Hybrid Constraint Tightening for Solving Hybrid Scheduling Problems

Paper Id #726

Abstract

Hybrid Scheduling Problems (HSPs) contain both temporal and finite-domain variables, as well as constraints between them. A hybrid constraint over temporal and finite-domain variables often models situations where different assignments to a subset of finite-domain variables result in different bounds on a temporal constraint. The insight we examine in this paper is that some temporal constraint propagation is possible even before finite-domain variables are assigned, by giving the temporal constraint the tightest bound consistent with all (remaining) feasible finite-domain variable values. We describe a hybrid constraint-tightening algorithm that can proactively prune the search space of HSPs and can be run as a preprocessing step independently of the CSP solver used. We examine the efficiency of this algorithm analytically, and demonstrate that it reduces the expected runtime of search by a significant margin in the HSPs we are studying.

Introduction

Recently, Schwartz (2007) has combined finite-domain Constraint Satisfaction Problems (CSPs) with Disjunctive Temporal Problems (DTPs) into a single framework of Hybrid Scheduling Problems (HSPs). This framework allows the representation of scheduling problems where certain aspects of the problem are best represented through finite-domain variables and constraints, while other aspects are best represented as a DTP. The HSP framework allows finite-domain variables and temporal variables to interact through hybrid constraints, which are simply a disjunction of a finite-domain constraint and a temporal constraint.

As an example HSP, suppose Amy is responsible for generating a schedule for an AI conference. For each session, finite-domain variables represent aspects such as topic and location, and temporal variables represent the start and end times. Aspects like duration and lag time between sessions would likely depend on the values of the topic (there are more papers on some topics than others) and location variables, respectively. Sessions might also be temporally constrained to occur within a specific period of time. Consider one such example: Amy must schedule two sessions within the last three hours of the day. The topic of the first session could be either CSPs or Multi-Agent Systems (MAS), which would take one hour or two hours respectively. Similarly, the second session could be on HSPs, Reinforcement Learning (RL), or Game Theory (GT), which would take one, two, or three hours respectively. Because of their similarity, Amy has decided not to schedule both a CSP and HSP session.

For this particular example, each session has three variables associated with it: a finite-domain variable representing the session's topic, and temporal variables representing the start and end times. A finite-domain constraint would prevent the possibility of assigning the values CSP and HSP to the first and second sessions respectively. Temporal constraints would force the start of the second session to happen after the end of the first and the end of the second session within three hours of the start of the first. Finally, hybrid constraints would dictate how the duration of each session, the distance between its start and end variables, is a function of the topic assigned.

Many hybrid constraints represent *conditional* temporal constraints, which are temporal constraints whose particular bounds depend on the values assigned to some subset of finite-domain values. For example, the durations of the sessions above rely on the value of the topic variable associated with each session, and thus could be represented as conditional temporal constraints. Moffitt, Peintner, and Pollack (2005) noted that conditional temporal constraints have the property that, at any point in time, one can assume the temporal constraint with the tightest bound consistent with *all* remaining feasible values. In the example above, we know that regardless of which topics are eventually assigned, the duration of each session will be at least an hour. Knowing this would allow Amy to prune the value "GT" from the topic variable of the second session, since it, together with the first session, would inevitably require more than three hours. As the domains of possible values for finite-domain variables are reduced, the temporal constraints tighten to guide CSP solving. Thus, if Amy eliminates HSPs as a possible topic for the second session, she would be able to reason that the second session would now take at least two hours, eliminating and pruning any topics that take longer than an hour for the first session.

The new insight we explore in this paper is that we can achieve the kind of reasoning described in the example above by explicitly transforming hybrid constraints into tighter, more effective constraints that enable typical search techniques to proactively prune infeasible values. This allows our hybrid constraint tightening (HCT) algorithm to remain modular, leaving the implementation of the search algorithm untouched, unlike previous approaches. The HCT algorithm applies to a more general set of hybrid constraints than prior approaches and can be more efficient than those approaches by reasoning about the constraints once where other approaches might reason about them exponentially many times. In summary, the advantage of this work over similar, previous approaches is three-fold: it applies to a more general set of hybrid constraints; it is modular, requiring no changes to the search implementation; and it is more efficient by

amortizing its cost over the entire search. We examine the efficiency of this algorithm analytically and demonstrate that it reduces the expected runtime of search by a significant margin within a particular space of hybrid scheduling problems.

This paper proceeds by providing the reader with a brief description of the relevant background necessary to understand our hybrid constraint algorithm. We then describe the algorithm, followed by an evaluation of its performance, before providing some closing thoughts.

Background

Finite-Domain CSPs

A finite-domain Constraint Satisfaction Problem (CSP) can be defined as a tuple $<V, C>$ where V is a set of variables and C is a set of constraints. V is itself a tuple $<\lambda, D>$ where λ is the variable's label, and D its domain, represented by a set of possible values. A constraint specifies valid tuples of values for some subset of variables. An assignment of a CSP is a binding of one value to each variable. Such an assignment is called a solution if it respects all the given constraints. Though finite-domain CSPs take time exponential in the number of variables to solve in the worst case, techniques such as variable and value ordering, constraint learning, and constraint propagation, among others, significantly reduce the expected solve time (Dechter, Meiri, and Pearl 1991).

DTPs

Because finite-domain CSPs are ill-equipped for handling variables with continuous domains, such as time, recent work has been done to solve temporal-CSPs efficiently (Stergiou & Koubarakis 1998; Oddi & Cesta 2000; Tsamardinos & Pollack 2003). The temporal-CSP is a subclass of the general CSP where each variable represents a point in time. An example of a temporal-CSP is the Simple Temporal Problem (STP) (Dechter, Meiri, and Pearl 1991). STPs contain a special class of constraints called temporal difference constraints (TD-constraints) that are linear inequalities of the form $x - y \leq b$, where x and y are both time points and b is an element of the real numbers. Thus, a STP is a tuple $<V, C>$ where V is a set of time points, and C is a set of TD-constraints. An element of V is still considered a variable, complete with a label and domain, but its associated domain of possible values is implicit. A solution to a STP is an assignment of times to all time points of V such that all constraints in C are satisfied, and a solution can be found in polynomial time using an all-pairs-shortest-path algorithm (Cormen 1990).

An additional example of a temporal-CSP is the Disjunctive Temporal Problem (DTP) (Stergiou & Koubarakis 1998). DTPs are constructed of disjunctive temporal constraints, which are simply disjunctions of TD-constraints. Such a constraint is satisfied if at least one of the component TD-constraints is satisfied. A DTP, then, is the pair $<V, C>$ where V is a set of time points, and C is a set of disjunctive temporal constraints. It is important to note that by selecting a TD-constraint for each disjunctive temporal constraint, we have a "component STP". Thus, a DTP is solvable if at least one of its component STPs is solvable. Although solving the STP takes polynomial time, there are exponentially many (in the number of disjunctive temporal constraints) possible STPs to check for consistency. Therefore solving DTPs in general takes exponential time (Dechter, Meiri, and Pearl 1991).

HSPs

A DTP is good for representing and solving scheduling problems, while finite-domain CSPs are good for selection problems. Put another way, DTPs are good for deciding when to do things, and finite-domain CSPs are good for deciding what combinations of things to do. A Hybrid Scheduling Problem (HSP) (Schwartz 2007) combines the full expressive power of the finite-domain CSPs and temporal CSPs into a single framework. A HSP is a tuple $<V, C>$. The set of variables V can be partitioned into two sets: V_F, the set of finite-domain variables represented by the tuple $<\lambda, D_F>$; and V_T, the set of temporal variables $<\lambda, D_T>$ where D_T contains real numbers. The set of constraints C can be partitioned into three sets: C_F, the set of finite-domain constraints defined over V_F; C_T, the set of disjunctive temporal constraints defined over V_T; and C_H, the set of hybrid constraints defined over $V_F \cup V_T$ excluding C_F and C_T. A hybrid constraint is defined to be the disjunction of exactly one finite-domain constraint and exactly one disjunctive temporal constraint, which can contain multiple finite-domain variables and multiple temporal differences, respectively.

A HSP is solved when a satisfying assignment is made to each variable in V such that all the constraints in C are respected. Clearly, a solution to the overall HSP problem must also be a solution to the finite-domain CSP sub-problem $<V_F, C_F>$ and to the temporal CSP sub-problem $<V_T, C_T>$. However, due to C_H, solutions to these pure subproblems cannot necessarily be combined to solve the overall HSP. We call these sub-problems the CSP and DTP sub-problems respectively.

Related Work: DTP_{FD}

A Disjunctive Temporal Problem with Finite-Domain Constraints (DTP_{FD}) (Moffitt, Peintner, & Pollack 2005) is defined as a tuple $<V, C>$. Each variable V designates a tuple $< \lambda, D_F, D_T>$ containing a label λ, a component D_F, which is a finite domain of values, and a temporal component D_T, which is an implicit temporal domain consisting of real numbers. C can be partitioned into two sets: C_F, a set of typical finite-domain constraints defined over the finite-domain component of the variables V; and C_T, a set containing both typical temporal constraints and *conditional* temporal constraint of the form

Topic = {HSP} → Start – End ≤ -60 Topic = {RL} → Start – End ≤ -120 Topic = {GT} → Start – End ≤ -180

Start – End ≤ -60 Topic = {RL, GT} → Start – End ≤ -120 Topic = {GT} → Start – End ≤ -180

Figure 1: Example Hybrid Constraint Transformation

a) Original hybrid constraints expressing minimum duration based on topic

b) Constraints after applying the HCT algorithm

$$x_i^t - x_j^t \leq B_{ij}(x_i^f, x_j^f)$$

where B_{ij} is the bounds function mapping $D_F \times D_F \rightarrow \Re$. Since D_F contains a finite number of values, this function B_{ij} can be considered a "bounds table". Thus, the actual bounds enforced are contingent on the value assigned to the finite-domain component of each variable's domain.

Moffitt *et al.* show that this representation enables some efficient pruning of values during search. Their, "least-commitment approach" works by recognizing that, at any point in time during a search, a conditional temporal constraint can be temporarily replaced by a non-conditional constraint with a constant bound, namely,

$$x_i^t - x_j^t \leq \max_{ij} [\, B_{ij}(x_i^f, x_j^f)].$$

This helps prune the search space among the temporal variables without eliminating possible solutions. Likewise, the other temporal constraints of the problem may imply a lower bound to the temporal difference $x_i^t - x_j^t$. This lower bound, $\min_{ij}$, precludes any combination of the finite-domain values such that $B_{ij}(x_i^f, x_j^f) < \min_{ij}$. Hence, these combinations are also pruned from consideration.

Since changing the domain of a finite-domain variable can introduce a new temporal constraint, or alternatively, a new temporal difference can prune finite-domain variables' values, this least-commitment pruning requires examining the conditional temporal constraints after each value assignment. As search assigns values to variables, domains are further pruned safely. However, if search finds an infeasibility, the decisions about added temporal constraints and pruned domains may need to be retracted. This means that the least-commitment reasoning must be intertwined with backtracking, or must be redone after every assignment.

Hybrid Constraint Tightening Approach

The DTP_{FD}'s conditional temporal constraint is a form of hybrid constraint, capturing the relationship between the temporal and finite-domain aspects of a particular variable. As Schwartz (2007) has pointed out, however, hybrid constraints in HSPs are more general because they support disjunctions across constraints involving arbitrary finite-domain and temporal variables. So, in our running conference scheduling example, the HSP formulation would allow the duration of a session to be affected by multiple variables (not just the session topic, but also location, moderator, etc.) In fact, just capturing the fact that the minimum duration of a session about HSPs is 1 hour using DTP_{FD} means defining two variables (for session start and session end), each of which has a "topic" component, with an added constraint that the finite-domain values (topic) for each variable be the same. The HSP representation is simpler, as shown in Figure 1a, where we have represented disjunctions in their implicative forms.

Furthermore, the DTP_{FD} specification requires a value in the bounds table for every combination of finite-domain values, whereas in an HSP the hybrid constraints might only apply to some values. In our example, if Amy could choose something like "mixer" as a session topic (meaning that there are no formal paper presentations), then this activity might be able to shrink or expand to any available time, such that it would be unlikely that a user would think to specify temporal constraints associated with it.

Our approach is to map the DTP_{FD} least-commitment strategy into the more general HSP framework so as to solve HSPs more efficiently thanks to being able to prune the search space based on temporal constraints even before associated finite-domain variables are set. As shown in Figure 1b, it is certainly possible to rewrite standard hybrid constraints (Figure 1a) in a way that makes explicit otherwise implicit constraints (in this case, that any session must take at least an hour). Arguably, the onus could be placed on the user to uncover all of these implicit constraints, but doing so can be unnatural and time-consuming for the user, and would need to be redone when constraints change. Thus, we have developed our Hybrid Constraint Tightening algorithm to automate this process.

The HCT Algorithm

Our Hybrid Constraint Tightening (HCT) algorithm (shown later in Figure 3) involves three basic steps. The first step is to group hybrid constraints based on the structure of the temporal constraint involved. Figure 1a shows one such group of three hybrid constraints corresponding to the example above, where the minimum duration of a particular session is determined by the choice of topic. Once these hybrid constraints have been grouped, the next step is to sort the hybrid constraints based on the bounds they impose on the temporal constraint, as is also already shown in Figure 1a. The third step is to construct new hybrid constraints based on the old hybrid constraints, as displayed in Figure 1b. These new hybrid constraints are formed by replacing the current finite-domain constraint portion of each hybrid constraint with a new finite-domain constraint that is a disjunction of the current finite-domain constraint with all finite-domain constraints that have more restrictive bounds. This is allowable because bounds are ordinal. In other words, if a given finite-domain constraint implies a relatively tight bound, it will also inherently imply any bound that is less-restrictive.

Notice in Figure 1b that, in this particular example, one temporal constraint always holds, and thus is added explicitly. Furthermore, as values are removed from the domains of finite-domain variables due to propagation, the

Location = {Room A} → End – Noon ≤ -60 ∨ Noon – Start ≤ -60
Location = {Room B} → End – Noon ≤ -120 ∨ Noon – Start ≤ 0
Location = {Room C} → End – Noon ≤ -120 ∨ Noon – Start ≤ -60

End – Noon ≤ -60 ∨ Noon – Start ≤ 0
Location = {Room A,C} → End – Noon ≤ -60 ∨ Noon – Start ≤ -60
Location = {Room B, C} → End – Noon ≤ -120 ∨ Noon – Start ≤ 0
Location = {Room C} → End – Noon ≤ -120 ∨ Noon – Start ≤ -60

Figure 2: Example Disjunctive Hybrid Constraint Transformation
a) Original hybrid constraints
b) Constraints after applying the HCT algorithm

tightened hybrid constraints will enforce the tightest bounds on the relevant temporal constraint possible. In contrast, the hybrid constraints, as expressed in Figure 1a, would enforce a temporal bound only after the finite domain variable involved was assigned a specific value. As an additional benefit, if a minimum bound was determined for the particular temporal constraint, then all combinations of values that would enforce bounds tighter than this minimum can be pruned from the relevant finite-domain variables. For example in Figure 1b, a minimum bound of -90 would immediately allow us to prune values RL and GT from the corresponding topic variable.

Notice that this section highlights a version of our HCT algorithm that works on hybrid constraints containing a single temporal difference. This has been done for clarity and conciseness, although the algorithm is easily amendable to handle hybrid constraints with multiple temporal disjuncts. In the single temporal difference version, hybrid constraints were constructed for each unique temporal bound, with the finite-domain constraint component constructed by merging all finite-domain constraints that imply a bound equally or more restrictive. When a disjunction of temporal differences is involved, this same process occurs, except that a hybrid constraint must be constructed for each combination of possible bounds, instead of each bound.

As an example, suppose Amy was selecting locations for a given session. Three rooms have the correct equipment available for the session, but all three rooms are occupied for a period of time in the middle of the day. To specify this requires hybrid constraints containing temporal disjunctions as shown in Figure 2a. When temporal disjunctions are involved, we are tightening over a set of bounds, instead of a single bound. Thus, a tightened hybrid constraint is added for each *combination* of bounds, with the corresponding finite-domain constraint being constructed by merging the finite-domain constraint portion of any hybrid constraint implying a set of bounds strictly tighter than the specified combination of bounds. In the example, there are two unique bounds for each temporal disjunct, and thus four possible combinations of bounds. The resulting tightened hybrid constraints are displayed in Figure 2b.

Analysis of the HCT Algorithm

Our algorithm involves iterating over every hybrid constraint. For each hybrid constraint involved in the

```
Map<TempConstStruct, SortedList> map;

// Group hybrid constraints based on the structure of
// the temporal constraint portion of the HC.
For each hybrid constraint hc in HSP
  SortedList group = map.get(hc.tempConstStruct);
  // Insert hybrid constraint into list sorted by
  // decreasing tightness of temporal bound.
  // If group already contains a hybrid constraint
  // with the same bound, merge finite-domain
  // constraints to form one new hybrid constraint.
  group.insert(hc);

//Build new tightened constraints based on groups.
For each Sorted List group in map
  FDConstraint newFDConstraint;
  For each hybrid constraint hc in group
    // To build new hybrid constraint, merge this
    // constraint with all finite-domain constraints
    // that implied tighter bounds.
    newFDConst = newFDConst ∪ hc.FDConstraint;
    problem.add (new HybridConstraint (newFDConst,
hc.tempConst));
```

Figure 3: A Hybrid Constraint Tightening Algorithm

problem, we compare the bounds of this hybrid constraint to the bounds of every other matching hybrid constraint. A hybrid constraint is considered "matching" if the structures of the temporal constraint portion of the hybrid constraints are the same. Thus, the runtime of this algorithm is $O(HB)$, where H is the number of hybrid constraints involved in the problem, and B is the size of the largest group of matching hybrid constraints. Since B is the size of some subset of the entire group of hybrid constraints, the runtime of our preprocessing algorithm is bounded by $O(H^2)$. If hybrid constraints containing multiple temporal disjunctions were included, the runtime of this algorithm would increase to $O(HB^k)$, where k is a constant representing the maximum number of temporal disjuncts, with an upper bound of $O(H^{k+1})$. In our experience, the value of k tends to be relatively low.

The "least-commitment" approach used in (Moffitt, Peintner, & Pollack 2005) is similar, but instead requires explicit updates of the implied temporal bounds during search. Their approach requires that upon each change to the domain of a finite-domain variable, each conditional hybrid constraint must be reexamined, and new tighter consistent temporal constraints be determined. Although this is also a polynomial-time algorithm, because finite-domain CSPs may require trying an exponential number of assignments to finite-domain variables, this algorithm may be applied an exponential number of times. Our approach is applied once, and thus the cost is amortized over the entire run of the search.

Additionally, since the least-commitment approach requires these temporal constraints to be updated upon any change to the finite-domain components, such an approach requires changes to the implementation of the search. Our HCT algorithm is much more modular, requiring a simple application of the algorithm prior to search. Because the

Original Approach: Completed search on 87.6% of problems

	Mean of Completed	Mean	Median
Time (ms)	7,795.2	23,461.1	62.0
Node Visits	10,075.2	23,110.1	211.0
Backtracks	186.1	382.2	0.0

HCT Approach: Completed search on 100% of problems

	Mean of Completed	Mean	Median
Time (ms)	184.4	184.4	47.0
Node Visits	211.5	211.5	102.0
Backtracks	2.7	2.7	0.0

Margin of Improvement

	Mean of Completed	Mean	Median
Time (ms)	97.60%	99.2%	24.2%
Node Visits	97.90%	99.1%	51.7%
Backtracks	98.50%	99.3%	0.0%

Table 1: *Mean and median search times of 500 randomly generated search problems using two approaches*

complex relationship between finite-domain and temporal variables become more explicit, the tightened constraints take advantage of typical search strategies to automatically assert the tightest possible temporal constraints at all times.

Empirical Evaluation of the HCT Algorithm

Having demonstrated the theoretical advantages of our HCT approach over previous approaches, we empirically investigate its performance on HSPs of the type that arise in application domains that we study. We randomly generated 500 problems roughly corresponding to the AI conference example as specified above. The details of the problem generator are available at [website not included for anonymity]. It was calibrated so that 60-65% of all problems generated resulted in a feasible solution. We use a Java implementation of an HSP solver, graciously lent to us by Schwartz, on a 2 GHz processor with 2 GB of RAM to solve each of the 500 problems using two approaches: one where we apply our HCT algorithm prior to solving, and one where we do not. A cap of two and a half minutes was placed on the search for each problem. All results were determined statistically significant using a Student paired T-test (Cohen 1995).

Table 1 shows the results from this problem set. The cap on search time led to a completion rate of only 87.6% when we did not apply the HCT algorithm compared to 100% completion when applying the HCT algorithm. Even with this capped search time, Table 1 indicates that the expected runtime, expected number of backtracks within search, and the expected number of node visits are all reduced by a margin of greater than 99% when our HCT algorithm is applied prior to search. It is promising that all measures of runtime, including the implementation independent measures of node visits and backtracks, have been reduced.

Table 1 also shows that the median statistics demonstrate a less drastic reduction of expected runtime. It is important to realize that these problems were

Topic = {"mixer"} → No constraint
Topic = {CSP} → Start – End ≤ -60
Topic = {MAS} → Start – End ≤ -120

Start – End ≤ -60
Topic = {CSP, MAS} → Start – End ≤ -60
Topic = {MAS} → Start – End ≤ -120

Figure 4: Example Partial Hybrid Constraint Transformation
a) Original hybrid constraints
b) Constraints after applying the HCT algorithm

generated randomly. Thus, some problems ended up relatively over and under-constrained, which tend to require much less search to find a solution or infeasibility. Under-constrained problems tend not to benefit as greatly from the additional pruning, since many possible assignments of values are likely to lead to a feasible solution. This explains why, although we see a stark margin of improvement in the mean statistics, the benefit was less pronounced on the median statistics. Promisingly, the observed reduction in the median statistics implies that our approach lowers the runtime on most problems. The mean runtime of the HCT algorithm over all runs was just under four milliseconds, with the maximum runtime taking 32 milliseconds. All recorded search times using the HCT approach include the runtime of the HCT algorithm. With a negligible expected runtime on the order of milliseconds compared to an expected search time savings on the order of tens of seconds, we suspect our HCT algorithm is worth applying to any HSP not known to be easily solved.

These results demonstrate that our preprocessing algorithm is effective at reducing the expected search time for this particular space of HSPs. However, we also wanted to test our method on HSPs that cannot be efficiently represented as a DTP_{FD}. As discussed above, Amy may want to consider a mixer topic, which would imply no inherent minimum duration. We display constraints relevant to this example in Figure 4. The disadvantage of a hybrid constraint like this is that, as

	a) Partially Spec.		b) Many FD-Vars	
Orig. Approach	**Mean**	**Median**	**Mean**	**Median**
Time (ms)	2,012.7	16.0	1,705.7	15.0
Node Visits	3,247.3	74.5	1,731.5	89.0
Backtracks	94.0	0.0	32.5	1.0
HCT Approach	**Mean**	**Median**	**Mean**	**Median**
Time (ms)	610.7	16.0	21.1	0.0
Node Visits	834.8	61.0	48.0	17.0
Backtracks	39.4	0.0	0.0	0.0
Improvement	**Mean**	**Median**	**Mean**	**Median**
Time (ms)	69.7%	0.0%	98.8%	100.0%
Node Visits	74.3%	18.1%	97.2%	80.9%
Backtracks	58.1%	0.0%	99.9%	100.0%

Table 2: *Mean and median search times of 500 randomly generated search problems using two approaches.*
a) In this problem set, temporal constraints are not implied for all assignment of values to finite domain variables.
b) In this problem set, conditional temporal constraints are based on larger number of finite-domain variables.

shown in Figure 4b, no temporal constraint can be immediately inferred. Thus, search is unable to take advantage of the tighter constraints as early in the search process.

To test the effectives of the HCT algorithm on such problems, we generated a new set of 500 problems that correspond to temporal constraints that exist for only certain combinations of finite-domain values. The results from these runs are in Table 2 (column a, on the left). Once again, we see significant reductions in the expected runtime statistics when the HCT algorithm is applied. However, because such problems require some assignment to variables before the search is able to take advantage of the tightened variables, such problems only demonstrated around 70% improvement. Since not all assignments of values to finite-domain variables imply temporal constraints, the problems tended to be less constrained, leading to even smaller improvements in the median runtime. Though the margins of improvement are much smaller than in the original problem space, it still easily offsets the overhead of the HCT algorithm.

The second variation of problems we looked at were those with hybrid constraints containing many finite-domain variables. The bounds tables involved in a DTP_{FD} have dimension equal to the number of finite-domains involved with the constraint. Since a DTP_{FD} contains at most two such finite-domains in its conditional temporal constraints, the bounds tables tend to stay reasonable in size. However, as the number of finite-domain variables grows, the number of combinations of their values grows exponentially, and each new combination may potentially imply a new, unique bound. Thus, these conditional temporal constraints may grow exponentially in size as we increase the number of finite-domain variables involved in hybrid constraints.

We randomly generated another set of problems that contain such hybrid constraints, and discovered that these problems also tended to be easier to solve. The reason is that the conditional temporal constraints are conditional on a greater number of finite-domain variables, eliminating values from many finite domain variables when conditional temporal constraint bounds were determined infeasible. Despite problems being easier to solve, as seen in Table 2 (column b on the right), applying the HCT algorithm reduces the expected runtime, node visits, and backtracks by a margin similar to the original problem set. Although the larger number of hybrid constraints will affect the runtime of the HCT algorithm, the number of hybrid constraints involved in the problem does not increase, only their effectiveness assisting search in pruning the search space.

Our results show that, in all but the easiest randomly-generated HSPs, applying the HCT algorithm reduces the time it takes search to find a solution or detect a global inconsistency, even in problems containing hybrid constraints that are not specifiable in a typical DTP_{FD}.

Conclusion

We have presented the HCT algorithm, which automates the explicit transformation of hybrid constraints into tighter, more effective constraints that proactively prune the search space during search. We demonstrated both analytically and empirically that the HCT algorithm applies to and reduces the expected search time of problems containing hybrid constraints that previous approaches were unable to handle efficiently. In our analysis section, we showed that we can apply the HCT algorithm separately from the search algorithm, in an efficient manner, amortizing the cost across the entire search. This is in contrast to previous approaches, which required such reasoning to be implemented into the search algorithm itself and applied after each of the possibly exponential number of assignments of values to variables. The significance of the work presented in this paper thus is that it generalizes the least-commitment strategy for speeding CSP solving to apply to a broad class of important HSP problems, using a modular and low-overhead hybrid constraint tightening algorithm.

The effectiveness of the HCT algorithm as it extends to other interesting HSPs is likely to depend on factors such as the relative number and tightness of the conditional temporal constraints involved in the problem as well as the overall level of constrainedness of the problem. In the future, we hope to better understand the range of HSPs where our HCT algorithm is most effective, possibly generating new, more complex versions of the HCT algorithm to adjust to such factors. We would also like to perform a more thorough, empirical comparison of the HCT algorithm with the prior least-commitment approach.

References

Cohen, P. 1995. Empirical Methods for Artificial Intelligence. Cambridge, Mass.: MIT Press.

Cormen, T., Leiserson, C., and Rivest, R. 1990. Introduction to Algorithms. Cambridge, Mass.: MIT Press.

Dechter, R., Meiri, I., and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence* 49, 61-95.

Moffitt, M., Peintner, B., and Pollack, M. 2005. Augmenting Disjunctive Temporal Problems with Finite-Domain Constraints. In *Proc. of AAAI-2005*, 1187–1192.

Oddi, A., and Cesta, A. 2000. Incremental forward checking for the disjunctive temporal problem. In *Proc. of ECAI-2000*, 108-112.

Schwartz, P. 2007 Managing Complex Scheduling Problems with Dynamic and Hybrid Constraints. Doctoral Thesis. University of Michigan.

Stergiou, K., and Koubarakis, M. 1998. Backtracking algorithms for disjunctions of temporal constraints. In *Proc. of AAAI-98*, 248-253.

Tsamardinos, I. and Pollack, M. 2003. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence* 151, 43-89.