Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

Hybrid Constraint Tightening for Solving Hybrid Scheduling Problems

James C. Boerkoel Jr. and Edmund H. Durfee

Computer Science and Engineering, University of Michigan
Ann Arbor, MI 48109
{boerkoel, durfee}@umich.edu

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 temporal constraints. 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 is run as a preprocessing step independently of the search algorithm used. We examine the efficiency of this algorithm analytically, and give preliminary results showing that it reduces the expected runtime of search by a significant margin in the kinds of 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 naturally represented through finite-domain variables and constraints, while other aspects are best represented as a DTP (Dechter, Meiri, and Pearl 1991). The HSP framework allows finite-domain variables and temporal variables to interact through hybrid constraints, each of which is 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 session start and end times. Aspects like the durations of and lag times 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 time period. 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, 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 a HSP session.

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) proposed augmenting the DTP with finite-domains (DTP_{FD}) to allow temporal constraints conditional on the value of finite-domains appended to the temporal variables. They noted that conditional temporal constraints have the property that, at any point in time, one can enforce the temporal constraint with the tightest bound consistent with *all* remaining feasible values. Their *least-commitment approach* enforces this constraint to prune more of the search space. 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 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 the least-commitment approach. The HCT algorithm applies to a more general

set of hybrid constraints than those expressible in a DTP_{FD}, and can be more efficient by reasoning about such constraints once, whereas the least-commitment approach might reason about them exponentially many times. We examine the efficiency of this algorithm analytically and give preliminary results demonstrating that it reduces the expected runtime of search by a significant margin within a particular space of hybrid scheduling problems.

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 is dependent on its topic using a 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 value (topic) for each variable be the same, as shown in Figure 1 (upper). Furthermore, the DTP_{FD} specification constructs a bounds lookup table to hold the value of the temporal bound for *every* combination of finite-domain values, even when such a constraint is unnatural or arbitrary, as denoted by the "?" entries that would need values in the bounds lookup table in Figure 1 (upper). In contrast, in a HSP the hybrid constraints might only apply to some values and the HSP representation is simpler, as shown in Figure 1 (middle), where we have represented disjunctions in their implicative forms.

Our approach is to map the DTP_{FD} least-commitment strategy into the more general HSP framework to solve HSPs more efficiently thanks to the ability to prune the search space based on temporal constraints even before associated finite-domain variables are set. As shown in Figure 1, it is certainly possible to rewrite standard hybrid constraints in a way that makes explicit otherwise implicit constraints (in this case, that any session must take at least an hour). Arguably, we could place the onus 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 involves three basic steps. The first step is to group hybrid constraints based on the structure of the temporal constraint involved. This is done efficiently by creating a simple hash function based on the temporal variables involved and the order in which they appear in the constraint. Figure 1 (middle) 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. Similar constraints could exist for earliest start time, latest end time, lag between events, etc., which may all be dependent on the values of variables like topic, location, and moderator. Once these hybrid constraints have been grouped, the second step is to sort the hybrid constraints based on the values of the bounds they impose on the temporal constraint, as is also shown in Figure 1 (middle). This step can be omitted if the group is stored as a sorted list during construction.

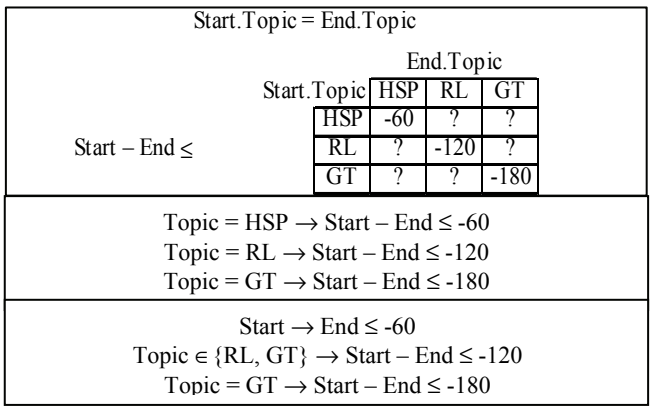

Figure 1: ***Example Hybrid Constraint Transformation:*** *DTP_{FD} representation of a conditional temporal constraint expressing minimum duration based on topic (upper). Same constraints expressed as HSP hybrid constraints (middle). HSP hybrid constraints after applying the HCT algorithm (lower).*

The third step is to construct new hybrid constraints based on the old hybrid constraints, as displayed in Figure 1 (lower). The insight here is that a more restrictive temporal constraint subsumes a less restrictive temporal constraint, allowing us to "lift" information that is inherently common to such sets of constraints so that the search algorithm can exploit it, a concept similar in spirit to constructive disjunction (Muller and Wurtz 1995). These new hybrid constraints are formed by replacing the current finite-domain component of each hybrid constraint with a disjunction of the finite-domain components of all hybrid constraints whose temporal bounds are at least as restrictive. This is done efficiently by traversing the constraints in sorted order. The hybrid constraints corresponding to the tightest temporal constraint in each group remains untouched. The finite-domain component of the hybrid constraint with the second tightest bound is merged (disjuncted) with the finite-domain component of the tightest hybrid constraint of the group to form a new hybrid constraint where the temporal bound is enforced when either of the disjuncted finite-domain constraints is consistent. Since hybrid constraints are examined in sorted order, all finite-domain constraints implying temporal bounds tighter than the current hybrid constraint would have been merged together to form the finite-domain component of the pre-

viously reformulated hybrid constraint. Hence, the current hybrid constraint can be reformulated simply by replacing its finite-domain component with a combination of its old finite-domain and the new finite-domain component of the previous hybrid constraint. Notice that no new hybrid constraints are created in this process.

This reformulation 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. Therefore, no assignments that would have previously been part of a consistent solution would be pruned or inconsistent under this reformulation. Additionally, this reformulation certainly does not allow previously inconsistent assignments, since the HCT algorithm strictly loosens the finite-domain constraint on which the corresponding temporal constraint is conditional. Thus, any previous inconsistencies would certainly remain inconsistent. Our HCT algorithm therefore provides a sound and complete reformulation of the hybrid constraints.

To demonstrate the efficacy of our approach, we reference the example in Figure 1. We must verify our claim that the HCT approach always enforces the tightest allowable temporal bound. As noted, hybrid constraints can be loosely thought of as conditional temporal constraints, with the corresponding finite-domain component representing the "preconditions" for applying the relevant temporal constraints. By combining a finite-domain constraint that implies a particular bound on a temporal constraint with all finite-domain constraints (from hybrid constraints) that imply tighter bounds, we allow the search algorithm to satisfy the preconditions of the corresponding temporal constraint earlier in the search process. By easing the preconditions of the temporal constraints, we are allowing a correct search algorithm to apply the tightest allowable temporal constraint at any given time. Notice in Figure 1 (lower) that this merging creates a "precondition" that is always true since the value of the topic variable will necessarily be an element of its entire domain. This allows us to add the corresponding temporal constraint explicitly, enabling the search algorithm to enforce it immediately. Furthermore, as values are removed from the domains of finite-domain variables due to the consistency maintenance policies of the search algorithm, the tightened hybrid constraints enable the search algorithm to apply the tightest bounds on the relevant temporal constraint as possible. In contrast, the hybrid constraints, as expressed in Figure 1 (middle), would allow the search algorithm to enforce a particular temporal bound only after the finite domain variable involved was assigned a specific value.

The efficacy of our approach relies on the corresponding search algorithm. Whether our approach enforces the tightest possible constraints as early as possible depends on the heuristic decisions made by the search algorithm concerning its consistency maintenance policy. Our approach simply articulates the hybrid constraints in such a way as to make the search algorithm's consistency maintenance as effective as possible when it is applied, essentially achieving a higher level of consistency between two heterogeneous sets of variables.

Analysis of the HCT Algorithm

The first step of our algorithm involves applying a hash function to the temporal component of each hybrid constraint to determine its group. This will take a runtime of $O(H)$, where H is the number of hybrid constraints. The second step of our algorithm involves sorting the hybrid constraints of each of these groups according to the tightness of the bounds they enforce, requiring $O(H \cdot \log(B))$, where B is the size of the largest group. The third step of the algorithm reformulates each hybrid constraint by merging the finite-domain components of all hybrid constraints with less restrictive constraints. The groups are maintained in sorted order, allowing this merge to be accomplished in constant time by merging the recently reformulated finite-domain component of the hybrid constraint with the immediately more restrictive bound, thus requiring a runtime of $O(H)$. Hence, since B is bounded by H, the overall runtime of our preprocessing algorithm is bounded by $O(H \cdot \log(H))$.

The least-commitment approach 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 must 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. Even if such updates could occur in constant time, this approach incurs a worst-case asymptotic overhead time complexity of $O(V^D)$, where V is the number of finite-domain variables, and D maximum finite-domain size. Our approach is applied once, and thus the cost is amortized over the entire run of the search. Additionally, since our approach only requires a unique tightened hybrid constraint for each unique temporal bound, we can take advantage of the often compact representation of finite-domain constraints, as opposed to an explicit enumeration of all possible combinations of finite-domain values present in each bounds table, as seen in Figure 1 (upper).

To empirically evaluate the effectiveness of our HCT algorithm, we compared the performance of a HSP solver with and without our HCT preprocessing algorithm applied to HSPs of the type that arise in application domains that we study. We should note that the HSP solver we used does not implement the DTP_{FD} least-commitment approach; our future plans (see next section) include a more systematic comparison of the DTP_{FD} least-commitment approach and our HCT algorithm. For our experiments, we randomly generated three sets of 500 HSP problems roughly corresponding to larger versions of the AI conference example as specified above. To demonstrate the greater generality of our approach, the latter two problem sets involve hybrid constraints not expressible as a DTP_{FD}. In each problem set, 60-65% of all problems generated

resulted in a feasible solution. The details of the problem generator, as well as the full resulting data, are available upon request. We used Schwartz's Java implementation of a HSP solver, which he graciously lent to us, on a two GHz processor with two GB of RAM to solve each of the problems in all three sets using two approaches: one where we applied our HCT algorithm prior to solving, and one where we did not. All results were determined statistically significant using a Student paired T-test.

A cap of two and a half minutes of search time was enforced. On the set of randomly generated HSPs with structure expressible as a DTP_{FD}, this runtime cap lead to a completion rate of 87.6% for problems that were not preprocessed using our HCT algorithm and 100% for problems to which HCT was first applied. Table 1 shows the results of these runs, truncated to exclude the 12.5% hardest (uncompleted) and the 12.5% easiest problems to solve relative to search time, leaving the middle 75% of the problems. As indicated in Table 1, preprocessing the problems with the HCT algorithm led to a significant speedup in search time (96 times faster), as well as a significant reduction in implementation-independent metrics such as the number of node visits and backtracks. This dramatic improvement was due to the increased pruning that the HCT algorithm induced, preventing assignments that are inconsistent across the hybrid constraints, confirming the value of such least-commitment pruning. On each of the other sets of randomly generated HSPs, the application of our HCT algorithm also led to a mean speed-up in runtime of 50-100 times. The margin of improvement depended on the specific structure of the hybrid constraints involved. 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 nontrivial HSPs in general.

	Without HCT	With HCT	Ratio of Improvement
Time(s)	9.104	0.095	96
Node Visits	11756.3	125.3	93.8
Backtracks	217.42	0.53	411.8

Table 1: *12.5% truncated mean search statistics of 500 randomly generated HSPs*

A direct empirical comparison of our HCT algorithm with the prior least-commitment approach is difficult, since the prior approach requires nontrivial changes to the implementation of the CSP solver used. Such a comparison would be highly sensitive to implementation details specific to each of the least-commitment algorithms, as well as the size and nature of problems being solved. We would like to perform a more thorough, empirical comparison of the HCT algorithm with the prior least-commitment approach utilizing more widely used CSP solvers. The effectiveness of the HCT algorithm as it extends to other interesting HSPs is likely to depend on factors such as the relative number, tightness, and complexity 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 understand more completely the range of HSPs where our HCT algorithm is most effective and generalize this approach to handle any hybrid constraint problem containing variables whose values have inherent ordinal meaning.

Conclusion

We have presented the HCT algorithm, which automates the explicit transformation of hybrid constraints into compact, tighter, and more effective constraints that proactively help to prune the search space. We demonstrated both analytically and empirically that the HCT algorithm applies to, and reduces the expected search time for, problems containing hybrid constraints that are not expressible in a DTP_{FD} representation. In the Analysis section, we showed that we could apply the HCT algorithm separately from the search algorithm amortizing the cost across the entire search. This is in contrast to the least-commitment approach of (Moffit, Peintner, and Pollack 2005), 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.

Acknowledgements

The work reported in this paper was supported, in part, by the National Science Foundation under grant IIS-0534280 and by the Air Force Office of Scientific Research under Contract No. FA9550-07-1-0262. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or United States Air Force. We would like to thank Keith Purrington, Michael Wellman, and the anonymous reviewers for valuable feedback on this work.

References

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.

Muller, T. and Wurtz, J. 1995. Constructive Disjunction in Oz. In *Proc. of Workshop Logische Prog.*, 113-122.

Schwartz, P. 2007. Managing Complex Scheduling Problems with Dynamic and Hybrid Constraints. PhD. Diss., Computer Science and Engin., Univ. of Mich., Ann Arbor.