# Evaluating Hybrid Constraint Tightening for Scheduling Agents

James C. Boerkoel Jr. & Edmund H. Durfee

Computer Science and Engineering

University of Michigan

Ann Arbor, MI 48109

{boerkoel, durfee}@umich.edu

## ABSTRACT

Hybrid Scheduling Problems (HSPs) combine temporal and finite-domain variables via hybrid constraints that dictate that specific bounds on temporal constraints rely on assignments to finite-domain variables. Hybrid constraint tightening (HCT) reformulates hybrid constraints to apply the tightest consistent temporal bound possible, assisting in search space pruning. The contribution of this paper is to empirically evaluate the HCT approach using a state-of-the-art Satisfiability Modulo Theory solver on realistic, interesting problems related to developing scheduling agents to assist people with cognitive impairments. We demonstrate that HCT leads to orders of magnitude reduction of search complexity. The success of HCT is enhanced as we apply HCT to hybrid constraints involving increasing numbers of finite-domain variables and finite-domains with increasing size, as well as hybrid constraints expressing increasing temporal precision. We show that while HCT reduces search complexity for all but the simplest problems, the relative effectiveness is dampened on problems with partially conditional temporal constraints and hybrid constraints with increasing temporal disjunctions. Finally, we present our preliminary investigations that indicate that HCT can assist in increasing communication efficacy in a multiagent setting.

## Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *scheduling*.

## General Terms

Algorithms, Performance, Experimentation

## Keywords

Hybrid Constraint Tightening, Scheduling Agents

## 1. INTRODUCTION

People suffering from cognitive impairments often struggle to live life independently. Recently, work has gone into creating systems that assist such people in completing everyday activities, such as scheduling their days [5]. For example, modeling a scheduling problem as a Simple Temporal Problem (STP) or a more general Disjunctive Temporal Problem (DTP) works well when scheduling a predetermined set of activities [5]. More generally, scheduling a day's activities involves two important

tasks: 1) selection: deciding which activities to do and how to perform them, and 2) scheduling: deciding when to perform such activities. Our goal is to create scheduling agents that act on behalf of people with cognitive impairments by both selecting which activities to do and scheduling when to do them.

Recognizing that the finite-domain Constraint Satisfaction Problem (fd-CSP) formulation handles selection tasks well, and that the DTP framework handles scheduling tasks well, Schwartz combined these approaches into a single framework, the Hybrid Scheduling Problem (HSP) [6]. 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. While studying the suitability of the HSP model for scheduling agents, we developed hybrid constraint tightening (HCT), which reformulates hybrid constraints so as to enhance the ability of the finite-domain variables to inform the temporal variables and *vice versa*, increasing pruning during search [1].

This paper makes several novel contributions to these prior efforts. First, we substantiate our claim that HCT can be used with arbitrary constraint system solvers by incorporating HCT into a state-of-the-art, off-the-shelf solver. Whereas [1] focused on HCT's analytical properties, we demonstrate *empirically* that HCT reduces the actual runtime of search by an order of magnitude while solving realistic, interesting problems. Additionally, we show that as scheduling agents handle conditional temporal constraints that are more general along certain dimensions, this speedup grows. We also show that other generalizations of conditional temporal constraint structures can dampen the benefits of HCT. Finally, we provide discussion about the appropriateness of using HCT and the HSP towards our goal of developing coordinated, autonomous scheduling agents to act on behalf of people with cognitive impairments, and though we have yet to develop complete, efficient strategies to solve multiagent HSPs, we estimate and discuss the impact of HCT on communication overhead.

## 2. A MOTIVATING EXAMPLE

Imagine a scheduling agent given the task of scheduling Bob's appointments during a visit to a healthcare facility. Bob has two appointments, both for routine checkups. Each checkup has associated with it a subset of available, qualified doctors. Dr. Smith can handle the first appointment; Dr. Jones can handle the second appointment, and Dr. Williams can handle either appointment. Bob has allotted 60 minutes to his visit, which should be plenty of time considering each appointment should only take 20 minutes, though the appointments cannot overlap. The order in which the appointments occur does not matter. If he meets with two different doctors, though, Bob likes to give

himself plenty of time to walk to the next doctor's office and get settled (5-15 minutes depending on the doctor).

To model this problem, the scheduling agent must represent various types of information. First, it must capture the choices it has regarding selecting a doctor for each appointment. Since there are a small number of choices here, it makes sense to represent these choices with *finite-domain variables*. The first appointment (variable A1) has a finite-domain containing two elements, Dr. Williams (D1) and Dr. Smith (D2), while the domain of the second appointment (A2) also has two values, Dr. Williams and Dr. Jones (D3). Although none exist in this example, constraints between variables could limit the allowable combinations of values, such as Bob not wanting to see Dr. Williams for both appointments. Problems containing only finite-domain variables and constraints are known as (finite-domain) Constraint Satisfaction Problems (fd-CSPs). Next, the scheduling agent must represent the timing constraints. First, it makes sense to represent each important point in time with timepoint variables; one for the starting and ending time of each appointment (A1.Start, A2.End, etc.). This allows the scheduling agent to express the timing constraints as *temporal differences,* which constrain the difference between two timepoint variables with a *temporal bound*. For example, A1.End − A1.Start $\leq$ 20 and A1.Start − A1.End $\leq$ -20 represent the maximum and minimum durations of appointment A1 respectfully, and A2.End − A1. Start $\leq$ 60 and A1.End − A2.Start $\leq$ 60 enforce the 60 minute overall (makespan) constraint.

The Simple Temporal Problem (STP) is a temporal problem that contains temporal differences as its constraints, and can be solved in polynomial time using an all-pairs-shortest-path-algorithm [4]. However, temporal difference constraints cannot be used to simply prevent appointments from overlapping when the order of the appointments does not matter. This requires a more general disjunctive temporal constraint, which expresses constraints as a disjunction of temporal difference constraints. For example, a general non-overlap constraint would look like A1.End − A2.Start $\leq$ 0 || A2.End − A1.Start $\leq$ 0, or in words, A1 must end before A2 starts or A2 must end before A1 starts. A problem containing these more general temporal constraints is called the Disjunctive Temporal Problem (DTP). DTPs are solved by selecting a temporal difference from each disjunctive temporal constraint to enforce. After a complete selection is made, the resulting component STP can be solved in polynomial time; however, since there are exponentially many such component STPs, solving DTPs in general takes exponential time [4]. Note that though we claimed Bob's non-overlap constraints actually depend on which doctors are selected, we have not accounted for this in our description yet. Before we explain how to model these constraints, we formally introduce the hybrid scheduling problem.

# 3. BACKGROUND

## 3.1 The HSP

DTPs are good for scheduling when to do things, and fd-CSPs are good for deciding what combination of things to do. A Hybrid Scheduling Problem (HSP) [6] combines the full expressive power of fd-CSPs and DTPs 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, each with its own

finite-domain of possible values, $D_F$; and $V_T$, the set of temporal variables each with temporal domain $D_T$, where $D_T$ contains real numbers and is considered implicit. 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. 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 fd-CSP sub-problem <$V_F$, $C_F$> and to the temporal CSP sub-problem <$V_T$, $C_T$>. However, due to $C_H$, arbitrary solutions to these pure subproblems cannot necessarily be combined to solve the overall HSP.

Returning to the example, the lag time between appointments depends on which doctors are selected. We call such constraints *conditional temporal constraints.* We express such constraints with a group of hybrid constraints. Figure 1 (upper) expresses the general conditional temporal constraint enforcing the non-overlap constraint of the problem. Notice that Figure 1 (middle) uses hybrid constraints in their implicative form to demonstrate that the values the temporal bounds (B1 and B2) take depend on the assignments to the finite variables. This particular conditional temporal constraint contains two finite variables (A1 and A2), a disjunction of two temporal differences, and requires four hybrid constraints to express, since each variable has two values, yielding four possible combinations of values.

| A1.End − A2.Start $\leq$ B1 $\vee$ A2.End − A1.Start $\leq$ B2 |
|---|
| A1 = D1 & A2 = D1 → B1 = 0 & B2 = 0 |
| A1 = D2 & A2 = D1 → B1 = -5 & B2 = -10 |
| A1 = D2 & A2 = D3 → B1 = -15 & B2 = -10 |
| A1 = D1 & A2 = D3 → B1 = -15 & B2 = -15 |

| |
|---|
| B1 = 0 & B2 = 0 |
| A1 $\neq$ D1 & A2 $\neq$ D1 → B1 = -5 & B2 = -10 |
| A2 = D3 → B1 = -15 & B2 = -10 |
| A1 = D1 & A2 = D3 → B1 = -15 & B2 = -15 |

**Figure 1. Example conditional temporal constraint (upper) whose bounds are subject to untightened hybrid constraints (middle) or equivalent tightened hybrid constraints (lower).**

## 3.2 The HCT Algorithm

As a precursor to the more general HSP, Moffitt, Peintner, and Pollack [4] described a finite-domain extension to the DTP and 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* augments a search algorithm to apply and prune more of the search space. We realized that similar pruning can be achieved by reformulating the hybrid constraints in an independent preprocessing step called Hybrid Constraint Tightening (HCT), instead of altering the search algorithm to perform similar reasoning but doing so a possibly exponential number of times [1].

Our HCT algorithm groups all hybrid constraints involved in a conditional temporal constraint together and then sorts them based

on their temporal bounds. Figure 1 (middle) shows the sorted group of hybrid constraints relating to our example. The algorithm then visits each hybrid constraint in sequence, starting with the tightest, and reformulates each constraint by replacing its finite-domain constraint with a new finite-domain constraint. The algorithm forms this new finite-domain constraint by merging its current set of allowable assignments with the set of allowable assignments of the tighter hybrid constraints. Figure 1 (lower) demonstrates how HCT would reformulate the constraints from our example. This algorithm runs in polynomial time, and the reformulated constraints result in a sound and complete search. A more complete description and analysis of this algorithm as well as a comparison to previous approaches appears in [1]. By adopting HCT, our scheduling agent can apply the tightest possible temporal bounds for the conditional temporal variable at all times, which our previous analysis suggests can help significantly reduce the temporal and, in turn, finite search spaces.

## 4. EMPIRICAL METHODOLOGY

The purpose of our empirical analysis is to look beyond abstract time-complexity analyses to more fully understand the space of HSPs where HCT is particularly effective, as well as to test our claim that HCT can be incorporated into off-the-shelf solvers. Thus, we present a problem generation technique to give us realistic, interesting problems, introduce HCT as a preprocessing step to a state-of-the-art SMT solver, and track performance metrics that illuminate HCT's effect on search complexity.

### 4.1 Problem Generation

Since HSPs, and hybrid constraints in particular, are largely the result of merging components from fd-CSPs and DTPs, aspects that influence the solution complexity of solving fd-CSPs and DTPs will clearly influence the solution complexity of HSPs. Before we discuss how we generate realistic, interesting scheduling problems, we first review the parameters known to affect solution complexity in generating interesting fd-CSPs and DTPs. First, fd-CSP generators [8] often use a variation of the parameters $<N_{FD}, m_{FD}, k_{FD}, d_{FD}, p_{FD}>$, where $N_{FD}$ is the number of variables, $m_{FD}$ is the number of constraints, $k_{FD}$ is the arity of each constraint, $d_{FD}$ is the maximum domain size, and $p_{FD}$ is the density of allowable tuples for a particular constraint. Canonical DTP generators [7] use the parameters $<N_T, m_T, k_T, L_T>$, where $N_T$ is the number of timepoints, $m_T$ is the number of constraints, $k_T$ is the number of disjuncts per constraint, and where all bounds are chosen uniformly between $[-L_T, L_T]$. Both types of generators have parameters dictating the number of variables ($N_{FD}, N_T$) the number of constraints ($m_{FD}, m_T$), as well as the structure and "tightness" of the constraints ($k_{FD}, d_{FD}, p_{FD}, k_T, L_T$). Exploring HCT's effect on the entire range of possible hybrid constraint structures, then, involves investigating the effect of varying each parameter in the union of these two problem generators types.

We adopt a strategy for generating difficult scheduling problems containing both finite-domain and temporal components from [4], which we adapt to loosely match our example. The generator takes parameters $<A, L, B, p_L, p_O, C_{MAX}, p_H, p_{FD}, k_{FD}, k_T>$, where $A$ is the number of appointments the scheduling agent must schedule, $L$ is the number of locations (with one doctor per location), $B$ is a bounds matrix indicating a more general lag time, ranging from 5 to 40 (including extra time for paper work, recovery period from stress tests, etc) between each pair of

locations, $p_L$ is the probability that the doctor at a particular location is able to perform the appointment, $p_O$ is the probability that a non-overlap constraint is enforced between any pair of appointments (e.g., appointments require the presence of a shared resident nurse), $C_{MAX}$ is the maximum allowable makespan of the schedule, or the total length of time the patients will be visiting the medical campus, $p_H$ dictates the relative constraint composition of the problem (hybrid vs. non-hybrid constraints), and finally $p_{FD}$, $k_{FD}$, and $k_T$, all correspond directly to their original semantics. Feasible and nontrivial problems are guaranteed by calculating the minimum possible makespan for each parameter setting.

These parameters relate directly to those of the fd-CSP and DTP generators. First, $A$ encompasses both $N_{FD}$ and $N_T$, since each activity has two timepoints (start and end) and one finite-domain variable (location) associated with it. Thus, $N_{FD}$ and $N_T$ grow linearly as $A$ grows. Next, $L$ represents $d_{FD}$, the maximum size of domain in fd-CSPs, with $p_L$ probabilistically dictating how large each individual finite-domain is. $C_{MAX}$ correlates with $L_T$, with $B$ restricting bounds to realistic times. Finally, $p_O$ probabilistically determines the number of constraints in the problem, thus capturing both $m_{FD}$ and $m_T$. Later, we will discuss how we adapt the generator to capture the effect that HCT has on the structural changes of hybrid constraints implied by $p_H, p_{FD}, k_{FD}$, and $k_T$.

We generate 50 problems for each setting of the parameters, varying parameters as described in Section 5. Any parameter that we do not vary in a given experiment is set to its following default value: $A=10$, $L=6$, $p_L=0.33$, $p_O=0.9$, $k_{FD}=2$, $k_T=2$, $p_{FD}=1.0$, $p_H=1.0$, $B[i][i] = 0$ and $B[i][j]$ is selected uniformly from $[5,40]$ when $i \neq j$, and $C_{MAX}$ is set as described above.

### 4.2 SMT Solvers

As noted in [6], Satisfiability Modulo Theory (SMT) technology represents the state-of-the-art in solving HSPs. SMT solvers generalize powerful boolean satisfiability (SAT) solving techniques with additional first-order theories. Z3 is a highly competitive SMT solver published by Microsoft Research, and widely available for download [2]. For our experimentation, we use Z3 version 1.2 on a Windows XP machine with 2.0 GHz processor and 2.0 GB of RAM. Since there is an element of randomness associated with the Z3 solver, we average the results for each problem across 20 uniquely seeded runs.

### 4.3 Performance Metrics

The SMT solver we use reports three statistics relating to search complexity. A *decision* is made any time the solver must choose a literal in a disjunctive clause, similar to a variable assignment in more traditional fd-CSP nomenclature. A *conflict* occurs any time the solver encounters an empty clause that leads to backtracking, similar to a variable with an empty domain. Finally, the solver also reports the amount of *time* that is required before a consistent assignment has been found. Since we are most interested in discovering the space of HSPs for which HCT is most effective, we will often report s*peedup,* or the ratio of the number of decisions, conflicts, or time using HCT to the corresponding metric without HCT. All speedups have been found to be statistically significant using a students paired T-test, unless otherwise noted.

## 5. EMIRICAL RESULTS

We explore the efficacy of HCT along each of the dimensions mentioned in Section 4.1. First, we look at how HCT changes search performance as we scale the problem size and difficulty. Next we show that the margin of improvement of HCT varies as the structure of the conditional temporal constraints varies. We examine four generalizations of this structure for which relative HCT performance improves. Finally, we examine two generalizations of conditional temporal constraint structure for which applying HCT becomes less beneficial. It is important to note that we easily succeeded in overlaying HCT on Z3. This was done by HCT performing the reformulation on the original HSP, and then translating it to an SMT problem, using a process described in [6].

## 5.1 The Efficacy of HCT on Problems with Simple Hybrid Constraints

Given our choice to model our scheduling problem as a HSP, it is important to show that HCT is effective as the number of activities ($A$) and constraints ($p_O$) increases. We examine scaling these two dimensions in the following subsections.

### 5.1.1 Increasing the Number of Activities

The exponential nature of search in the HSP means it is critical that any approach to improve the effectiveness of search scales well as the number of activities ($A$) increases. In this section, we increase the number of activities involved in the problem, while allowing the number of non-overlap constraints to grow proportionally based on $p_O$ as described in Section 4.1. This scaling is similar to imagining that instead of scheduling just two doctor appointments for Bob, a scheduling agent has been handed the task of scheduling appointments for an increasing number of people.
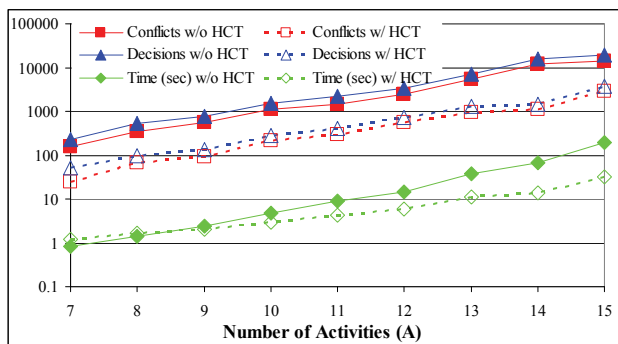


**Figure 2. Logarithmic Scale of the Number of Conflicts, Decisions, and Seconds as Agents Handle Additional Activities**

Figure 2 shows how the median number of conflicts, decisions, and time required for search with and without HCT grows, using a log scale on problems with 7 to 15 activities. Overall, HCT yields an improvement of nearly an order of magnitude, regardless of the size of the problem. However, the difference in time for smaller problems is much less significant, and in fact, there is a visible crossover point between 8 and 9 activities. Thus, for more than half of the problems with fewer than 9 activities, the complexity savings yielded by HCT does not overcome the overhead of performing the HCT preprocessing algorithm. While we mostly care about relative performance, Figure 2 reports median,

absolute statistics to inform the reader of the general complexity of the problem space and demonstrates that HCT yields an improvement both in expectation and for the majority (median) of the problems. All subsequent figures will report only relative performance (in terms of speedup).

### 5.1.2 Increasing the Number of Constraints

Although our strategy for setting $C_{MAX}$ already guarantees the tightest feasible makespan, the burden on search increases as we increase $p_O$. It is easy for the scheduler to find a feasible schedule for a problem with no overlap constraints: simply schedule everything at the same time. However, by adding a non-overlap constraint, the solver may have to search over the possible orders in which the appointments might occur before finding an order that respects the minimal feasible $C_{MAX}$, since the lag time from one doctor to another is not necessarily reflexive. Adding non-overlap constraints increases the possible appointment orderings that must be searched before finding an ordering that "fits".

Figure 3, which varies $p_O$ while holding the number of activities steady, is similar to Figure 2 in many qualitative ways. There is an initial period where the overall runtime using the HCT fails to significantly beat the runtime of search without HCT. Further, it is obvious that sufficiently many constraints are needed before HCT yields improvements in time efficiency that match the significance we see in the other statistics, though, overall, HCT scales well with the number of constraints.
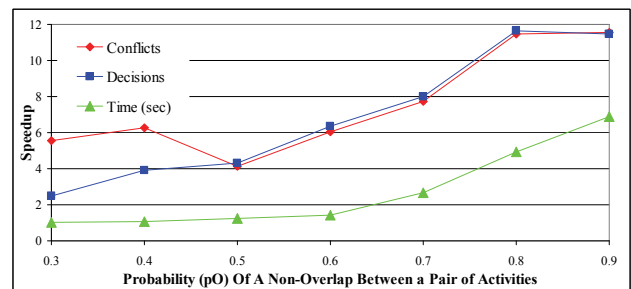


**Figure 3. Agents Handling More Constraints**

## 5.2 Conditional Temporal Constraint Generalizations That Increase HCT Efficacy

In this section, we examine generalizations of the conditional temporal constraint representation by varying the settings of parameters $p_H$, $d_{FD}$, $k_{FD}$, and $C_{MAX}$. These generalizations all have one thing in common: scheduling agents using HCT enjoy an increased speedup over scheduling agents not using HCT. This is critical because realistic, interesting problems tend to push the boundaries of problem representations in these ways.

### 5.2.1 Increasing Temporal Constraint Precision

Consider the example conditional temporal constraint in Figure 1. A knowledge engineer could replace B1 and B2 with their respective minimum possible values (-15 for both) instead of allowing them to be conditional on finite-domain variables. The result is that all these conditional temporal constraints could be approximated with more restrictive, non-conditional temporal constraints. Such a decision would come at the cost of temporal constraints that imply tighter bounds than needed, resulting in schedules with unnecessary slack time, and more alarmingly,

resulting in an algorithm that is no longer complete (with respect to the original, more general constraints). For example, by always spending 15 minutes between appointments, the scheduling agent would have not been able to find Bob a feasible schedule if he only had 45 minutes even though one exists using conditional temporal constraints. To test how HCT performs on these problems, we duplicate the experimentation of Section 5.1.2. However, we now hold $p_O$ constant while we vary $p_H$, which is the probability that a non-overlap constraint is expressed using a conditional temporal constraint instead of the corresponding non-conditional temporal constraint.

Figure 4 shows that speedup increases with $p_H$ across the statistics, although the benefit does not overcome the preprocessing overhead until $p_H > 0.2$. Beyond the results in Figure 4, note that, though it may seem that replacing normal temporal constraints with their hybrid counter-parts would lead to a more complex search, solving the problem containing only hybrid non-overlap constraints using HCT ($p_H$=1.0) finds a solution over 30 times faster than it takes the same solver to find a solution using only temporal constraints ($p_H$=0.0). Furthermore, when $p_H$=1.0 search is complete and does not force unnecessary slack into the schedule.
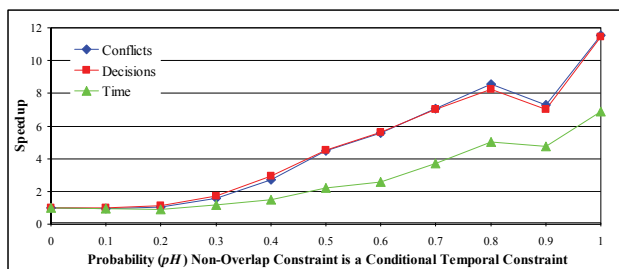


**Figure 4. Increasing Temporal Constraints Precision**

### 5.2.2 Increasing Finite-Domain Size

Our motivating example involved a scheduling agent selecting between two doctors for each appointment, and as a result, the conditional temporal constraint enforcing non-overlap contained four hybrid constraints. Similarly, our problem generator constructs finite-domains containing, in expectation, two values. However, more generally, a scheduling agent must handle temporal constraints conditioned on finite-domains even as the number of values per domain ($d_{FD}$) increases. If the domains of A1 and A2 from the example problem both contained only a single doctor, the conditional temporal constraint in Figure 1 would have been expressed using a single hybrid constraint, since only one consistent finite-domain assignment would exist. Thus B1 and B2 could simply be replaced with the bounds of the sole hybrid constraint, eliminating the need for HCT. So for HCT to have any effect at all, an agent must be able to select values for at least some of the finite-domain variables that influence the bounds on corresponding temporal constraints. As the finite-domains that a temporal constraint is conditioned on grow, more hybrid constraints are needed to express the overarching conditional temporal constraint, one for each possible combination of value assignments. For example, suppose A1 was a more routine physical, and thus could be performed by any of 8 doctors. The conditional temporal constraint would now require 16 hybrid constraints, instead of four like in Figure 1 (middle). We explore the impact of HCT when increasing the number of

hybrid constraints used to express a conditional temporal constraint.

HCT works by lifting information common to multiple finite-domain variable/value assignments involved in a particular conditional temporal constraint, and using that information to assist in pruning. Since the overall search space grows exponentially as the size of the finite-domains grow, we hypothesize that any information that can help pruning may become increasingly valuable. We test this hypothesis by varying the domain sizes of the finite-domain variables in our current doctor appointment scheduling domain. Since previously the size of variable domains were probabilistically determined (each doctor can service each appointment with a given probability), we altered the formulation slightly, replacing $p_L$ with a new parameter $n_L$, which dictates the number of doctors who can service the appointment (size of each finite-domain), with doctors being chosen randomly from the pool of doctors with uniform probability. Since we had been using a $p_L$ value of 0.33, we emulate this by updating L to be equal to $3*n_L$ for each problem.

As shown in Figure 5, HCT grows in effectiveness as the number of values per finite-domain variable increases. While we see immediate benefit in the speedup of decisions and conflicts, it takes problems with at least 4 values in each domain before this reduction in complexity compensates for the preprocessing overhead. This confirms our hypothesis that HCT works best when temporal bounds are dependent on many values, each specified as a hybrid constraint, allowing search to prune using the tightest consistent value.
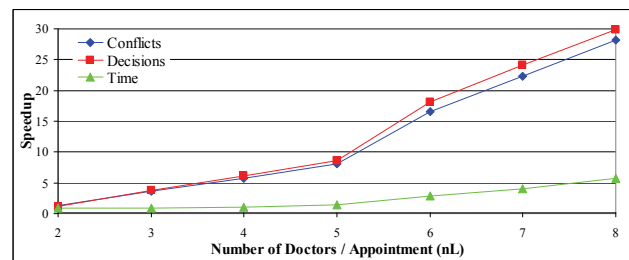


**Figure 5. Increasing Finite Domain Size**

### 5.2.3 Increasing Finite Variables

Section 5.2.2 demonstrates that, as the number of values per finite-domain variable involved in a conditional temporal constraint increases, HCT becomes increasingly important for efficiently finding a schedule. However, instead of increasing the number of *values* per finite-domain, we could also generalize conditional temporal constraints by increasing the number of finite *variables* on which they depend. This is achieved by increasing the finite-domain arity ($k_{FD}$) of the hybrid constraints. The conditional temporal constraint of our original example (Figure 1) represented a non-overlap constraint whose bounds depended on two finite-domain variables: the doctors selected for each appointment. However, more generally, the lag time required between appointments could depend on any number of variables (e.g., also paperwork requirements, recovery time, etc.). Such a change would require hybrid constraints whose finite-domain component contains either fewer (or more) variables.

As Section 5.2.2 demonstrates, increasing the number of hybrid constraints used to express a conditional temporal constraint (due

to larger finite-domains) increases the efficacy of HCT. So to avoid conflating these results, we ensure that each conditional temporal constraint depends on a constant number of hybrid constraints by using the $n_L$ parameter to adjust the number of values in the domain of each finite-domain variable so that the total number of possible assignments is held constant (e.g., one variable with 9 elements in its domain is replaced with two variables, each with 3 elements in their domains). As discussed earlier, HCT excels at applying the tightest consistent bound possible regardless of the number of remaining domain values. So, whereas an untightened constraint requires a full assignment to every involved finite-domain variable, tightened hybrid constraints can successfully leverage knowledge from partial assignments to tighten the bounds on the respective temporal constraint. Therefore, it is our expectation that HCT will be more effective as temporal constraints become conditioned on an increasing number of variables.

Figure 6 confirms our expectation that HCT's efficacy increases as the number of variables increases. This result is important since, combined with 5.2.2, knowledge engineers can have less fear of overwhelming scheduling agents by giving them more control over deciding aspects of activities that impact the schedule. Notice that speed-ups, reported using a logarithmic scale, are much higher in general than occurred in the experiments seen so far. This is because temporal constraints were conditioned on many more finite-domain assignments, as well as fewer overall activities, and this magnifies the effect seen in 5.2.2.
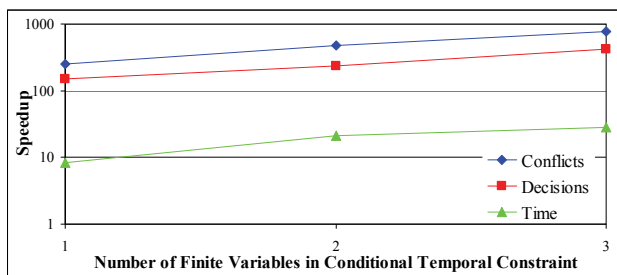


**Figure 6. Increasing the Number of Finite-domain Variables**

### 5.2.4 Finer Granularity of Temporal Bounds

Time is inherently continuous, and the DTP exploits this for greater scheduling flexibility while mitigating the potential exponential blowup often associated with dealing with an infinitely large domain. Since the DTP, and thus implicitly the HSP as well, remains largely agnostic to the granularity of temporal bounds, it is worth investigating if the HCT process also remains unaffected. In the extreme, imagine that all the temporal bounds in Figure 1 (middle) were the same value. Here, HCT would reformulate the hybrid constraint into a single hybrid constraint, and thus a single, inevitable non-conditional temporal constraint. Though search would be able to exploit this constraint immediately, no further opportunities would exist for tightening the bounds. In contrast, when each of the temporal bounds is unique, these bounds provide significant tightening opportunities as search progresses. In the experiments so far, all temporal bounds were chosen uniformly from integers between 5 and 40 (35 different bounds). We test how HCT performs as we change the number of allowable values to choose from, ranging from 1 to 35, corresponding to varying levels of granularity. We do this by

selecting, for example, two integers uniformly from [5,40], and then assigning temporal bounds randomly from this pair of values.

Figure 7 shows that, in fact, as scheduling agents handle temporal constraints expressed using an increasing number of unique bounds, the speedup achieved by applying HCT also grows. This is good news for scheduling agents; HCT mitigates computational concerns over expressing temporal constraints as precisely as possible, without concern for the granularity at which temporal bounds are chosen. Once again, we see that the importance of HCT grows as we give scheduling agents increasingly general conditional temporal constraints. HCT assists search in exploiting even subtle differences in temporal bounds.
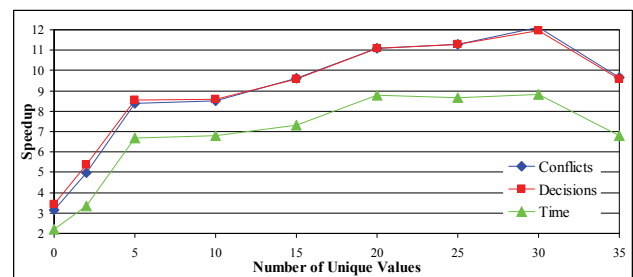


**Figure 7. Finer Granularity of Temporal Bounds**

## 5.3 Conditional Temporal Constraint Generalizations That Decrease HCT Efficacy

Unfortunately, while we have shown HCT efficacy grows with increasing generality along dimensions like the number of variables and values they are conditioned on, HCT efficacy does not grow with increasingly general hybrid constraints across all dimensions. Here we examine two such generalizations, involving $p_{FD}$ and $k_T$, that dampen HCT's ability to assist in pruning the search space. It is important to note that while certain conditional temporal constraint structures decrease the relative benefits of HCT, the only time applying HCT is actually detrimental is when the time cost associated with the applying the HCT algorithm does not outweigh the benefits of the HCT reformulation of constraints on sufficiently easy problems.

### 5.3.1 Partially Conditional Temporal Constraints

In Figure 1 (middle), every possible finite-domain assignment combination is involved in a hybrid constraint and implies a unique set of temporal bounds. However, in general it is not required that every assignment has to imply a temporal bound. This would correspond to Figure 1 (middle) where one or more of the hybrid constraints were eliminated. For example, imagine that the constraint in Figure 1 was between appointments belonging to two different patients. Then, we might only care about not overlapping when both appointments are with the same doctor. In this case, all but the first row of Figure 1 (middle) could be eliminated. A hybrid constraint thus not only expresses *which* temporal bound to enforce, but also, by its presence, *if* a temporal bound should be enforced at all. We call this generalization a *partially* conditional temporal constraint. To investigate the effect that these more general partially conditional temporal constraints have on HCT efficacy, we vary the parameter $p_{FD}$, which is the probability that a particular assignment of values to the finite-domain variables will imply a temporal constraint in a given conditional temporal constraint.

Decreasing $p_{FD}$ is likely to change the effectiveness of HCT. When $p_{FD} = 1.0$, tightened hybrid constraints can immediately apply the tightest consistent temporal constraint with the remaining values of the finite-domain constraints to assist in pruning the temporal space, with the bounds tightening as the domains of the finite-domain variables are reduced. With $p_{FD} <$ 1.0, the tightened hybrid constraints will still apply the tightest consistent temporal constraint; however now the tightest *consistent* constraint may, in fact, be no constraint at all. Furthermore, HCT is now lifting information from a smaller set of value combinations, which, shown in 5.2.2, decreases its efficacy.

Figure 8 confirms that as $p_{FD}$ decreases, so does the relative effectiveness of HCT. Partial specification of temporal bounds dampens the ability of HCT to lift and use information as successfully during search. Not only does the relative performance degrade as $p_{FD}$ decreases, there is actually a span of $p_{FD}$ where the runtime overhead of performing and using HCT exceeds the reduction in decisions and conflicts. Scheduling agents must use HCT judiciously when their problems contain conditional temporal constraints where the majority of the value combinations do not imply temporal constraints.
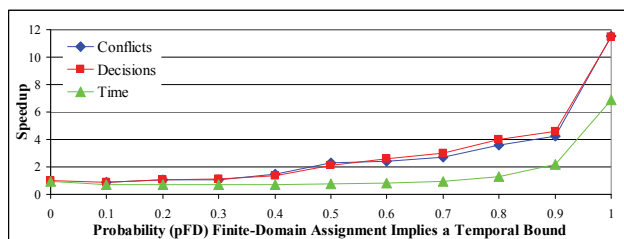


**Figure 8. Partially Conditional Temporal Constraints**

### 5.3.2 Increasing Temporal Disjunctions

In Figure 1 (upper) we showed that a non-overlap, conditional temporal constraint is expressed by using two temporal disjunctions whose temporal bounds were both conditioned on the assignment of values to finite variables. However, a conditional temporal constraint, just like a normal temporal constraint, is not limited in the number of temporal differences it can involve in its disjunction, and can have any temporal arity, $k_T$. In fact, it is completely realistic to assume that one doctor appointment depends on the results of another. For example, if A1 must occur before A2, Figure 1 (upper) could be replaced with a temporal constraint looking like A1.End – A2.Start $\leq$ B1 (where we remove the disjunction). Thus, we generalize conditional temporal constraints by generalizing the structure of the temporal constraint itself, varying the number of temporal disjuncts involved.

We first alter our example to contain constraints with a single temporal difference by replacing the current non-overlap constraints with non-overlap constraints where the relative order of appointments has already been determined as described above. Although conditional temporal constraints in problems we are interested in tend to contain disjunctions of two or fewer temporal differences, we can also alter the problem to contain conditional temporal constraint with three or more temporal disjunctions. Assume that doctors have two shifts, with a 60 minute break for lunch. The doctors may be required to change rooms for their afternoon session, thus travel times between doctors on campus may change after lunch. In this case, there would be two sets of non-overlap constraints, one for the morning and one for the afternoon, each containing disjunctions among four temporal differences. To capture this situation, a "morning" hybrid constraint would contain a disjunction of four temporal differences: either the appointments A1 and A2 do not overlap (as expressed in Figure 1 (top)) or one of the activities occurs in the afternoon (Noon – A1.Start $\leq$ 0 $\vee$ Noon – A2.Start $\leq$ 0). A similar constraint is introduced to enforce afternoon constraints.

The number of temporal differences involved in a hybrid constraint has a direct effect on the runtime of the HCT algorithm, since the tightest bound must be discovered for each temporal difference, causing a linear increase in runtime. We expect that as the number of temporal disjuncts involved in hybrid constraints grows, the effectiveness of HCT will diminish, because a tightened hybrid constraint will have a less immediate impact. For example, if a scheduling agent knows ahead of time that one appointment must end at least five minutes before another appointment starts, it can take advantage of this information to immediately prune the search space. In the case that the ordering of appointments is irrelevant, the agent then knows either the first appointment ends at least 5 minutes before the start of the second, *or* the second appointment ends 5 minutes before the start of the first. At this point, the scheduling agent must decide on an ordering *before* it can apply the knowledge gleaned from HCT.

Figure 9 confirms that HCT prunes the search space more effectively on problems containing hybrid constraints with only a single temporal disjunction, with the speedup of conflicts and decisions decreasing nearly identically. Surprisingly, the runtime speedup actually increases slightly with increases in the number of temporal disjuncts. This is partially an artifact of overall search complexity. Problems with only a single temporal disjunct require search over only the finite-domain variables, while problems containing more temporal disjuncts require search over both finite-domain and temporal meta-variables. Thus, problems containing only a single temporal disjunct are solved much more quickly, regardless of whether HCT is used or not, and so we see a bit of a horizon effect where the problems are solved so quickly that the HCT overhead prevents the time speedup from reaching speedup levels similar to those seen in the number of decisions.
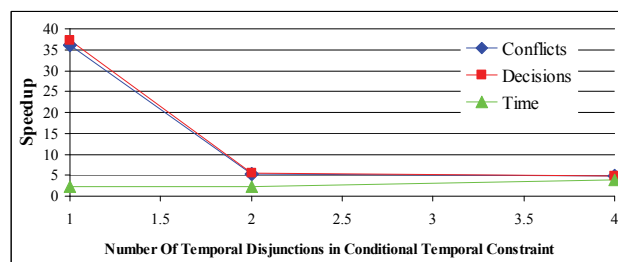


**Figure 9. Increasing Temporal Disjunctions**

## 6. DISCUSSION

This paper takes significant steps forward in our understanding of HSP solution methods, and in particular about using HCT preprocessing. Validating that HCT can be applied easily to off-the-shelf search algorithms expands the applicability of HCT to a wider variety of cutting-edge solvers. Our systematic examination of performance over the space of HSPs, especially the structure of hybrid constraints, leads to a more complete understanding of when HCT can be applied most successfully.

HCT scales well with the number of activities and constraints, and is only detrimental when problems are too easy, leading to preprocessing costs that exceed the gains. We showed that scheduling agents can use HCT to solve problems containing conditional temporal constraints with increasing finite-domain variable choices and bounds expressed at finer temporal granularity more efficiently. Although we noted that HCT becomes less effective when applied to partially conditional temporal constraints and conditional temporal constraints with increasingly large temporal disjunctions, the upside is that using HCT sill improves the temporal complexity on HSPs with these constraints. Furthermore, challenging problems generally contain a rich mixture of hybrid constraint structures, leading to an order (or greater) of magnitude reduction in solver runtime thanks to HCT.

Our next steps are to use the HSP for scheduling problems that are distributed across multiple scheduling agents, each autonomously scheduling on behalf of a person with possible cognitive disabilities. This paper highlights the effectiveness of using HCT to prune the search space to speed performance. However, this kind of forward checking could be costly when a HSP is distributed across multiple agents, because to perform effective pruning can require significant communication about partial assignments. Figure 10 shows our empirically generated estimates of the impact of using HCT on communication based on a centralized, sequential search using a primitive solver (that we could instrument for this purpose) lent to us by Schwartz [6]. We estimate the upper-bound on the frequency of communication as the number of agents increases by tallying every time a change is made in one agent that could potentially affect another agent, while we estimate the lower bound by optimistically assuming that all communication between a pair of agents during a decision cycle could be aggregated into a single message.

Prior to search, the increased pruning does in fact lead to a much higher upper-bound on the amount of communication when HCT is used. However, once search commences, the overall communication is reduced significantly, due to less backtracking thanks to HCT's forward-checking. The lower-bound estimate, which assumes all preprocessing communication can be aggregated into a single message, highlights the potential for HCT to reduce communication. These preliminary results suggest that combining the additional pruning of HCT with communication aggregation can lead to a more efficient multiagent search.

Furthermore, we conjecture that HSPs can exploit other strategies to increase parallelism and reduce communication. Hunsberger, in solving STPs [3], eliminates the need for communication during search by adding additional constraints to decouple the problem. Although additional constraints could lead to an incomplete search method, which we would want to avoid, we do think that using techniques similar to Hunsberger's can augment HCT reasoning. That is, while tightening hybrid constraints, agents could negotiate "communication thresholds". These thresholds would represent how far one agent can tighten a common temporal constraint before another agent is likely to care about it. This is important when using HCT, because not every tightening of a bound is likely to affect other agents. This raises the possibility that two agents could both establish partial assignments that are inconsistent with each other. In these scenarios, we hope to use strategies similar to those in [9] to make adjustments local to an agent to resolve the issue, while

maintaining a sound, complete search. Once again, we believe HCT will be critical in making local adjustments, since if the conflict is due to a hybrid constraint, HCT could assist in recognizing similar hybrid constraints that will also cause problems if the correct adjustments are not made.
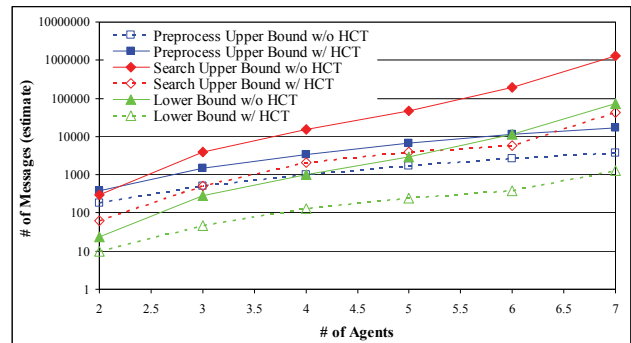


**Figure 10. Estimated Number of Messages Passed**

# 7. ACKNOWLEDGEMENTS

# 8. REFERENCES

[1] Boerkoel, J. and Durfee, E. 2008. Hybrid Constraint Tightening for Hybrid Constraint Scheduling. In *Proc. Of AAAI-2008*, 1446-1449.

[2] de Moura, L. and Bjørner, N. 2008. Z3: An efficient SMT solver. In *Proc. Of TACACS-2008*, 337-340.

[3] Hunsberger, L. 2003. Distributing the Control of a Temporal Network among Multiple Agents. In *Proc. Of AAMAS-2003*, 899-906.

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

[5] Myers, K. Berry, P. Blythe, J. Conleyn, K. Gervasio, M. McGuinness, D. Morley, D. Pfeffer, A. Pollack, M. and Tambe, M. An intelligent personal assistant for task and time management. In AI Magazine, 2007.

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

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

[8] Xu, K, Boussemart, F. Hemery, F. and Lecoutre, C. 2007. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence 171*: 514-534.

[9] Yokoo, M. and Hirayama, K. 2000. Algorithms for distributed constraint satisfaction: A review. In *Proc. Of AAMAS-2000*, 198-212.