A Fast Incremental Algorithm for Maintaining Dispatchability of Partially Controllable Plans

Deepak Kumar, Divanshu Kaushik
Department of Information Technology
Dronacharya college of engineering, Gurgaon , Haryana, India

Abstract- **Compilers are perceived to be magical artifacts, carefully crafted by the wizards, and unfathomable by the mere mortals. Books on compilers are better described as wizard-talk: written by and for a clique of all-knowing practitioners. Real-life compilers are too complex to serve as an educational tool. And the gap between real-life compilers and the educational toy compilers is too wide. The novice compiler writer stands puzzled facing an impenetrable barrier, "better write an interpreter instead." Compilers are perceived to be magical artifacts, carefully crafted by the wizards, and unfathomable by the mere mortals. Books on compilers are better described as wizard-talk: written by and for a clique of all-knowing practitioners. Real-life compilers are too complex to serve as an educational tool. And the gap between real-life compilers and the educational toy compilers is too wide. The novice compiler writer stands puzzled facing an impenetrable barrier, "better write an interpreter instead." The development of the compiler is described in detail in an extended tutorial. Supporting material for the tutorial such as an automated testing facility coupled with a comprehension.**

I. INTRODUCTION

Often, autonomous agents that operate in real-world
environments must be able to plan, schedule, and execute
missions while robustly anticipating and adapting to
uncertainty and disturbances. Typically an agent only
controls the timing of a subset of a plan's events; timing of
the other events is controlled exogenously by nature or
other agents. For example, a Mars rover can control when
it starts driving to a rock; however, its precise arrival time
is influenced by environmental factors. To achieve
successful execution of a partially controllable plan, the
scheduler must guarantee that all temporal constraints are
satisfied, even though some events are uncontrollable.
Since it is difficult to provide such a guarantee without any
knowledge about the behavior of uncontrollable events, the
scheduler exploits a model, called a simple temporal
network with uncertainty (STNU) [Vidal 1996, Vidal and
Fargier 1999], to explicitly represent plan uncertainty by
bounding the behavior of uncontrollable events.
The domain of application for STNUs is embedded
systems, such as airplanes and robotic systems, which
perform scheduling within their controller. The field ofSTNU dispatching focuses on embedded control
applications in which the scheduler must satisfy hard
scheduling constraints while accommodating disturbances.
Applications include scheduling within the avionics
processor of commercial aircraft [Tsamardinos et al. 1998],

and control of space probes [Muscettola et al. 1998b], autonomous air vehicles [Stedl 2004], and walking robots [Hofmann et al. 2006].

For a given STNU, it may not be possible to generate a static schedule a priori that guarantees successful plan execution over all possible execution times of uncontrollable events. If a static schedule does exist, it may be overly conservative in plan completion time. This problem is addressed through dynamic control [Vidal 2000], a strategy that schedules controllable events online just before they are executed. This strategy exploits the fact that uncertainty associated with past uncontrollable events is eliminated, allowing the scheduler to be less conservative in the schedules it generates. Given a set of temporal constraints over controllable and uncontrollable events, and observations of past events, a dynamic control strategy generates a schedule online that guarantees the temporal constraints of the plan are satisfied.

Dynamic control is achieved through dispatchable execution [Muscettola 1998, Morris et al. 2000], the incremental computation of feasible schedules performed through constraint propagation, to update the network as new information is received. The timing of executed events is propagated throughout the network to ensure that time windows of later events are appropriately narrowed to satisfy timing constraints of the plan. Dynamic control of STNUs, achieved through dispatchable execution, is domain independent, and applicable to online scheduling for systems that have uncontrollable events and must satisfy hard scheduling constraints.

To achieve the goal of scheduling STNUs in real-time, [Morris et al. 2001] introduced a dynamic controllability (DC) algorithm to 1) determine if a dynamic control strategy exists for an STNU, and if so, 2) compile the STNU to a dispatchable form which reduces the amount of propagation necessary during execution, making it possible to schedule in real-time. The dispatchable plan is precompiled before plan execution, and is then used by a dispatcher to schedule quickly online.

This paper focuses on the additional technical challenge of responding, in real-time, to disturbances that require a mission phase, the agent may need a way to quickly replan and then recompile the modified plan into a dispatchable form. Significant progress has been made on the first problem – fast replanning. Efficient solutions include the use of local repair [Zweben 1993, Rabideau et al. 1999], and incremental search [Shu 2003, Effinger 2006]. However, existing compilation algorithms are insufficient for real-time performance.

We confront the challenge of real-time compilation based on the observation that during replanning, typically only a small portion of a plan is modified. Our compilation algorithm improves efficiency substantially by

incrementally updating the dispatchable plan in response to
plan changes in the spirit of other incremental algorithms
for truth maintenance [Doyle 1979] and informed search
[Koenig et al. 2001]. Current DC compilation algorithms
repeatedly compute an all-pairs shortest path (APSP) graph
and then check all possible triangles in the network for
reductions. In contrast, our IDC algorithm maintains
dispatchability as constraints in the plan are tightened (or
added) and loosened (or removed). This is achieved
through a set of incremental update rules that exploit the
causal structure of the plan to efficiently propagate the
effect of each changed constraint throughout the network.

This paper presents our incremental compilation
algorithm and its empirical validation. First, we describe a
practical scenario involving the coordination of rovers as
an example for the rest of the paper. Next we review
STNUs and the DC algorithm. We then develop our
incremental algorithm in two parts: we present how to
maintain dispatchability for the case when a constraint is
tightened or added to the plan, and then for the case where
constraints are loosened or removed from the plan. Finally,
we present empirical results comparing the incremental
algorithm to the DC algorithm and conclude.

II. PRACTICAL SCENARIO

Consider a two rover scenario used to demonstrate online replanning, DC compilation and dispatching on a hardware
test-bed [Robertson and Williams 2005] (Fig.1). In this
scenario, the two rovers cooperatively search for science
targets in a simulated Martian environment. Rover 1
drives to location 3, via a route through location 1. At
location 3, Rover 1 surveys the area for interesting science
targets. Simultaneously, Rover 2 drives to location 4, via a
route through location 2. At location 4, Rover 2 surveys the
area for interesting science targets. When both rovers
finish surveying their respective areas, they rendezvous at
the same time at location 0. The plan corresponding to this
description involves uncontrollable events; for example,
the time the rovers spend “finding targets” is uncertain
since the rovers may find a target right away or use the
maximum allotted time without finding a target.

III. BACKGROUND

A Simple Temporal Network with Uncertainty (STNU)
[Vidal and Fargier 1999] is an extension of an STN
[Dechter 1991] that distinguishes between controllable and
uncontrollable events. An STNU is a directed graph,
consisting of a set of nodes, representing timepoints, and a
set of edges, called links, constraining the duration between
the timepoints. The links fall into two categories:
requirement links and contingent links. A requirement link
specifies a constraint on the duration between two

timepoints. A contingent link models an uncontrollable process whose uncertain duration, !, may last any duration between the specified lower and upper bounds. All contingent links terminate on a contingent timepoint whose timing is controlled exogenously. All other timepoints are called requirement timepoints and are controlled by the agent. Fig.2 presents an STNU representing the cooperative rover scenario of the preceding section.

Figure 2: STNU for Cooperative Rover Scenario

This STNU must be checked to determine if a dynamically controllable execution strategy exists, and then compiled into a dispatchable form. Once the plan is dispatchable, the dispatcher consistently and efficiently schedules timepoints through local propagation of timebounds [Muscettola 1998, Morris et al. 2000]. We now review how STNUs are compiled into dispatchable form and present the dispatchable form for the rover scenario (Fig.5).

To support efficient inference, an STNU is mapped to an equivalent distance graph [Dechter 1991], which we call a Distance Graph with Uncertainty (DGU). Each link of the STNU, containing both lower and upper bounds, is converted to a pair of edges in the DGU. One edge in the forward direction is labeled with the value of the upper time bound, and one edge in the reverse direction is labeled with the negative of the lower time bound. The distinction between contingent and requirement edges is maintained.

Fig.3 presents the cooperative rover scenario DGU. Edge BC [5, 7] in Fig.2 is converted to a pair of edges in Fig.3, where the forward edge BC value is 7, and the reverse edge CB value is -5.

An STNU is consistent only if its associated distance graph contains no negative cycles [Dechter 1991]. This can be efficiently checked by applying the Bellman-Ford SSSP algorithm [CLR 1990] on the DGU. However, consistency is not sufficient to guarantee dynamic controllability, meaning that there is enough flexibility in the plan to compensate at execution time for temporal uncertainty in the plan. The dynamic controllability (DC) algorithm introduced by [Morris et al. 2001] reformulates the DGU to ensure that each uncontrollable duration, !i, is free to finish any time between [li,ui], as specified by the contingent link, Ci.

We review the three steps of the DC algorithm. The first step (1) computes the APSP-graph of the DGU using the Floyd-Warshall algorithm [CLR 1990] in order to expose implicit temporal constraints. Exposing implicit constraints is necessary to ensure events are scheduled in the proper order, and with requisite temporal distances between events. If the exposed constraints imply strictly tighter bounds on an uncontrollable duration, then that uncontrollable duration is squeezed [Morris et al. 2001]

and the plan is not dynamically controllable. In this case
there exists a situation [Vidal 1999] where the outcome of
the uncontrollable duration results in no feasible schedule
of controllable events to satisfy the STNU. An STNU is
pseudo-controllable [Morris et al. 2001] if it is both
temporally consistent and none of its uncontrollable
durations are squeezed.

However, even if an STNU is pseudo-controllable, the
uncontrollable durations may be squeezed at execution
time [Morris et al. 2001] as follows. When the dispatcher
executes a timepoint, it fixes the value of the timepoint.
Updating the implicit constraints based on this value may
then squeeze, meaning imply tighter bounds, on a
contingent link. To avoid squeezing uncontrollable
durations, the DC algorithm, Step (2) adds constraints to
the plan. The constraints take the form of simple temporal
constraints and conditional constraints (or "wait"
constraints) and are applied according to the precede,
unordered, and unconditional unordered reduction rules
described in [Morris et al. 2001]. We review the reduction
rules since they are important to understanding our
incremental update rules.

Consider the triangular DGU shown in Fig.4. Assume
the DGU is both pseudo-controllable and in APSP-form.

When C is executed before B (v"0, u<0), the dispatcher
will never know the execution of the contingent timepoint
B when it needs to schedule timepoint C. To maintain
dynamic controllability, the dispatcher must avoid a
situation in which uncontrollable duration AB is squeezed
due to propagations from CB and BC during dispatching.
To ensure this does not happen, the dispatcher must
constrain the temporal relationship between timepoints A
and C such that, no matter how long uncontrollable
duration AB takes within [x, y], timepoint C can be
executed to satisfy constraints CB and BC. The precede
reduction achieves this by tightening constraints AC and
CA as follows. When the execution of B and C are unordered (v # 0 and
u"0), the unordered reduction uses a conditional constraint
to prevent propagations from possibly squeezing the
uncontrollable duration AB during dispatching. If C is
executed before B (as in the precede reduction), constraint
CA must be tightened to ensure that no matter how long
uncontrollable duration AB takes within [x, y], constraint
CB will be satisfied. If B is executed before C, then the
dispatcher knows the execution time of B when scheduling
timepoint C, and tightening CA is not necessary.

Figure 4: Triangular Distance Graph with Uncertainty (DGU) Definition (Unordered Reduction [Morris et al. 2001])

If v # 0 and u ! 0, apply a conditional constraint CA of <B,
v-y>.

For example, in Fig.5 conditional edge GC labeled <-3, D>
specifies that G must wait at least 3 time units after C

executes or until D executes, whichever comes first. We
call a DGU containing a set of conditional constraints a
Conditional Distance Graph with Uncertainty (CDGU).

If the conditional edge created by the unordered reduction requires that C is always executed before B, then
the edge is unconditional. The unconditional unordered
reduction describes when to convert the conditional edge
into a requirement edge.

Definition (Unconditional Unordered Reduction [Morris et al. 2001]) Given an STNU with contingent link
AB [x,y], and associated CDGU with a conditional
constraint CA of <B,-t>, if x>t, then convert the
conditional constraint into a requirement edge CA with
distance –x.

Step (3) of the DC algorithm applies the rules for
regression to the conditional constraints in the CDGU. The
rules for regression, described in [Morris et al. 2001], add
constraints to the CDGU to ensure that the conditional
constraints created by the reduction rules are not violated
at execution and are satisfied for all outcomes of
uncontrollable events. We review the regression rules since
they are also important to understanding our incremental
update rules.

Lemma (Regression [Morris et al. 2001]): Given a
conditional constraint CA of <B,t>, where -t is less than or
equal to the upper bound of contingent link AB. Then (in a
schedule resulting from a dynamic strategy):

i.) If there is a requirement edge DC with distance w,
where w # 0 and D $ B, we can deduce a conditional
constraint DA of <w+t, B>.

ii.) If t < 0 and if there is a contingent link DC with bounds
[x,y] and B " C, then we can deduce a conditional
constraint DA of <x+t, B>.

The rules for regression are applied recursively to all
conditional constraints in the CDGU, until no more
regressions are possible.

IV. INCREMENTAL ALGORITHM

In this section, we present our incremental algorithm, IDC,
which enables the agent to quickly maintain dispatchability
after a fast replanner modifies a subset of the constraints.

IDC uses incremental update rules in the spirit of incremental search algorithms [Koenig and Likhachev
2001], and employs a set of support similar to truth
maintenance systems [Doyle 1979]. The key innovation of
our algorithm is a unified set of incremental update rules
that exploit the causal structure of the plan to interleave
and efficiently apply the different types of propagation in
the DC algorithm. This is in contrast to how the DC
algorithm repeatedly computes the all-pairs shortest path
(APSP) graph and repeatedly checks all possible triangles
in the network for reductions.

Our IDC algorithm maintains dispatchability when
constraints in the plan are both tightened (or added) and
loosened (or removed). We first address the problem of
maintaining dispatchability when constraints are tightened.

We then provide an intuitive explanation for the difference
between maintaining dispatchability when constraints are
tightened versus loosened, and address the problem of
maintaining dispatchability when constraints are loosened.

V. CONSTRAINT TIGHTENING

The speed of our IDC algorithm is derived from exploiting
the causal structure of a dispatchable plan to propagate
constraint modifications throughout the plan. We introduce
a technique we call dispatchability back-propagation
(DBP) to resolve STN constraint tightening. We then
present a unified set of incremental update rules derived
from DBP, reduction, and regression rules to resolve the
constraint tightening in an STNU; by resolve we mean toConstraint Tightening
T(C) - d(BC), which implies T(C) - T(A) < d(AB) +
d(BC). Adding an edge AC of d(AB) + d(BC) to G
encodes this constraint. Similar reasoning applies for case
(ii) when a negative edge changes.
Recursively applying rules (i) and (ii), when an edge is
tightened in a dispatchable distance graph, will either
expose a direct inconsistency or result in a dispatchable
graph1
. The key feature of DBP is that it only requires a subset of the edges be checked to ensure the modified
constraint is consistent, rather than all edges when the
APSP-graph is computed.
For the DC algorithm, in addition to computing implied
constraints by generating the APSP-graph, the algorithm
applies reduction and regression rules to ensure that
uncontrollable durations are not squeezed at execution
time. Likewise, to resolve squeezing in our IDC algorithm,
we interleave the DBP rules with incremental updates rules
derived from the reduction and regression rule sets. This
unified set of incremental update rules (described in Table
1) is used to ensure dynamic control. Each incremental
update rule differs, depending on the types of edges
involved, the signs of the edge distances, and the relative
direction of the edges. A DGU consists of five types of
edges: positive and negative requirement edges, positive
and negative contingent edges, and negative conditional
edges. The incremental update rules describe the
propagation of three of these edge types: negative
requirement edges, positive requirement edges, and
negative conditional edges - these are the only three types
of edges that may be added or modified during
compilation. (Any positive conditional edge is converted to
a requirement edge by the unconditional unordered
reduction rule.)
TIGHTEN, which uses the unified set of incremental
update rules to maintain dispatchability of a conditional
distance graph with uncertainty (G) when a subset (e1…en)
of edges are tightened or added to the graph. Since the
incremental update rules propagate edge updates towards

the start of the plan, we reduce the amount of redundant
work in BACKPROPAGATE-TIGHTEN by initiating
propagations near the end of the plan first. In Line 1, the
relevant timepoint for each new or modified edge is
ordered according to single-destination shortest-path
(SDSP), from lowest to highest. IDC chooses the relevant
timepoint based on how the edge is back-propagated, it:
(1)
uses the source timepoint of the edge if the edge is
conditional or the edge value is less than or equal to zero,
and
(2)
uses the target timepoint if the edge value is greater
than zero. Then, for each edge in the ordered list, IDC
checks if edge ei is a loop (i.e. starts and ends at
the sameTIGHTEN, which uses the unified set of incremental
update rules to maintain dispatchability of a conditional
distance graph with uncertainty (G) when a subset (e1…en)

VI. TIGHTENING VS. LOOSENING CONSTRAINTS

In this section, we use a simple STN example to provide an
intuitive explanation for the difference between
maintaining dispatchability when constraints are tightened
versus loosened. Consider the distance graph of a STN
shown in Fig.7a. The associated APSP graph is shown in
Fig.7b. The APSP computation is used to reduce a STN
into dispatchable form [Muscettola 1998].

When a constraint is tightened, this change needs only to
be made consistent with the past scheduling decisions and
the dispatcher will then ensure that this constraint change
is consistent with the future at execution time. To illustrate
this, consider what happens when edge AB is tightened
from 5 to 4 (Fig.7c). As long as this change is consistent
with the past (it is), then the dispatcher is able to
compensate for the tightening of AB by choosing the
appropriate execution time of C within the range [11, 13]
after B.

In contrast, consider what happens when edge AB is
loosened from 5 to 6 (shown in Fig.7d). Timepoint C must
now be executed with a new lower bound of 9 time units
after B to ensure that C occurs exactly 15 time units after
A. The value 9 is not within the range [10, 13]; a situation
may arise where the dispatcher cannot compensate for the
loosening of AB using the dispatchable form. However,
remember that the BC timebound before the APSP
computation was [9, 13]. The value of edge CB was
tightened from -9 to -10 during the APSP computation
using edge values CA and AB as support. Since the value
of AB has changed, CB can revert back to -9.
Dispatchability is maintained as long as the AB value of 6
and CB value of -9 are consistent with previous timepoints.

This simple STN example shows that it is necessary to
maintain a list of edge value support to identify the
influence of loosening a temporal constraint. A similar

argument can be made for maintaining a list of dominated
edge support. Support lists are also used if a constraint is
removed from a network, since this is an edge loosening
from a finite value to positive or negative infinity.

Support lists, also called set of support, were first used
for incremental updates in truth-maintenance systems
[Doyle 1979], in order to record justification, recognize
inconsistencies, and remember derivations. In this spirit,
IDC, like other incremental graph algorithms, uses support
when constraints are loosened to identify edge values that
are no longer valid and revert them to supported values.
IDC is unique in that it also uses support to .

REFRENCES

[Demestrescu, C., Emiliozzi, S., Italiano, G. 2004]
Experimental analysis of dynamic all pairs shortest path
algorithms. Proc. SODA'04, pp. 362-271.
[Doyle 1979] A truth maintenance system. AI, 12:231-272.
[Effinger 2006] Optimal Temporal Planning at Reactive
Time Scales via Dynamic Backtracking Branch and
Bound, S.M. Thesis, MIT.
[Hofmann, A., Williams, B. 2006] Robust execution of
temporally flexible plans for bipedal walking devices.
Proc. ICAPS-06.
[Koenig, S., Likhachev, M. 2001] Incremental A*.
Advances in Neural Information Processing Systems (14).
[Morris, P., Muscettola, N. 2000] Execution of temporal
plans with uncertainty. In: Proc. AAAI-00.
[Morris, P., Muscettola, N., Vidal, T. 2001] Dynamic
Control of plans with temporal uncertainty. In: Proc.
IJCAI-01.
[Muscettola, N., Morris, P., Tsmardinos, I. 1998].
Reformulating temporal plans for efficient execution. Proc.
KRR-98.
[Muscettola, N., Nayak, P., Pell, B., Williams, B. 1998b]
To boldly go where no AI system has gone before. AI
103(1):5-48.
[Radibeau, G., Knight, R., Chien, S., Fukunaga, A, et
al. 1999] Iterative Repair Planning for Spacecraft
Operations in the ASPEN System, Proc. i-SAIRAS.
[Robertson, P., Williams, B. 2005] A Model-Based
System Supporting Automatic Self-Regeneration of
Critical Software, Proceedings of the IFIP/IEEE Intl
Workshop on Self-Managed Systems & Services, France.
[Shu, I., Effinger, R., Williams, C. 2005] Enabling Fast
Flexible Planning through Incremental Temporal.