimental and Theoretical AI* 7:81–120.