Heuristics with Choice Variables: Bridging the Gap between Planning and Graphical Models

Emil Keyder
INRIA, France

Miquel Ramírez
Universitat Pompeu Fabra
08018 Barcelona, Spain

Héctor Geffner
ICREA & Universitat Pompeu Fabra
08018 Barcelona, Spain

Abstract

Motivated by the goal of applying inference techniques used in graphical models to planning, we introduce a novel heuristic based on the idea of choice variables, implicit multivalued variables to which no plan can assign more than one value. We adapt the recursive conditioning algorithm for calculating the probability of evidence in Bayesian networks to efficiently compute the values of this heuristic by considering problem structure and reasoning by cases about different assignments to these variables. The resulting algorithm is exponential in the treewidth of the graph describing the causal relationships between the choice variables of the problem. We present some examples of the computation of the heuristic, and discuss the issues faced in applying it to existing planning benchmarks, a goal which for now remains elusive.

Introduction

Many recently proposed techniques in planning make use of *invariants* such as landmarks and mutexes. Here, we consider a new type of invariant in planning: implicit multivalued variables to which a value can be assigned *at most once* by a plan. These variables, which we call *choice variables*, are different from standard multivalued variables in that they do not represent properties of the current state that may change in future states, but rather commitments made by the planner to solve the problem while accepting the constraints that a choice imposes upon the space of possible plans. Indeed, they are more similar in spirit to the variables used in graphical models such as constraint satisfaction problems and Bayesian networks, in which a single value that is consistent with the rest of the solution to the problem, or a single most likely instantiation, must be chosen for each variable.

Choice variables can be used as a tool for improving the quality of planning heuristics, by forcing them to respect features of a problem that otherwise might not be present in a relaxation. Consider a problem in which an agent must travel from an initial location to *one of* a set of markets and buy a set of items. In an optimal plan, the agent moves from its initial location to a market which minimizes the sum of the movement cost and the total cost of the items at that market. The delete relaxation of this problem, however, throws away information that restricts the agent to traveling to a *single* market. When movement costs are low, the optimal delete relaxation plan is then to move to all markets that carry some required item at the lowest price and buy each at a different market. The knowledge that *the set of markets constitutes a choice variable*, only one of whose values can be chosen, allows us to improve the value of a delete-relaxation heuristic by computing its value in several different versions of the same problem, in each of which the actions allowing the agent to move to all but one of the markets are *excluded*. Taking the *minimum* among these estimations then gives a strictly more informative estimate for the cost of the problem.

Since the number of possible assignments to a set of variables grows exponentially in the size of the set, the approach of enumerating all possible assignments becomes unfeasible as the number of variables increases. However, this effect can be mitigated when the interactions between choice variables are in some way *structured*. Consider a variant of the problem above in which there are several different types of market, so the agent must first go to one of a set of markets which sells food, then to one of a set of markets which sells drinks, etc. As long as the sequence in which the different types of markets must be visited is fixed, the choice of the nth market to visit is dependent only on the market chosen at step $n-1$. In effect, the graph describing the interactions between the choice variables of the problem forms a *chain*, a structure that various algorithms for graphical models can take advantage of to reduce the time complexity to linear in the number of variables. Here we adapt the recursive conditioning algorithm (Darwiche 2001) to the planning setting to take advantage of such problem structure.

The resulting heuristic is related to *factored* approaches to planning, which decompose a planning problem into a set of *factors* and attempt to find plans for each that can be interleaved with the plans for the others in order to obtain a global solution (Amir and Engelhardt 2003; Fabre et al. 2010). Such factors may share with each other either fluents or actions; our approach is similar to the former, with the shared fluents being the different values of choice variables. The number of times that the value of a variable shared between two factors must be changed has been investigated in this setting as a complexity parameter of the planning problem (Brafman and Domshlak 2006); our heuristic can be seen as having *a priori* knowledge that this parameter

is limited to 1 for the set of choice variables. The behaviour of our heuristic when this assumption about the problem is not satisfied and constitutes a relaxation of the problem remains to be investigated. While factored planning methods attempt to find explicit plans for the global problem, we instead obtain heuristic estimates of the cost of solving each subproblem and combine these costs in order to obtain a heuristic for the global problem.

In this work, we focus on how to use choice variables to efficiently compute better heuristic estimates for planning problems, and assume that the choice variables themselves are given to the planner.

Background

We use the STRIPS formalization of planning, in which problem states and operators are defined in terms of a set of propositional variables, or fluents. Formally, a planning problem is a 4-tuple $\Pi = \langle F, O, I, G\rangle$, where F is such a set of variables, O is the set of operators, with each operator $o \in O$ given by a 3-tuple of sets of fluents $\langle \mathit{Pre}(o), \mathit{Add}(o), \mathit{Del}(o)\rangle$ and a cost $\mathit{cost}(o)$, $I \subseteq F$ is the initial state, and $G \subseteq F$ describes the set of goal states. A *state* $s \subseteq F$ is described by the set of fluents that are true in that state. An operator o is *applicable* in s when $\mathit{Pre}(o) \subseteq s$, and the result of applying it is $s[o] = (s \setminus \mathit{Del}(o)) \cup \mathit{Add}(o)$. The set of goal states is $\{s \mid G \subseteq s\}$, and a plan is a sequence of operators $\pi = o_1, \ldots, o_n$ such that applying it in I results in a goal state. The *cost* of π is $\sum_{i=1}^{n} \mathit{cost}(o_i)$, with an *optimal* plan π^* being a plan with minimal cost. The perfect heuristic $h^*(s)$ is the cost of an optimal plan for a state s, and an *admissible* heuristic h is one for which $h(s) \leq h^*(s)$ for all s.

The Choice Variables Heuristic

Choice variables in STRIPS can be defined as sets of fluents such that each operator adds at most one of them, and such that no plan applicable in the initial state contains more than one operator that adds some fluent in the set:

Definition 1 (Choice variable in STRIPS) *A choice variable $C_i = \{d_1, \ldots, d_n\}$ consists of a set of fluents such that $\max_{o \in O} |\mathit{Add}(o) \cap C_i| = 1$, and for any sequence of actions π applicable in I, $|\pi_{C_i}| \leq 1$, where $\pi_{C_i} = \{o \in \pi \mid \mathit{Add}(o) \cap C_i \neq \emptyset\}$.*

Given a set of choice variables $C = \{C_1, \ldots, C_n\}$, an assignment to C is a set $\mathbf{v}$ such that $|\mathbf{v} \cap C_i| \leq 1$ for $i = 1, \ldots, n$. Note that an assignment does not necessarily specify a value for each choice variable. For an assignment $\mathbf{v}$, a problem $\Pi^{\mathbf{v}}$ that *respects* $\mathbf{v}$ can be obtained by removing from the problem all operators that add a value that is inconsistent with $\mathbf{v}$:

Definition 2 ($\Pi^{\mathbf{v}}$) *Given a problem $\Pi = \langle F, I, O, G\rangle$, a set of choice variables C, and an assignment $\mathbf{v}$ to C, let $\Pi^{\mathbf{v}} = \langle F, I, O^{\mathbf{v}}, G\rangle$, where $O^{\mathbf{v}}$ is given by*

$$O^{\mathbf{v}} = \bigcap_{C_i \in C} \{o \in O \mid (C_i \setminus \mathbf{v}) \cap \mathit{Add}(o) = \emptyset\} \qquad (1)$$

In other words, $O^{\mathbf{v}}$ consists of the set of operators that either add no fluent in C_i or add the value d such that $\mathbf{v} \cap C_i = \{d\}$. When $\mathbf{v} \cap C_i = \emptyset$, no operator adding any value of C_i is included in $O^{\mathbf{v}}$. Given a *base heuristic* h, we can define the *choice variable heuristic* h^c in terms of h and the problems $\Pi^{\mathbf{v}}$ for $\mathbf{v} \in \phi(C)$, where $\phi(C)$ denotes the set of possible assignments to C:

$$h^c = \min_{\mathbf{v} \in \phi(C)} h(\Pi^{\mathbf{v}}) \qquad (2)$$

Proposition 3 (Admissibility and optimality of h^c) *Given a problem Π with a set of choice variables C and an admissible base heuristic h, h^c is also admissible. Furthermore, if h is the perfect heuristic h^*, then $h^c = h^*$.*

Proof sketch: Let $\mathbf{v}$ be the assignment made to C by an optimal plan π^*. Then π^* is also a plan for $\Pi^{\mathbf{v}}$, since it cannot contain any operators assigning values to C other than those in $\mathbf{v}$, and $h^*(\Pi^{\mathbf{v}}) = \mathit{cost}(\pi^*)$. The properties above follow from the fact that h^c is defined to be the minimum heuristic value for any $\Pi^{\mathbf{v}}$. □

While the value of h^c can be computed as implied by Equation 2, this requires that h be evaluated $|\phi(C)|$ times, which is exponential in $|C|$. For problems in which choice variables exhibit some degree of conditional independence, however, it is possible to reduce the number of evaluations by taking advantage of this structure, which we encode by means of the *choice variable graph*.

The Choice Variable Graph

The choice variable graph (CVG) is a directed graph $G_C = \langle C \cup C_G, E_C\rangle$, where $C_G = \cup_{g \in G}\{C_g\}$ is the set of unary *goal choice variables*, $C_g = \{g\}$. In the following, C_i may denote either one of the explicitly defined choice variables of the problem or some $C_g \in C_G$. To describe the construction of G_C, we first define the notion of *conditional relevance*:

Definition 4 (Conditional relevance) *For $p, q \in F$ and $B \subseteq F$, p is* conditionally relevant *to q given B if:*

- *$p = q$, or*
- *There exists $o \in O$ with $p \in \mathit{Pre}(o)$, $r \in (\mathit{Add}(o) \cup \mathit{Del}(o)) \setminus B$, and r is conditionally relevant to q given B.*

In other words, p is conditionally relevant to q given B if whether p is true in the current state or not changes the cost of achieving q, even when the truth values of fluents in B are held constant. We denote this by $\mathit{rel}(p, q, B)$. This definition can easily be extended to *sets* of fluents: $P \subseteq F$ is conditionally relevant to $Q \subseteq F$ given B if there exist $p \in P$, $q \in Q$ such that $\mathit{rel}(p, q, B)$. The set of edges E_C of G_C is then given by:

$$E_C = \{\langle C_i, C_j\rangle \mid \mathit{rel}(C_i, C_j, \cup_{C_k \in C \setminus \{C_i, C_j\}} C_k)\}$$

In words, there is an edge $e = \langle C_i, C_j\rangle$ in G_C if C_i is conditionally relevant to C_j given the values of all other choice variables. In what follows, we will assume that the CVG G_C given by this definition is *directed acyclic*.

G_C encodes a set of relationships between its nodes similar to those encoded by a Bayesian network, but rather

than associating with each node C_i a *conditional probability table* (CPT) that specifies the probability of a node's taking different values given the values of its parents, it associates a *conditional cost table* (CCT) that specifies the cost of the subproblem of making true some $d \in C_i$ given an assignment to its parent nodes $Pa(C_i)$. For an assignment $\mathbf{v} \in \phi(C_i \cup Pa(C_i))$, this cost is that of a STRIPS problem with goal $G = \mathbf{v}[\{C_i\}]$ and initial state $s \cup \mathbf{v}[Pa(C_i)]$, where the notation $\mathbf{v}[C']$ denotes the set of values in $\mathbf{v}$ which belong to the domain of some $C_i \in C'$. The set of operators of the problem is given by $O^{\mathbf{v}}$, which is computed as in Definition 2.

Rather than calculating the costs of these subproblems and storing them in a table beforehand, the CCT can be represented implicitly as a heuristic that estimates the costs of these problems as they are needed. We denote the heuristic values for such a subproblem with $h(\mathbf{v}[C_i] \mid \mathbf{v}[Pa(C_i)])$. This notation parallels that used in Bayesian nets for the conditional probability of a node being instantiated to a value given the values of its parent nodes, and makes clear the relationship between the subproblems considered here and *factors* in Bayesian nets that consist of a node together with all of its parents. The *choice variable decomposition heuristic* h^{cd} can then be written as follows:

$$h^{cd}(\Pi) = \min_{\mathbf{v} \in \phi(C)} \left[\sum_{i=1}^{|G|} h(g_i \mid \mathbf{v}[Pa(C_{g_i})]) \; + \; \sum_{i=1}^{|C|} h(\mathbf{v}[C_i] \mid \mathbf{v}[Pa(C_i)]) \right] \quad (3)$$

h^{cd} approximates the value of h^c under the assumption of acyclicity, discussed above, and the additional assumption of decomposability, which allows us to obtain heuristic estimates for the global problem by *summing* estimates for disjoint subproblems. It can be shown that under these two assumptions, the values of h^c and h^{cd} are equal for certain base heuristics, notably h^* and h^+. However, for non-optimal delete relaxation heuristics such as h^{add}, the h^{cd} heuristic can sometimes result in more accurate heuristic estimates than h^c, as partitioning the problem into subproblems allows the elimination of the some of the overcounting behaviour typically observed with h^{add}. Note that in Equation 3, while all possible assignments to non-goal choice variables are considered, the values of the goal choice variables are forced to take on their (unique) values g_i. This can be seen as analogous to the notion of *evidence* in Bayesian nets, which are nodes whose values have been observed and which therefore cannot take on other values.

To formalize the assumption of decomposability discussed above, we extend the definition of conditional relevance: an operator o is conditionally relevant to a set of fluents Q given another set of fluents B if $rel(Add(o) \cup Del(o), Q, B)$ holds. We define the *decomposability* of a problem with respect to a set of choice variables in terms of this idea:

Definition 5 (Decomposability) *A problem* Π *is* decomposable *with a set of choice variables* C *if each operator in* Π *is relevant to a* single $C_i \in C$ *given all the other choice variables in* C*, i.e. if for each* $o \in O$*,* $|\{C_i \mid rel(o, C_i, \cup_{C_k \in C \setminus \{C_i\}} C_k)\}| = 1$.

When this decomposability condition is not met, the cost of o may be counted in more than one subproblem, leading to an inadmissible heuristic even if the underlying heuristic used to compute the costs of subproblems is admissible.

Proposition 6 (Equivalence of h^c and h^{cd}) *Given a problem* Π *with a set of choice variables* C *such that the CVG* G_C *is directed acyclic and* Π *is decomposable with* C*, and base heuristic* h^* *or* h^+*,* $h^c = h^{cd}$.

Proof sketch: The proof follows from the fact that due to the definition of decomposability, an optimal (relaxed) plan can be partitioned into a set of subsets such that each constitutes a (relaxed) plan for a subproblem in h^{cd}, and optimal (relaxed) plans for each subproblem can be combined to obtain a global optimal (relaxed) plan. □

Proposition 7 (Admissibility and optimality of h^{cd}) *Given a problem* Π *with a set of choice variables* C *such that the CVG* G_C *is directed acyclic and* Π *is decomposable with* C*, and an admissible base heuristic* h*,* h^{cd} *is also admissible. Furthermore, if* $h = h^*$*, then* $h^{cd} = h^*$.

Proof sketch: The proof follows from that of Proposition 3, and the observation that each operator in an optimal plan π^* contributes cost to exactly one subproblem in h^{cd} due to decomposability. Optimal (admissible) estimates for each subproblem can therefore be summed to obtain optimal (admissible) estimates for the global problem. □

Efficient Computation of h^{cd}

We now turn our attention to how to take advantage of the conditional independence relations between choice variables encoded by G_C, showing how to adapt the recursive conditioning algorithm (Darwiche 2001) to the computation of h^{cd}. Given a Bayesian network $\mathcal{N}$ and evidence $\mathbf{e}$ in the form of an observed instantiation of a subset of the nodes of $\mathcal{N}$, the recursive conditioning algorithm operates by selecting a *cutset* C that when instantiated results in two connected components $\mathcal{N}^l, \mathcal{N}^r$ that are *conditionally independent* of one another given C. The method then enumerates the possible instantiations $\mathbf{c}$ of C, recording each and solving $\mathcal{N}^l$ and $\mathcal{N}^r$ by applying the same method recursively with the pertinent evidence. In enumerating the possible instantiations of the cutset, the observed values of nodes that constitute the evidence $\mathbf{e}$ are respected. When the network on which the method is called consists of a single node, the probability corresponding to the current instantiations of the node and its parents can be looked up directly in the associated CPT. Given two conditionally independent networks $\mathcal{N}^l, \mathcal{N}^r$, the joint probability of the evidence for an instantiation $\mathbf{c}$ of the cutset C is calculated as the product of the probability of the evidence for each, and these results for all possible instantiations of C are summed to obtain the total probability.

The recursive conditioning algorithm can be driven by a structure known as a *dtree*, a binary tree whose leaves correspond to the factors of $\mathcal{N}$ (Darwiche 2001). Each internal node d of the dtree corresponds to a subnetwork $\mathcal{N}^d$ of

$\mathcal{N}$, with the root node corresponding to the full network, and specifies a cutset consisting of those variables shared between its two children d^l and d^r which do not appear in its *acutset*, defined as the set of variables that appear in the cutsets of the ancestors of d. An instantiation of all of the variables that appear in a node's cutset and acutset then ensures that all variables shared between its two children are instantiated, resulting in two networks which are conditionally independent given the instantiation. A dtree node specifies that recursive conditioning be applied to the associated network by instantiating its cutset, and then applying recursive conditioning to the resulting two networks $\mathcal{N}^l$ and $\mathcal{N}^r$ by using d^l and d^r as dtrees for those.

To avoid repeated computations, each dtree node may also maintain a *cache*, which records for each instantiation $\mathbf{y}$ of its *context*, defined as the intersection of the set of variables of the corresponding subnetwork with the nodes of its acutset, the probability resulting from $\mathbf{y}$. While this increases the space requirements of the algorithm, in many settings the associated gains in time are of greater importance. With full caching, recursive conditioning driven with such a dtree is guaranteed to run in $O(n^w)$ time, where n is the number of nodes in the network and the *width* w is a property of the dtree. Given an *elimination ordering* of width w for a network, a dtree with width $\leq w$ can be constructed in linear time.[1] While finding the optimal elimination ordering for a graph which results in the lowest width dtree is an NP-hard problem, many greedy heuristics provide reasonable performance in practice (Dechter 2003).

In order to compute h^{cd} using recursive conditioning, we replace the CPTs associated with each factor with CCTs represented implicitly by a *heuristic estimator* h, which rather than calculating the probability of some instantiation of a node given an instantiation of its parent nodes, estimates the *cost* of achieving a value of a choice variable given values in the domains of its parent variables. To obtain the cost resulting from an instantiation of a cutset, the costs of the two components are *summed* rather than multiplied, and the *minimum* such cost is taken over all of the possible instantiations of the cutset. The fact that each of the goal choice variables C_g must be assigned its single possible value g, and that a choice variable may already be assigned a value in a state from which the h^{cd} is computed, can be seen as equivalent to evidence in the Bayesian networks setting.

The modifications described above result in Algorithm 1. Since the CVGs that we consider are typically small and the time taken by computing the value of a heuristic at each state is the largest factor in heuristic search planners' runtime, we use full caching of computed costs for each subproblem. We construct the dtree used in the algorithm from the elimination ordering suggested by the greedy *min-degree* heuristic, which orders the nodes last to first in increasing order of their degree. Finally, the values of choice variables in the state from which the heuristic is computed, as well as the single values of each of the goal choice variables, are given to the algorithm as evidence.

[1] Elimination orderings and width are beyond the scope of this paper. For an overview of the subject, see e. g. Bodlaender.

Input: A dtree node D
Input: An assignment $\mathbf{v}$ to C
Input: A base heuristic function h
Output: A heuristic estimate $h^{cd} \in \mathbb{R}_0^+$

function RC-h $(D, \mathbf{v})$ **begin**
  if D *is a leaf node* **then**
    $C_i \leftarrow$ the choice variable associated with D
    return $h(\mathbf{v}[C_i] \mid Pa(C_i))$
  endif
  else if $cache_D[\mathbf{v}[context(D)]] \neq$ *undefined* **then**
    return $cache_D[\mathbf{v}[context(D)]]$
  endif
  else
    $h^{cd} \leftarrow \infty$
    for $\mathbf{c} \in \phi(cutset(D))$ **do**
      $h^l \leftarrow \texttt{RC-h}(D^l, \mathbf{v} \cup \mathbf{c})$
      $h^r \leftarrow \texttt{RC-h}(D^r, \mathbf{v} \cup \mathbf{c})$
      $h^{cd} \leftarrow \min(h^{cd}, h^l + h^r)$
    end
    $cache_D[\mathbf{v}[context(D)]] \leftarrow h^{cd}$
    return h^{cd}
  endif
end

Algorithm 1: Recursive conditioning for calculating h^{cd} in a planning problem with choice variables.

Proposition 8 (Correctness of RC-h) *If G_C is acyclic, the* RC-*h algorithm computes the values of the choice variable decomposition heuristic h^{cd}.*

We omit the proof due to lack of space.

Proposition 9 (Complexity of RC-h) *The number of calls made to the underlying heuristic estimator h by* RC-*h is $O(n^w)$, where n is the number of nodes in the CVG G_C and w is the width of the dtree used.*

Proof: Direct from results concerning the complexity of recursive conditioning (Darwiche 2001). □

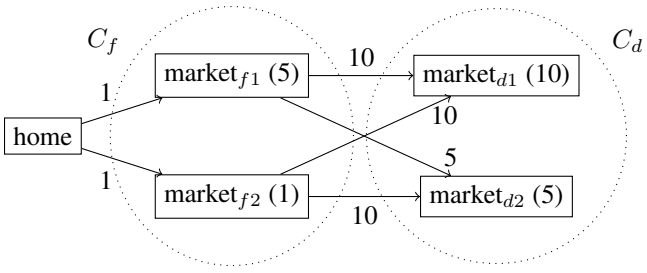

Figure 1: A market problem. Food is available at *market*$_{f1}$ and *market*$_{f2}$, while drink is available at *market*$_{d1}$ and *market*$_{d2}$. Numbers in parenthesis give the cost of each good at the specified market, and those next to each edge show the movement cost.

Example. Consider a problem in which an agent must buy food and drink, and two different markets are available selling each (Figure 1). The CVG for this problem is a tree, with the choice of the food market independent of all other choice variables, the choice of the drink market dependent

only on the food market, and the choice variables for the two goals *have-food* and *have-drink* dependent only on the market chosen for each (Figure 2). The RC-h algorithm begins at the root node of the dtree (Figure 3), and must enumerate the possible instantiations of its cutset $\{C_f\}$. market_{f1} is selected as the first value, and the recursion proceeds with a call to the right subtree. This is an internal node with an empty cutset, so no further variables are instantiated. The algorithm then recurses to the right child, which is a leaf node in which the base heuristic is used to estimate the cost of the goal *have-food* from the initial state $\{\mathit{home}, \mathit{market}_{f1}\}$, with actions adding other values of the choice variable C_f disallowed. There is a single possible plan for this component which consists of buying food at market_{f1}, and this plan has cost 5, which is returned to the parent node. The algorithm then proceeds to evaluate the cost of the left node, which is the cost of making market_{f1} true from the initial state $\{\mathit{home}\}$, 1. Since the cutset of the node is empty, no further instantiations are required and the value $5 + 1 = 6$ is returned to the root node. The algorithm now estimates the cost of the node with cutset $\{C_d\}$, instantiating it first to market_{d1}. In the call to the right child, the cost of achieving market_{d1} from initial state $\{\mathit{home}, \mathit{market}_{f1}\}$ is evaluated, to give cost 10, and in the left child, the cost of buying drink at market_{d1} is evaluated, to give cost 10. The returned values are summed in the internal node with cutset C_d to give cost 20. The value resulting from $C_d = \mathit{market}_{d2}$ is calculated similarly, giving cost $5 + 5 = 10$, which is lower than the previous instantiation, and therefore returned to the root node, in which it is summed with the cost calculated for the right child to give a final cost of $10+6 = 16$ for the instantiation $C_f = \mathit{market}_{f1}$. The cost of choosing $C_f = \mathit{market}_{f2}$ is computed similarly and turns out to be 17, which is higher than the previous estimate, so 16 is returned as the final value. Note that the optimal delete relaxation plan for this problem is to move to market_{f2} and buy food there, and to move from *home* to market_{f1} and from there to market_{d2} to buy drinks there, for a total cost of $1 + 1 + 1 + 5 + 5 = 13$.

The "sequential markets problem" problem can also be seen as the most probable explanation (MPE) problem on hidden Markov models (HMMs). HMMs are dynamic Bayesian processes defined by $H = \langle S, O, T, I, E\rangle$, where S is a set of states, O is a set of observations, $T(s' \mid s)$ for $s, s' \in S$ is the probability of transitioning from s to s', $I(s)$ for $s \in S$ is the probability that the initial state is s, and $E(o|s)$ for $o \in O$, $s \in S$ is the probability of observation o in state s. The MPE problem for HMMs is to compute a sequence of states $s_0, \ldots, s_t$ that maximizes the probability of a sequence of observations $o_1, \ldots, o_t$, where this probability is given by $I(s_0)\prod_{i=1}^{t} T(s_i \mid s_{i-1})E(o_i \mid s_i)$. An MPE problem for an HMM can be encoded as a market problem with a sequence of t different market types, at each of which there is a choice of $|S|$ different markets from which the single required item of that type must be bought. The costs of buying the required item and moving between markets are obtained by taking the negative logarithm of the associated probabilities, so that finding a plan with minimal cost is equivalent to finding a state trajectory with maximum probability for the HMM.

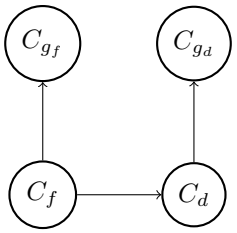

Figure 2: The CVG for the two markets problem shown in Figure 1, with factors $C_f, C_f \to C_d, C_f \to C_{g_f}, C_d \to C_{g_d}$.

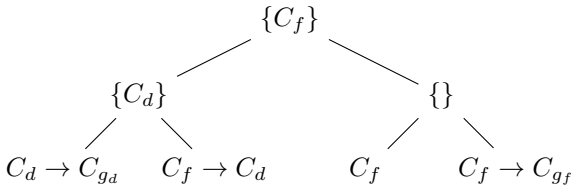

Figure 3: A dtree for the CVG shown in Figure 2, resulting from the min-degree elimination ordering $\langle C_{g_d}, C_{g_f}, C_d, C_f\rangle$. Cutsets are shown for internal nodes.

Implementation

Though the recursive conditioning algorithm is naturally able to avoid recomputing the cost of the same component twice for the same assignment to its set of choice variables, it is designed to be run once and makes no attempt to preserve information between runs. We observe that the heuristic values $h^{\mathbf{v}}(C_i = \mathbf{v}[C_i] \mid \mathit{Pa}(C_i))$ do not change from state to state for components in which there is no non-choice fluent that is conditionally relevant to C_i given $\mathit{Pa}(C_i)$. These values can then be cached in each leaf node of the dtree the first time they are computed, and reused in later heuristic computations. We have implemented this optimization in the results described below.

Domains and Experimental Results

We have found that in practice few problems conform to the rather strict requirements that we have laid out for h^{cd}. Multivalued variables in planning problems typically represent properties of states, rather than choices made by plans, and their values tend to change many times during the execution of a plan, violating the fundamental property that we have used to define choice variables. One technique that we have investigated in certain problems is *stratification*, which consists of replacing a single multivalued variable X with several multivalued variables $X_0, \ldots, X_n$, with X_i representing the ith value assigned to X in the original problem. Each of these variables can than be treated as a choice variable in the new encoding, and the part of the CVG corresponding to these variables forms a chain in which each X_i is dependent only on X_{i-1}, leading to a width of 1. However this approach does not generally pay off as variables that were previously dependent only on X now depend on each of the variables X_i, increasing the width of the graph to the horizon used to obtain the new encoding.

We were also initially optimistic that h^{cd} could give informative estimates in domains such as Sokoban or Storage, in which a number of objects have to be placed in a set of goal locations. The final goal location chosen for an object can here be seen as a choice variable. However, in the absence of an ordering over the objects, the position of each object affects the available choices for the others, and even if an ordering is imposed, the cost of the last object to be placed is dependent on the locations of all other objects, leading to the CVG's width growing with problem size.

One area in which it has been easy to apply h^{cd}, however, is in planning encodings of problems from the graphical models setting. We now present two domains that we have adapted to the planning setting for which h^{cd} is able to compute optimal values. We compare the performance of h^{cd} to that of a standard delete relaxation heuristic, the cost of the relaxed plan obtained from the additive heuristic h^{add} (Keyder and Geffner 2008). We use the same heuristic within the framework of h^{cd} to obtain the heuristic estimates for the cost of each subproblem $h(C_i \mid Pa(C_i))$. The heuristics are used in greedy best-first search with delayed evaluation, with a second open queue for states resulting from helpful actions (Helmert 2006). Helpful actions for h^{cd} are those which are helpful in any one of the subproblems and which can be applied in the current state. All experiments were run on Xeon Woodcrest 2.33 GHz computers, with time and memory cutoffs of 1800 seconds and 2GBs, respectively.

Minimum Cost SAT. As a canonical example of a constraint satisfaction problem, we consider a version of the boolean satisfiability problem with three literals per clause in which a satisfying assignment with minimum cost must be found (MCSAT) (Li 2004). The natural choice variables for the problem consist of the sets $\{x_i, \neg x_i\}$ for each problem variable x_i. An encoding that results in an acyclic CVG is obtained by imposing an ordering over the set of variables of the problem, and using operators which can set variables which are not highest-ranked in any clause freely, but which ensure for the other variables that the clauses in which they are highest-ranked are already satisfied if the assignment made by the operator itself does not satisfy the clause. We omit some details of the encoding here due to lack of space, but the end result is that h^{cd} can be used to compute the optimal cost in this type of encoding. The number of evaluations of the base heuristic for a single call to h^{cd} is exponential in the width of the ordering over the variables that is used to generate the problem.

We generated random MCSAT problems with 4.3 times as many clauses as variables, a ratio that has been shown to produce problems for which satisfiability testing is hard (Mitchell, Selman, and Levesque 1992). The number of variables in the problems ranged from 5 to 34. The choice variable heuristic h^{cd} is optimal for this domain, while the additive heuristic h^{add} produces both overestimates and underestimates, with the error in the estimation growing as the number of variables increases. When used in search, h^{cd} is able to solve problems of up to 25 variables with orders of magnitude fewer node expansions than h^{add}. However, for the ratio of clauses to variables that we consider, the CVG is usually clique-like, and therefore has treewidth close to the number of variables in the problem. The computation of h^{cd} therefore rapidly becomes unfeasible, while h^{add} is able to scale up to larger instances. The higher informativeness of h^{cd} does not pay off in terms of plan cost either, with the plans found having roughly equal cost for both heuristics.

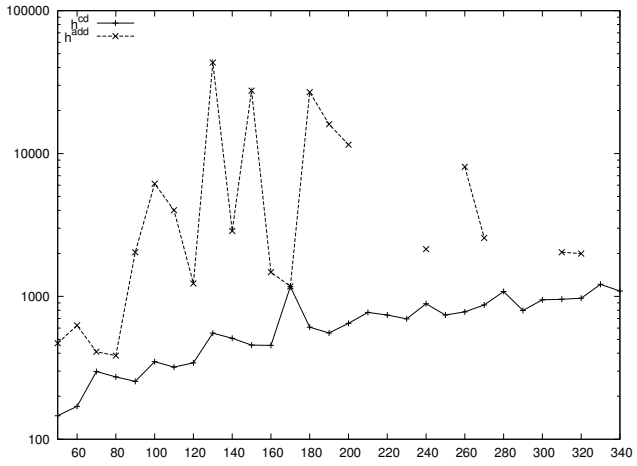

Figure 4: Node expansions on the Sequential Markets domain.

Sequential Markets Problem. We generated market problems such as those described above, with the number of markets of each type fixed at 15, and the number of different types of markets varying between 50 and 340. The h^{cd} heuristic computes optimal heuristic values for this domain that are 2-6 times cheaper than those computed by h^{add}, with the effect becoming more pronounced as the number of market types is increased. When the two heuristics are used in greedy best-first search, h^{cd} scales up well due to the constant width of the domain, solving all 30 problems with much fewer node evaluations than those required by h^{add} in the 21 problems it is able to solve (Figure 4). The computation time of h^{cd} is roughly 10 - 20 times slower than that of h^{add} in this domain, and due to the constant width of the CVG does not change as problem size is varied. The heuristic is also beneficial in terms of the costs of the plans that are found, with h^{cd} finding lower cost plans for all of the instances solved with both heuristics, and the difference in cost growing with the size of the problem.

Conclusions

We have introduced a new type of invariant in planning that we call choice variables, multivalued variables whose values can be set at most once by any plan. By reasoning by cases about different assignments to these variables and excluding operators that violate these assignments, we obtain the h^{cd} heuristic that goes beyond the delete relaxation in its estimates. The values of this heuristic can be computed by adapting inference techniques for graphical models to the planning setting, with the complexity of the heuristic computation then depending on the treewidth of the graph that describes the causal relationships between choice variables.

Our attempts to apply h^{cd} to benchmark planning problems have until now proved disappointing due to the lack of

domains with variables that naturally exhibit the choice variable property. We have investigated the use of new encodings to induce this structure, however such transformations usually result in the treewidth of the CVGs increasing with domain size, and the computational overhead of the heuristic swiftly becoming impractical. We hope to eventually address this challenge by considering meaningful relaxations of these graphs that decrease the width parameter while respecting the essential features of the problem.

References

Amir, E., and Engelhardt, B. 2003. Factored planning. In *Proc. of the 18th IJCAI*, 929–935.

Bodlaender, H. L. 1993. A tourist guide through treewidth. *Acta Cybernetica* 11:1–23.

Brafman, R. I., and Domshlak, C. 2006. Factored planning: How, when, and when not. In *Proc. of the 21st AAAI*, 809–814.

Darwiche, A. 2001. Recursive conditioning. *Artificial Intelligence* 126(1-2):5–41.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Fabre, E.; Jezequel, L.; Haslum, P.; and Thiébaux, S. 2010. Cost-optimal factored planning: Promises and pitfalls. In *Proc. of the 20th ICAPS*, 65–72.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In *Proc. of the 18th ECAI*, 588–592.

Li, X. Y. 2004. *Optimization Algorithms for the Minimum-Cost Satisfiability Problem*. Ph.D. Dissertation, North Carolina State University.

Mitchell, D. G.; Selman, B.; and Levesque, H. J. 1992. Hard and easy distributions of SAT problems. In *Proc. of the 12th AAAI*, 459–465.