MIT-CSAIL-TR-2018-004 January 29, 2018

Fast, Approximate State Estimation of Concurrent Probabilistic Hybrid Automata

Eric Timmons

Fast, Approximate State Estimation of Concurrent Probabilistic Hybrid Automata

by

Eric Timmons

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

Author . . . .
Department of Aeronautics and Astronautics
December 11, 2013

Certified by . . . .
Brian C. Williams
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by . . . .
Eytan H. Modiano
Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Fast, Approximate State Estimation of Concurrent Probabilistic Hybrid Automata

by

Eric Timmons

Submitted to the Department of Aeronautics and Astronautics
on December 11, 2013, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

It is an undeniable fact that autonomous systems are simultaneously becoming more common place, more complex, and deployed in more inhospitable environments. Examples include smart homes, smart cars, Mars rovers, unmanned aerial vehicles, and autonomous underwater vehicles. A common theme that all of these autonomous systems share is that in order to appropriately control them and prevent mission failure, they must be able to quickly estimate their internal state and the state of the world.

A natural representation of many real world systems is to describe them in terms of a mixture of continuous and discrete variables. Unfortunately, hybrid estimation is typically intractable due to the large space of possible assignments to the discrete variables.

In this thesis, we investigate how to incorporate conflict directed techniques from the consistency-based, model-based diagnosis community into a hybrid framework that is no longer purely consistency based. We introduce a novel search algorithm, A* with Bounding Conflicts, that uses conflicts to not only record infeasiblilities, but also learn where in the search space the heuristic function provided to the A* search is weak (possibly due to heavy to moderate sensor or process noise). Additionally, we describe a hybrid state estimation algorithm that uses this new search to perform estimation on hybrid discrete/continuous systems.

Thesis Supervisor: Brian C. Williams
Title: Professor of Aeronautics and Astronautics

Acknowledgments

First, I would like to thank my friends and family for helping support me throughout this entire process and being there whenever I needed them. Special thanks must be given to those that have had to put up with me as I wrote, rewrote, and rewrote this thesis many times. You know who you are.

Second, I would like to thank my me fellow graduate students in the MERS lab. Late night discussions and presentations at group meetings helped me realize the shortcomings in my research and all of you encouraged me, just by talking with me about it, to push harder on my research until I broke through the conceptual barriers holding me back. You all have also made my time in the lab a very enjoyable experience, so much so that I'm still sticking around for a while!

Last, but certainly not least, I would like to thank my thesis advisor, Brian Williams. Brian has been an excellent sounding board throughout this process and has helped make my technical writing much stronger than it was when I first started grad school. However, I know I still have many areas I can improve in both my writing and research skill set, but I have no doubt that Brian will be there whenever I need him, especially as I start to work on my next publication.

This work was funded by a grant from the Office of Naval Research through the Johns Hopkins University Applied Physics Laboratory, contract number N00014-08-C-0681.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Algorithms

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

It is an undeniable fact that autonomous systems are simultaneously becoming more common place, more complex, and deployed in more inhospitable environments. Examples include smart homes, smart cars, Mars rovers, unmanned aerial vehicles, and autonomous underwater vehicles. A common theme that all of these autonomous systems share is that in order to appropriately control them and prevent mission failure, they must be able to quickly estimate their internal state and the state of the world.

A natural representation of many real world systems is to describe them in terms of a mixture of continuous and discrete variables. Examples of these *hybrid discrete/continuous systems* include fluid systems where each component has a discrete operating mode (open/closed, on/off) and continuous dynamics on pressure and flow; power systems where breakers can be open or closed and the continuous values are power, voltage, and current; and an image tracking system that attempts to identify discrete properties of elements in the image such as in free fall or at rest based on the continuous values of position and velocity.

In order to estimate the state of the system, hybrid estimators must fully leverage both the discrete and continuous properties of the system during state estimation. The system's discrete modes are often of great interest to system operators. For example, operators would like to know if a valve is stuck open, or if a pipe has burst. However, these discrete parts of the system are often not directly observable; they can only be observed through the effect they have on the continuous dynamics of the system. Additionally, hybrid systems may display the early symptoms of a failure by slight changes in the continuous dynamics or the violation of one or more constraints. In order to detect the onset of failure early, hybrid estimators need to be able to reason about the known constraints of the system and how sensor noise can affect the observations of the system.

Classical state estimation techniques, such as the discrete hidden Markov model forward algorithm and Kalman filtering, provide crucial insight on how to approach these problems for purely discrete or purely continuous systems in a Bayesian framework. However, even when combined together, these techniques are not sufficient to deal with the scaling inherent in mixing discrete and continuous variables together in the same system; the number of discrete states the system can be in scales exponentially in the number of components. For instance, a simple system with 20 components that each have 3 discrete operating modes has over 3 billion possible discrete states, that also means 3 billion possible sets of continuous dynamics to track.

On the other hand, the model-based diagnosis community has provided insight on how to deal with the combinatorics inherent in dealing with discrete variables. One technique commonly seen in this community is the use of *conflict directed* methods to prune large sections of the discrete search space proven to be infeasible. These techniques work by first enumerating mode candidates in order of decreasing likelihood. Each mode candidate is then tested for consistency with the system observations, and if it is inconsistent, the reason for the inconsistency is learned as a conflict in the form of a partial mode assignment. The learned conflicts are then used in the candidate generation phase to avoid candidates that look promising (have high a priori probability), but are actually infeasible (inconsistent with the observations).

While powerful, it is unclear how to directly apply these conflict directed techniques to Bayesian estimation of hybrid discrete/continuous systems. The reason for this is that real world sensors and systems are noisy, and many noise models, including Gaussian models, consider any amount of disturbance to be feasible, although increasingly unlikely as the disturbances grow large. This lack of hard constraints on the noise means that the infeasibilities required by the model-based diagnosis community to learn conflicts don't exist.

In this thesis, we develop hybrid discrete/continuous state estimation algorithms capable of quickly generating the most likely state estimates for complex systems with noisy processes and observations, by addressing these shortcomings through the introduction of *bounding conflicts*. We do this by developing a hybrid state estimator, named Bones, that unifies the work of Hofbaur and Williams [5] with Williams and Ragno [9] and introducing two key innovations:

1. we introduce a novel conflict directed enumeration algorithm that speeds up the enumeration process relative to the state of the art for both constrained and unconstrained optimization problems;

2. we provide an innovative approach to scale hybrid estimation to large systems by generalizing conflict directed methods for purely discrete models to probabilistic hybrid systems, using the previously mentioned enumeration algorithm;

1.1 Approach and Innovations

Consistency based mode estimation methods, both static and dynamic, have been developed that are effective for diagnosis and mode confirmation at the system level. These methods handle single and multiple faults, modeled and unmodeled failures, sensor failures, autonomous and commanded changes in behavior and intermittency. These methods efficiently solve very large, real world problems through a combination of best first enumeration and conflict-directed search.

Hybrid estimation methods were introduced in order to infer continuous state as well as discrete modes, infer the unobserved discrete modes using continuous observations, and to deal with subtle deviations in system behavior that are hidden by noise. A major challenge for the community has been to scale hybrid estimation to large scale applications, by generalizing conflict-directed best first search to hybrid problems.

Hofbaur and Williams [5] demonstrated the effectiveness of BFE for hybrid problems, applied to life support monitoring. The generalization of hybrid estimation to conflict-directed best first enumeration so crucial to the consistency based mode estimation community, however, has eluded the community over the last decade.

In this thesis, we close that gap by presenting a generally applicable, theoretically sound method for applying conflict-directed techniques to best first enumeration for problems without hard constraints. This method has been applied to, and practically validated on, large hybrid systems to enumerate state estimates in order of decreasing posterior probability (modulo some constant multiplicative factor)[1].

The major innovation in this thesis is the development of the A* with Bounding Conflicts (A*BC) algorithm for constrained or unconstrained optimization problems. This algorithm enumerates assignments to discrete variables in best first order with respect to some utility or cost function when given access to an appropriate heuristic function. It accomplishes this using *bounding conflicts* to describe areas of the search space that have a higher cost than the heuristic function A*BC is given access to would otherwise suggest. Bounding conflicts are presented as a natural extension of conflicts from a consistency based domain to a real valued domain. Figure 1-1 shows conceptually how bounding conflicts are used to skip over portions of the search space while maintaining completeness.

The second contribution of this thesis is a detailed account of how to apply the bounding conflict-directed best first enumeration algorithm to the problem of Bayesian hybrid state estimation. This approach uses the A*BC algorithm to enumerate mode assignments in best first order with respect to an appropriate bounding function on the posterior probability of the mode. This upper bound is then corrected to the true posterior probability by using the observation likelihood function. If the correction factor is large, the reason for the large correction is extracted as a bounding conflict and given to the mode candidate generator.

1.2 Thesis Layout

The rest of this thesis is structured as follows. Chapter 2 provides definitions for the probabilistic hybrid automaton and concurrent probabilistic hybrid automaton modeling formalisms used in the rest of the thesis and introduces the running examples. Chapter 3 then develops the fundamentals of state estimation on hybrid automata by introducing an

[1]For the rest of this section, posterior probability will be taken to mean “posterior probability, modulo some constant multiplicative factor.”

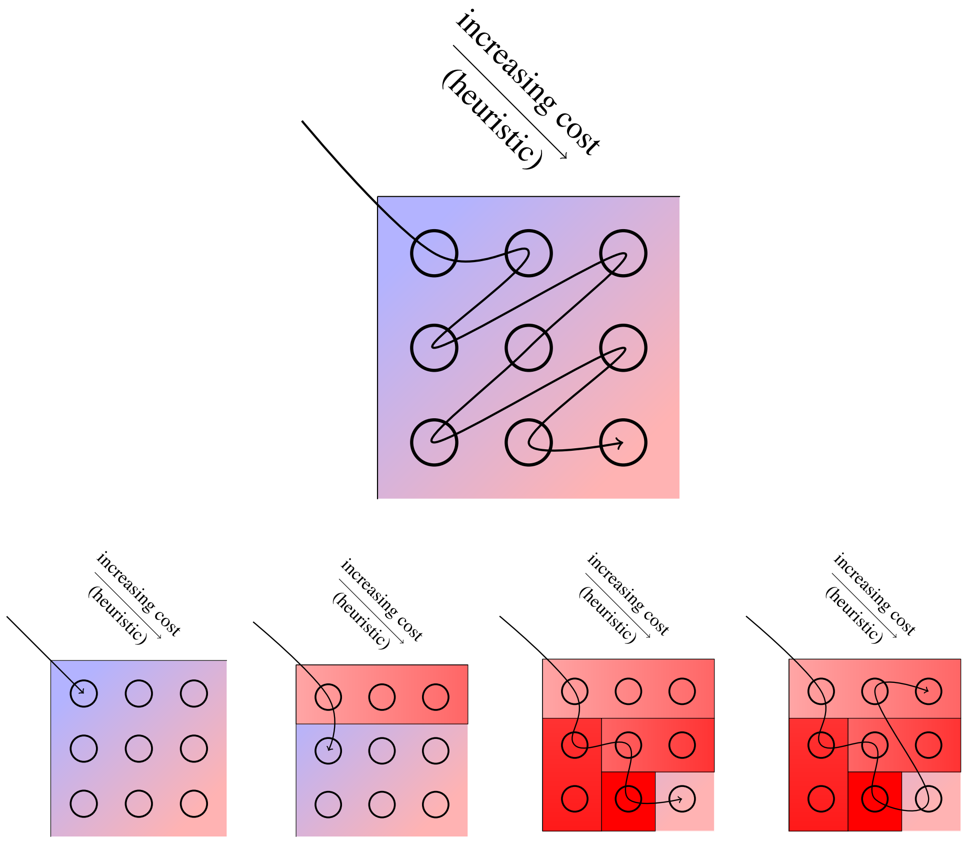


Figure 1-1: This figure shows a conceptual view of the A* with Bounding Conflicts (A*BC) search algorithm. The circles represent nodes in a discrete search space and the color of the background represents the heuristic cost of the node, with red being higher than blue. Unlike [9], in this problem **all** states are feasible, and no hard constraints exist. Assume that the bottom right node is the true optimal node. A traditional A* search (top) will visit every single node with a heuristic cost lower than the heuristic cost of the optimal. However, in an A*BC search, a candidate will be generated, and its value estimated to a tighter bound. If the bound tightens significantly, the new bound is generalized to similar states, and the enumeration of those states is deferred until later in the search process. This is achieved by using bounding conflict learning to improve the heuristic estimate employed by A*. A*BC maintains completeness by exploring regions denoted by bounding conflicts when no better search node exists (bottom right).

algorithm to perform static state estimation on cPHAs. Static state estimation ignores both the continuous and discrete dynamics of the system and focuses on algebraic constraints. Next, Chapter 4 describes the A* with bounding conflicts algorithm, a key enabling algorithm that can be used by hybrid state estimation algorithms to generate mode assignments. Chapter 5 then builds on Chapter 3 by considering the continuous and discrete dynamics of the system. By the end of Chapter 5, a complete hybrid state estimation algorithm will be defined. Last, Chapter 6 analyzes the experimental results of this thesis' algorithms, summarizes the contributions of this thesis, and discusses areas for further research.

Chapter 2

Definitions

This chapter provides definitions used throughout the remainder of this thesis. These definitions include probabilistic hybrid automata which describe components with both discrete and continuous properties, concurrent probabilistic hybrid automata which describe systems of probabilistic hybrid automata operating together, and hybrid state estimates which are the output of the core algorithms of this thesis. Additionally, we provide examples to concretely illustrate each of these definitions.

2.1 Models

2.1.1 PHA

A probabilistic hybrid automaton is a way to model components of many real world systems that have both discrete and continuous dynamics. Each automaton contains one discrete valued mode variable, the value of which affects the dynamics of the continuous system. This modeling formalism is similar to jump Markov systems and has been used previously by [5]. The definition adopted in this thesis, however, is slightly different than [5] and emphasizes the inclusion of process and sensor noise characteristics with the automaton definition.

Definition 2.1 (PHA)**.** *A discrete time probabilistic hybrid automaton (PHA) $\mathcal{A}$ can be described as a tuple $\langle \mathbf{x}, \mathbf{w}, F, N_x, N_y, T, \mathcal{X}_d, \mathcal{U}_d, T_s \rangle$:*

- *$\mathbf{x}$ is the set of the automaton's hybrid state variables. It is composed of $\mathbf{x} = \{x_d\} \cup \mathbf{x}_c$, where x_d is a discrete variable in the domain $\mathcal{X}_d$ that denotes the mode of the automaton. $\mathbf{x}_c$ represents the continuous state of the automaton. $\mathbf{x}_c = \{x_{c,1}, \ldots, x_{c,n}\}$ and has domain $\mathbb{R}^n$. We assume without loss of generality that a PHA has only one discrete variable.*

- *$\mathbf{w} = \mathbf{u}_d \cup \mathbf{u}_c \cup \mathbf{y}_c$ represents the set of input and output variables for this automaton. These variables are composed of discrete input variables $\mathbf{u}_d = \{u_{d,1}, \ldots, u_{d,m_d}\}$ with domain $\mathcal{U}_d$, continuous inputs and outputs $\mathbf{u}_c = \{u_{c,1}, \ldots, u_{c,m_i}\}$ with domain $\mathbb{R}^{m_i}$, and continuous observables $\mathbf{y}_c = \{y_{c,1}, \ldots, y_{c,m_o}\}$ with domain $\mathbb{R}^{m_o}$.*

- *The function $F : \mathcal{X}_d \rightarrow F_{DE} \cup F_{AE} \cup F_{OBS}$ maps the automaton's operating mode to a set of equations that determines the continuous evolution of the automaton. This evolution is specified in terms of algebraic equations F_{AE}, observation equations F_{OBS}, and discrete time differential equations F_{DE} with sampling time T_s. The algebraic equations are of the form:*

$$f_{AE,i}(\mathbf{x}_c, \mathbf{u}_c) \leq 0 + \nu_{a,i} \tag{2.1}$$

and the differential equations are of the form:

$$\dot{x}_{c,j} = f_{DE,j}(\mathbf{x}_c, \mathbf{u}_c) + \nu_{d,j} \tag{2.2}$$

and the observations are of the form:

$$y_{c,k} = f_{OBS,k}(\mathbf{x}_c, \mathbf{u}_c) + \nu_{o,k} \tag{2.3}$$

ν_a, ν_d, and ν_o are all zero mean, Gaussian random variables.

- *As seen in Equations 2.1 and 2.2, the algebraic and differential equations are corrupted by white Gaussian noise, ν_a and ν_d. These disturbances are specified by the covariance matrix $\mathbf{Q}$ where $\mathbf{Q}$ is mode dependent and given by the function*

$N_x : \mathcal{X}_d \rightarrow \mathbb{R}^{p \times p}$, *where:*

$$p = n + (\textit{number of algebraic equations}). \tag{2.4}$$

- *As seen in Equation 2.3, the observations are also corrupted by additive, white Gaussian noise. The disturbance ν_o is specified by the covariance matrix $\mathbf{R}$ where $\mathbf{R}$ is mode dependent and given by the function $N_y : \mathcal{X}_d \rightarrow \mathbb{R}^{m \times m}$.*

- *T is a finite set of transitions that specify the probabilistic, discrete evolution of the automaton. Each transition is a tuple $\langle x_{d,source}, \tau_i, c_i \rangle$. $x_{d,source}$ is the source mode for transition i. The transition function τ_i specifies the probability mass function over target modes $x_d \in \mathcal{X}_d$ for transition i and $c_i : \mathbb{R}^n \times \mathcal{U}_d \rightarrow \{true, false\}$ is the guard condition. Transition i is activated only when $c_i(\mathbf{x}_c) = true$.*

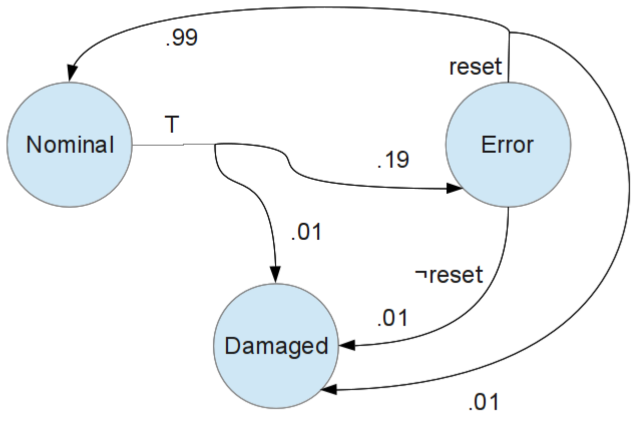


Figure 2-1: Transitions for the fluid sensor.

Example 2.1. *Below is a PHA model of a flow meter for fluid systems. Table 2.1 summarizes the PHA.*

The sensor has three operating modes: nominal, error, and damaged, and four continuous state variables: pressure difference Δp, flow rate f, sensor error ϵ_r, and flow leakage Δr.

The inputs to the sensor are the input pressure p_{in}, input flow rate f_{in}, and discrete command $u_{command}$. The domain of the discrete command variable is $\{$ reset, nothing $\}$.

The outputs are the sensor reading y_f, the output pressure p_{out}, and the output flow rate, f_{out}.

When the sensor is in the "nominal" mode, everything behaves as expected, the pressure difference is equal to the flow rate multiplied by the sensor's resistance and the sensor reports the true flow rate through the sensor. In this mode, the sensor has a 20% chance of entering the "error" state at any given time and a 1% chance of entering the "damaged" mode.

When the sensor is in the "error" mode, the only difference from the "nominal" mode dynamics is that the sensor reports the true flow rate offset by some constant error. When in this mode, the sensor has a 1% chance of entering the "damaged" mode. If the "reset" command is given, it will enter "nominal" with a probability of 100%.

When the sensor is in the "damaged" mode, all of the fluid that enters the sensor leaks out into the surrounding environment. This also results in the flow reading from the sensor reporting only noise. When in this mode, the sensor cannot leave it.

2.1.2 cPHA

A PHA describes the components of a system and a concurrent probabilistic hybrid automaton describes systems composed of many PHA. These automata describe the components they contain, how the components are connected together, and the system level inputs and observables. Concurrent probabilistic hybrid automata have also been used in [5]. The definition adopted in this thesis is modified to emphasize the static constraints that exist between components in the automaton.

Definition 2.2 (cPHA). *A concurrent probabilistic hybrid automaton (cPHA) $\mathcal{CA}$ can be described by the tuple $\langle A, \mathbf{u}, \mathbf{y}_c, S \rangle$, where:*

- $A = \{\mathcal{A}_1, \ldots, \mathcal{A}_l\}$ *denotes the the finite set of l PHAs contained within this automaton. We will denote the component of a specific automaton $\mathcal{A}_i$ by appending $[i]$ to it, e.g., $x_d[i], \mathbf{x}_c[i], F[i]$, etc.*

- $\mathbf{u} = \mathbf{u}_d \cup \mathbf{u}_c$ *denotes the set of input variables for the automaton.* $\mathbf{u}_d = \mathbf{u}_d[1] \cup \ldots \cup$

Table 2.1: Summary of the PHA model of a fluid flow sensor.

Variable	Value
$\mathcal{X}_d$	{ nominal, error, damaged }
$\mathbf{x}_c$	$\{\varepsilon_r\}$
$\mathbf{u}_c$	$\{p_{in}, f_{in}\}$
$\mathbf{y}_c$	$\{p_{out}, f_{out}, y_f\}$
$\mathbf{u}_d$	$\{u_{command}\}$
F_{AE}(nominal)	$\{fr = p_{in} - p_{out} + \nu_{a,1},$ $0 = f_{in} - f_{out} + \nu_{a,2}\}$
F_{AE}(error)	$\{fr = p_{in} - p_{out} + \nu_{a,1},$ $0 = f_{in} - f_{out} + \nu_{a,2}\}$
F_{AE}(damaged)	$\{f_{in} \geq 0 + \nu_{a,1}$ $f_{out} \leq 0 + \nu_{a,2}\}$
F_{DE}(error)	$\{\varepsilon_{r,k} = \varepsilon_{r,k-1} + \nu_{d,1}\}$
F_{OBS}(nominal)	$\{y_f = f_{out} + \nu_{o,1}\}$
F_{OBS}(error)	$\{y_f = f_{out} + \varepsilon_r + \nu_{o,1}\}$
F_{OBS}(damaged)	$\{y_f = \nu_{o,1}\}$
N_x(nominal)	$\begin{bmatrix} .01 & 0 \\ 0 & .01 \end{bmatrix}$
N_x(error)	$\begin{bmatrix} .01 & 0 & 0 \\ 0 & .01 & 0 \\ 0 & 0 & 5 \end{bmatrix}$
N_x(damaged)	$\begin{bmatrix} .01 & 0 \\ 0 & .01 \end{bmatrix}$
N_y(nominal)	$\begin{bmatrix} .01 \end{bmatrix}$
N_y(error)	$\begin{bmatrix} 1 \end{bmatrix}$
N_y(damaged)	$\begin{bmatrix} 5 \end{bmatrix}$
T	$\{\langle$nominal, $\{$error $= 20\%,$ damaged $= 1\%\}, \top\rangle,$ $\langle$error, $\{$nominal $= 99\%,$ damaged $= 1\%\}, u_{command} =$ reset$\rangle,$ $\langle$error, $\{$damaged $= 1\%\}, u_{command} \neq$ reset$\rangle$

$\mathbf{u}_d[l]$ *specifies the discrete command variables and* $\mathbf{u}_c \subseteq \mathbf{u}_c[1] \cup \ldots \cup \mathbf{u}_c[l]$ *represents the continuous input variables.*

- $\mathbf{y}_c \subseteq \mathbf{y}_c[1] \cup \ldots \cup \mathbf{y}_c[l]$ *specifies the output variables of the automaton that are observed.*

- $S = \{s_1, \ldots, s_j\}$ *specifies a series of static constraints on the system. These constraints specify how the I/O variables of the constituent automata are connected together.*

Appendix A contains algorithms to extract information to extract information from a cPHA, including the process and observation noises for a given mode. In later chapters, we will use notation such as $\mathbf{R}_{\mathbf{x}_d}$ to denote the covariance matrix for the sensor noise in mode $\mathbf{x}_d$ for the entire cPHA; this is short hand for MAKE-R-MATRIX($\mathbf{x}_d$, $\mathcal{CA}$). Similarly for $\mathbf{Q}_{\mathbf{x}_d}$, $F_{AE,\mathbf{x}_d}$ and so on.

Example 2.2 (Fluid System Example)**.** *One example problem used throughout this thesis is a fluid system consisting of a pump, flow meters, valves, loads, and a reservoir. The system is depicted in Figure 2-2. Each line connecting the components represents a set of static constraints in* S*. For example, the output pressure from FM3 is equal to the input pressure to V1 and V2 and the flow out of FM3 is equal to the sum of flows into V1 and V2.*

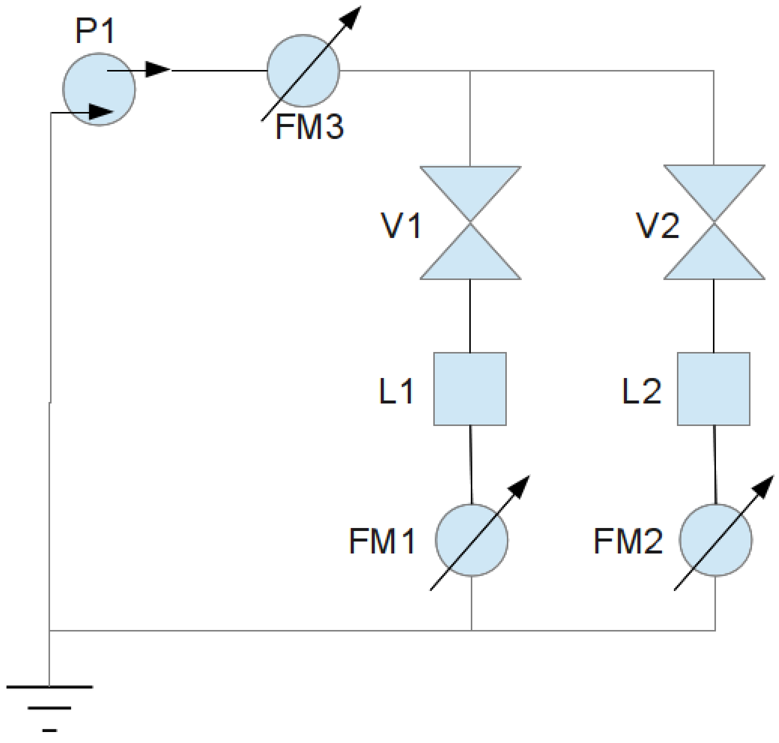


Figure 2-2: Example cPHA.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Static State Estimation of cPHAs

In this chapter, we briefly describe the problem of static state estimation of concurrent probabilistic hybrid automata (cPHAs) and provide a method for solving the static hybrid estimation problem using a best-first enumeration approach. In this chapter we focus solely on the problem of *static hybrid state estimation*; this focus allows us to develop the core insights to formulating the problem of state estimation as a best-first enumeration problem without being burdened by accounting for the dynamics of the system being estimated. Chapter 4 will provide a method to solve best-first enumeration problems of the form described in this chapter, and then Chapter 5 will describe how to solve hybrid state estimation as a best-first enumeration problem by extending the static state estimation modeling to include both discrete and continuous dynamics.

There are two prominent categories used to describe non-sampling based hybrid state estimation techniques: Multiple Model (MM) techniques and Best-First Enumeration (BFE) techniques. MM techniques, with the Interacting Multiple Model (IMM) [1] technique being the commonly used variant, typically require a Kalman or other Bayesian filter to be run for every operating mode the system can enter. For an IMM algorithm, the input to the filters at time t is a mixture of the the estimates from each filter at time $t-1$ that depends on transition probabilities between modes. For systems with a large number of operating modes, such as cPHAs where the number of modes is exponential in the number of components, the requirement to have a filter for each operating mode can make MM approaches intractable.

BFE methods sidestep the need to maintain a filter for every mode by filtering the continuous state only for the most likely modes. This is accomplished by, at every time step, taking the state estimate from the previous time step, determining the mode transitions that maximize the posterior probability, and maintaining only the best modes as input for the next time step. This results in BFE techniques generally running far fewer filters at every time step than IMM techniques, at the expense of performing a search at every update and only approximating the resulting posterior distribution. Previous work on BFE techniques [5] show that they perform well on systems with a large number of operating modes where IMM techniques grind to a halt. In this thesis, we are interested in systems with a number of operating modes that is exponential in the number of components in the system, therefore we will focus on BFE techniques for the remainder of this work.

BFE based estimators make extensive use of heuristics to guide the search for successor modes. However, the posterior probability of a mode assignment typically cannot be fully determined without using a continuous filter for that mode. This leads the BFE techniques to use a *generate and test* approach; mode candidates are enumerated in best first order based on an admissible heuristic (the transition probabilities), and are then tested with the observations to determine the actual posterior probability. This is the same general technique used by diagnosis engines for discrete systems such as GDE and Conflict-directed A*.

These discrete diagnosis engines have long enjoyed the benefits of using *conflicts* learned during the testing phase to prune out large amounts of the search space for the candidate generator. However, these conflicts represent hard constraints on the system, i.e., subsets of the mode assignments that are completely infeasible. When working with hybrid systems subject to noise it is difficult to extract such hard conflicts on the system, as many common noise models, such as Gaussian noise, assign non-zero probability to every observation in every mode. Some BFE estimators, such as HyDE [6], effectively truncate the noise models by extracting a hard conflict when the likelihood of an observation drops below some user specified threshold. However, this truncation approach presents the user with a dilemma. The higher the threshold is set, the more efficient the search becomes due to more of the search space being pruned; however, increasing the threshold also increases the probability that the true mode assignment will be pruned.

3.1 Problem Statement

State estimation is the act of calculating a belief over the possible states of a system, given the system's inputs and outputs. Given that we are performing hybrid estimation and the continuous and discrete states are tightly coupled, a hybrid estimator must estimate the complete hybrid state, $\mathbf{x} = \langle \mathbf{x}_d, \mathbf{x}_c \rangle$ of the system. As we are focusing only on static state estimation, we are assuming that the system stays in the same state as the observations are being made, meaning effectively that we do not consider the system's continuous or discrete dynamics.

Definition 3.1 (Static Hybrid State Estimation of cPHAs)**.** *Given a system modeled as a cPHA $\mathcal{CA}$, an a priori distribution of the system state (continuous and discrete) $p(\mathbf{x})$, the inputs to the system $\mathbf{u}$, and the observations of the system $\mathbf{y}_c$, estimate the hybrid state of the system $\hat{\mathbf{x}}$.*

We will represent the joint prior distribution over the discrete and continuous states using a marginal distribution for each mode, $p(\mathbf{x}_d)$, and a continuous distribution conditioned on the mode, $p(\mathbf{x}_c|\mathbf{x}_d)$. Where:

$$p(\mathbf{x}) = p(\mathbf{x}_c|\mathbf{x}_d)p(\mathbf{x}_d) \tag{3.1}$$

In this chapter, we will stipulate that we are only interested in estimating the most likely continuous state of the system for a given mode. This means that the hybrid estimate is a tuple over a mode and the most likely continuous state:

$$\hat{\mathbf{x}} = \langle \mathbf{x}_d, \hat{\mathbf{x}}_c^{ML} \rangle \tag{3.2}$$

This assumption is adopted purely for ease of explanation and will be relaxed in Chapter 5.

In order to fully specify a posterior distribution of the system, we would need to specify a discrete distribution whose domain is every mode. As the number of possible modes for a cPHA grows exponentially in the number of components, fully representing the posterior distribution quickly becomes intractable. We work around this by only approximating the posterior distribution using only the modes with the highest probability, and approximating the remaining low probability modes as having zero probability of being the correct mode.

This approach of approximating the posterior distribution using only the most likely modes leads us to consider using a best-first enumeration technique in order to cover as much of the total probability mass as possible.

We represent the posterior hybrid belief state as b. The hybrid belief state of hybrid state estimate $\hat{\mathbf{x}}^{(i)}$ is denoted by $b[\hat{\mathbf{x}}^{(i)}]$, where:

$$b[\hat{\mathbf{x}}^{(i)}] = Pr(\hat{\mathbf{x}}^{(i)}|\mathbf{u}, \mathbf{y}_c) \tag{3.3}$$

3.2 Static State Estimation as Best-First Enumeration

In this section, we examine how best first enumeration can be used to perform static hybrid estimation. First, we review how, given a mode assignment, the most likely continuous estimate can be determined. Then, we review how the hybrid belief b is calculated for a given estimate. Last, we briefly describe an algorithm for static hybrid estimation using best first enumeration.

3.2.1 Maximum Likelihood Continuous Estimate

As stated in the problem definition we are considering in this chapter, we are interested in producing hybrid estimates est, containing a mode and the most likely continuous state for that mode. In determining the ML estimate, we are attempting to calculate:

$$\hat{\mathbf{x}}_c^{ML} = \arg\max_{\hat{\mathbf{x}}_c} p(\hat{\mathbf{x}}_c|\mathbf{x}_d, \mathbf{y}_c). \tag{3.4}$$

Using Bayes' rule, this can be shown to be equivalent to:

$$\hat{\mathbf{x}}_c^{ML} = \arg\max_{\hat{\mathbf{x}}_c} p(\mathbf{y}_c|\hat{\mathbf{x}}_c, \mathbf{x}_d)p(\hat{\mathbf{x}}_c|\mathbf{x}_d). \tag{3.5}$$

Under the assumption that both the prior and observation likelihoods are Gaussian distributions, $\hat{\mathbf{x}}_c^{ML}$ is the value that solves the following Quadratic Program (QP):

$$\begin{aligned} &\min_{\hat{\mathbf{x}}_c} (\mathbf{y}_c - F_{OBS,\mathbf{x}_d}(\hat{\mathbf{x}}_c))^T \mathbf{R}_{\mathbf{x}_d}^{-1} (\mathbf{y}_c - F_{OBS,\mathbf{x}_d}(\hat{\mathbf{x}}_c)) + (\hat{\mathbf{x}}_c - \mu)^T \mathbf{Q}_{\mathbf{x}_d}^{-1} (\hat{\mathbf{x}}_c - \mu) \qquad (3.6) \\ s.t. \qquad & F_{AE,\mathbf{x}_d}(\hat{\mathbf{x}}_c) \leq 0 \end{aligned}$$

Where μ and Q represent the prior uncertainty in $\hat{\mathbf{x}}_c$ before measurements are taken.

3.2.2 Hybrid Belief State

Now that we have determined how to calculate the maximum likelihood estimate of a system's continuous state for a specific mode, we will investigate how to calculate the belief that a hybrid estimate $\hat{\mathbf{x}}$ is the correct estimate.

As discussed previously in this chapter, b represents the belief state over the state of the system being estimated. b evaluated for a hybrid estimate returns the likelihood that the system is the state described by the estimate. That is:

$$\begin{aligned} b(\hat{\mathbf{x}}) &= p(\hat{\mathbf{x}}|\mathbf{y}_c) && (3.7) \\ &\propto p(\mathbf{y}_c|\hat{\mathbf{x}})p(\hat{\mathbf{x}}) && (3.8) \\ &\propto p(\mathbf{y}_c|\hat{\mathbf{x}}_c, \mathbf{x}_d)p(\hat{\mathbf{x}}_c|\mathbf{x}_d)p(\mathbf{x}_d) && (3.9) \end{aligned}$$

Where Equation 3.8 is a result of Bayes' rule. We can calculate the first part of b using the results from the QP described in the previous section. We will use:

$$p(\mathbf{y}_c|\hat{\mathbf{x}}_c, \mathbf{x}_d)p(\hat{\mathbf{x}}_c|\mathbf{x}_d) = e^{-\dot{\{\}}} \qquad (3.10)$$

where $\dot{\{\}}$ is the value of the objective function from the QP. The reason for not including the standard Gaussian normalization constant is discussed in Chapter 5. Then, the last term in b is simply the prior belief that the system is in mode $\mathbf{x}_d$.

3.2.3 BFE Approach

In this section, we give an algorithm that can perform static hybrid estimation using a best first approach. The key to this algorithm is noting that, due to the observation likelihood in Equation 3.10 containing no normalization factor, $p(\mathbf{y}_c|\hat{\mathbf{x}}_c, \mathbf{x}_d)p(\hat{\mathbf{x}}_c|\mathbf{x}_d)$ is always ≤ 1. This means that $p(\mathbf{x}_d)$ gives an upper bound on the value of b for *any* hybrid estimate.

For now, assume we have access to a function that can enumerate modes in order of decreasing $p(\mathbf{x}_d)$. We could then go through the modes from most likely to least likely, calculate the true b for the estimates, and then when a mode is found where there are estimates with b values greater than $p(\mathbf{x}_d)$, those estimates are guaranteed to be optimal. Algorithm 3.1 shows this algorithm at a high level.

Algorithm 3.1: STATIC-BFE-ESTIMATION

```
1  begin
2      $\theta \longleftarrow \emptyset$;
3      $\Theta \longleftarrow \emptyset$;
4      while Not Terminated do
5          $\hat{\mathbf{x}}_d$, bound $\longleftarrow$ NEXT-BEST-MODE();
6          $\hat{\mathbf{x}}_c^{ML}, \hat{b} \longleftarrow$ SOLVE-QP($\hat{\mathbf{x}}_d$);
7          $\hat{\mathbf{x}} \longleftarrow \langle \hat{\mathbf{x}}_d, \hat{\mathbf{x}}_c^{ML} \rangle$;
8          $\theta \longleftarrow$ INSERT-AND-SORT($\theta, \hat{\mathbf{x}}, \hat{b}$);
9          while $\hat{b}$[HEAD($\theta$)] $\geq h$ do
10             $\Theta \longleftarrow$ APPEND($\Theta$, POP($\theta$));
11         end
12     end
13     return $\Theta$ as the best estimates;
14 end
```

In the next chapter, we introduce an algorithm not only capable of generating modes in order of decreasing $p(\mathbf{x}_d)$, but can also use information gleaned when calculating the b value of an estimate to improve its enumeration.

Chapter 4

A* with Bounding Conflicts

In the previous chapter, we showed how the problem of static state estimation can be framed as a best first enumeration problem, where the most likely modes of the system are enumerated. In this chapter, we provide the core enabling algorithm for this capability, A* with Bounding Conflicts.

4.1 Introduction

Optimization problems are seen in a wide variety of research topics, including operations research, artificial intelligence, planning, and diagnosis/state estimation. In many of these fields, particularly diagnosis and planning, there is a desire to find several solutions that are close to optimal instead of just the single best solution. There are many potential reasons for this, such as providing a technician with a number of the most likely faults a system is experiencing or providing a user with a number of near optimal plans to accomplish some goal and then letting the user make the final decision.

Typical optimization algorithms include Branch and Bound (BB), A*, and their variants. These algorithms are widely used because they guarantee optimality and are efficient. Best first searches, particularly A* and its variants are well suited for finding multiple good solutions to optimization problems because they can be run as long as needed to produce as many results as wanted or needed. In order to increase the efficiency of BB and A*, variants of the algorithms exist that exploit conflicts, also known as nogoods. Conflicts allow the search algorithms to learn the underlying structure of a problem as it is solving it by recording partial assignments that result in impossible solutions. The algorithms can then use these conflicts to prune the search space. Conflicts have been used in backtrack [3], BB [8], and A* [7] style searches.

Conflict Directed A* (CDA*) [9] is one such algorithm that uses conflicts. CDA* is a variant of A* used to solve optimal constraint satisfaction problems. It uses conflicts to skip over areas of the search space that are estimated, based on their heuristic to have high utility, but are actually infeasible due to the constraints put on the problem variables.

This chapter generalizes the idea of using conflicts to represent areas of both infeasibility and high cost by introducing *bounding conflicts*. We then define a new algorithm, *Bounding Conflict Directed A**, that uses bounding conflicts to prune large portions of the search space, resulting in the algorithm finding the first solutions quickly, but in such a way that completeness is maintained, i.e., all feasible solutions will be enumerated given enough time.

4.2 Problem Statement

We are interested in solving problems known as Optimal Constraint Satisfaction Problems (OCSPs). Prior work [7, 9] has shown that many real world problems, including model-based diagnosis can be modeled as OCSPs. An OCSP consists of a Constraint Satisfaction Problem (CSP), paired with an objective function used to put preferences on valid solutions to the CSP. The goal is to find the solution(s) to the CSP that optimize the objective function. For concreteness, we assume that the objective function is a cost to be minimized.

Definition 4.1 (Constraint Satisfaction Problem). *A constraint satisfaction problem (CSP) is described by the tuple* $\langle \mathbf{X}, \mathbf{D_x}, \mathbf{C_x} \rangle$ *where:*

- $\mathbf{X} = \{x_1, \ldots, x_{n_x}\}$ *is the set of variables in the CSP,*
- *Each variable* x_i *ranges over a domain* D_i *where* $D_i \in \mathbf{D_x}$*,*
- *and* $\mathbf{C_x} = \{c_1, \ldots, c_{n_c}\}$ *is a set of constraints over* $\mathbf{X}$ *where each constraint is a function* $c_i : \mathbf{X} \rightarrow \{True, False\}$*.*

The solution to a CSP is an assignment X to all the variables in $\mathbf{X}$ such that the assignment X satisfies all the constraints in $\mathbf{C_x}$.

A CSP can then extended to an OCSP by the addition of an objective function.

Definition 4.2 (Optimal Constraint Satisfaction Problem). *An optimal constraint satisfaction problem is described by the tuple $\langle \mathbf{Y}, g, \mathit{CSP} \rangle$ where:*

- $\mathbf{Y}$ *is a set of variables called the "decision variables" such that* $\mathbf{Y} \in \mathbf{X}[\mathit{CSP}]$.
- g *is a cost function defined over the decision variables,* $g : \mathbf{Y} \to \mathcal{R}$,
- *CSP is a constraint satisfaction problem as defined in Definition 4.1.*

We will be using a grounded example throughout this chapter to demonstrate the workings of our new algorithm.

Example 4.1 (Simple OCSP). *The running example for the A*BC algorithm development will be the OCSP with:*

$$\mathbf{X} = \{x_1, x_2, x_3, x_4\} \tag{4.1}$$

$$\mathit{dom}(x_i) = \{T, F\} \tag{4.2}$$

$$\mathbf{C} = \{\neg(\neg x_1 \wedge x_2)\} \tag{4.3}$$

$$\mathbf{Y} = \mathbf{X} \tag{4.4}$$

$$\begin{aligned} g(\mathbf{Y}) = {} & 8k(x_1) + 4k(x_2) + 2k(x_3) + k(x_4) \\ & + 15k(\neg x_2 \wedge \neg x_4) + 30k(\neg x_2 \wedge x_4) \end{aligned} \tag{4.5}$$

where k is an indicator function that evaluates to one iff the argument evaluates to true:

$$k(y) = \begin{cases} 1 & \textit{if } y \\ 0 & \textit{if } \neg y. \end{cases} \tag{4.6}$$

This example problem is analogous to a feature selection problem when designing an aircraft. There is a fixed cost for each new feature added, a constraint stipulating a required combination of features, and costs that are incurred due to the interactions between some features.

4.3 Review of Conflict-Directed A*

One state of the art algorithm used to solve OCSPs is Conflict Directed A* (CDA*) [9]. CDA* uses a generate and test approach to solve an OCSP. In the generation step, it produces candidate assignments to the decision variables that minimize the cost in best first order. In the test phase, it then checks to see if a full assignment to the CSP variables exists that makes the problem consistent. If such an assignment to the problem variables exists, then it is returned as the optimal solution. If no consistent solution exists using the candidate's assignments to the decision variables, a *conflict* is extracted that summarizes the reason for the inconsistency. This conflict is then given to the candidate generator so that future candidate assignments to the decision variables will never be inconsistent for the same reason.

4.3.1 Conflicts

Conflicts are a general concept used by many algorithms, including CDA*, to improve efficiency [9, 8, 2] .[1] Essentially, conflicts are used to compactly record a subspace of the solution set that is known to be logically infeasible. The conflict is then used by the search algorithm to avoid that subspace in the future, saving the computations that would have otherwise gone into expanding and testing infeasible solutions.

Definition 4.3 (Conflict)**.** *Let* $\mathbf{X}$ *be a set of variables with domain* $\mathbf{D}$ *and let* C *be a constraint(s) on the variables in* $\mathbf{X}$*. A conflict* γ *of* C *is defined to be a partial assignment to* $\mathbf{X}$ *such that any assignment* y *that extends* γ*, i.e.,* $\gamma \subset y$*, is inconsistent with constraint* C*.*

Any assignment that extends conflict γ *is said to manifest conflict* γ *and any assignment that implies the negation of* γ *is said to resolve the conflict.*

[1]They are also known as *nogoods*.

Given a conflict, it is possible to generate a set of partial assignments that resolve the conflict. These partial assignments are called *constituent kernels*. Conceptually, the set of constituent kernels for a conflict can be calculated by negating the conflict; if some partial assignment entails the negation of a conflict, then it must resolve said conflict.

Definition 4.4 (Constituent Kernel)**.** *k is a constituent kernel of conflict γ, if $k \rightarrow \neg\gamma$.*

4.3.2 Candidate Generator

The heart of CDA* is the candidate generator, as any SAT solver that generates conflicts can be used in the test phase. The candidate generator maintains a search tree over the assignments to the decision variables $\mathbf{Y}$ and expands the search tree by alternately splitting on unassigned variables or unresolved conflicts. The candidate generator expands the search tree in best first order by using an A* search with an admissible heuristic. Simplified pseudo-code for CDA*'s candidate generator is given by Algorithm 4.1. In this algorithm, Γ represents the set of known constraints. The state of the open list and Γ is saved between invocations to the algorithm.

Algorithm 4.1: CDA*-CANDIDATE-GENERATOR

```
Input: OCSP
begin
    open ⟵{ { } };
    Γ ⟵{ };
    while open is not empty do
        x ⟵pop node from open with lowest f value;
        if x manifests a conflict in Γ then
            Continue;
        else if x is a complete assignment to Y[OCSP] then
            return x;
        else if there exists a conflict in Γ not resolved by x then
            SPLIT-ON-CONFLICT(x, OCSP);
        else
            SPLIT-ON-VARIABLE(x, OCSP);
        end
    end
    return no candidate;
end
```

The candidate generator works by maintaining a list of open nodes in the search tree that correspond to partial assignments to $\mathbf{Y}$ and incrementally expanding the most promising nodes. First, the most promising node is popped off the open queue and is checked to determine if it manifests any known conflict. If it does manifest a conflict, that node is discarded because it can not be extended to a logically feasible full assignment. Next, the candidate generator tests if the most promising node is in fact a full assignment and returns it as the next candidate if it is.

If the most promising node is a partial assignment that manifests no conflicts, the candidate generator then attempts to expand it. First, it checks to see if there is a conflict in Γ that is not resolved by the partial assignment. If there is such a conflict, the candidate generator expands the partial assignment so that it resolves that conflict, proactively steering away from infeasible ares of the search space. Last, if the the partial assignment resolves all the known conflicts the candidate generator simply chooses an unassigned variable and expands the partial assignment with every possible assignment to the chosen variable and adds them to the queue. Definitions for SPLIT-ON-VARIABLE and SPLIT-ON-CONFLICT can be found in [9].

4.3.3 CDA* on Example Problem

In order to better demonstrate the improvements our new algorithm makes on a state of the art algorithm, we show how CDA* would solve the example OCSP.

First, CDA* starts with the empty assignment on the queue. Because there are no known conflicts, CDA* will split on an unassigned variable to find the best full assignment to the variables based on the cost to go and cost so far functions. This search continues until a full assignment is generated that minimizes g. For the example problem, this would be $\{x_1 = F, x_2 = T, x_3 = F, x_4 = F\}$. Figure 4-1 shows this first step

This full assignment is then tested for consistency with the constraints in $\mathbf{C}$. It fails the consistency check and the conflict $\{x_1 = F, x_2 = T\}$ is extracted. CDA* then begins to expand nodes again to find the best full assignment that resolves the known conflict. The nodes with g values of 5 and 6 in Figure 4-1 are checked first and each is found to

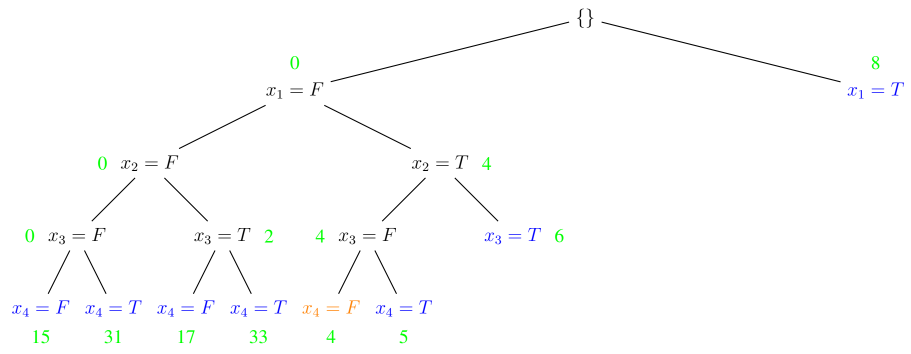


Figure 4-1: The first steps of CDA* on the example problem. The g value for each node is displayed in green next to each node. The cost to go h is zero for all nodes. Nodes are expanded until the best full assignment $\{x_1 = F, x_2 = T, x_3 = F, x_4 = F\}$ is found (shown in orange). The nodes highlighted in blue represent the expansion frontier of the search (the nodes on the queue to be expanded).

manifest the conflict so they are marked as infeasible and not expanded. CDA* then begins expanding the right side of the search tree $(\{x_1 = T\}$ until it finds the next solution that minimizes the cost, which is $\{x_1 = T, x_2 = T, x_3 = F, x_4 = F\}$. The state of the search after this step is shown in Figure 4-2.

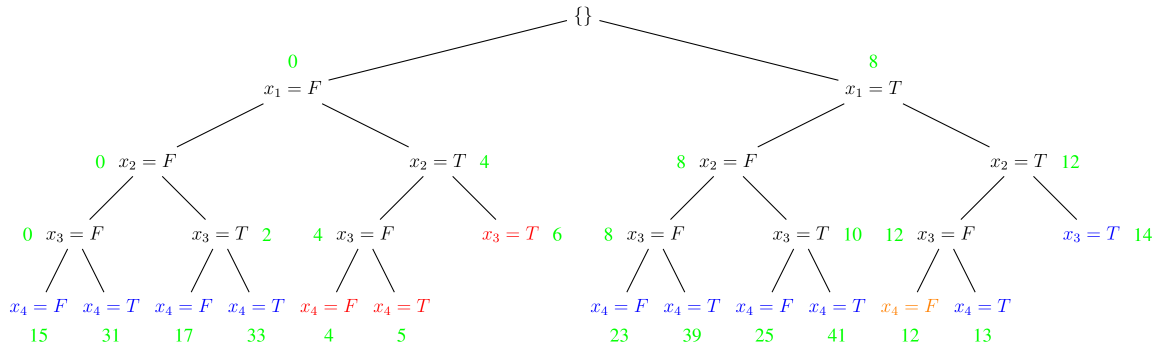


Figure 4-2: The state of the CDA* search after the next best full assignment is found. The g value for each node is displayed in green next to each node. The cost to go h is zero for all nodes. Nodes are expanded until the best full assignment $\{x_1 = T, x_2 = T, x_3 = F, x_4 = F\}$ is found (shown in orange). The nodes highlighted in blue represent the expansion frontier of the search (the nodes on the queue to be expanded). The nodes highlighted in red denote nodes found to violate the learned conflict.

This full assignment is then checked for consistency against the constraints. It is found to be consistent and is therefore the optimal solution to the example problem. Note how many nodes CDA* had to touch before finding the optimal: 26, including 10 nodes on the queue to be expanded at the end of the search. CDA* visited nearly every full assignment before finding the optimal.

4.4 A* with Bounding Conflicts

This section develops the A* with Bounding Conflicts (A*BC) algorithm. This algorithm efficiently solves OCSP problems by using *bounding conflicts* to guide the search away from suboptimal subspaces of the state space similar to how CDA* uses conflicts to guide the search away from infeasible subspaces. First, we provide an intuitive explanation as to how A*BC works. Next, we give an overview of how A*BC would solve the example. Then, we develop the A*BC algorithm in detail, starting with a formal definition of bounding conflicts, progressing to kernel methods for conflict resolution, and ending with a detailed description of A*BC.

4.4.1 A*BC Intuition

CDA* owes its improvement over constraint-based A* on OCSP problems to the candidate generator learning the structure of the problem during the search. Essentially, the conflicts represent a tighter heuristic that can be used in candidate generation. To better see how this works, we can reformulate the OCSP cost function as:

$$g'(\mathbf{Y}) = j(\mathbf{Y}) + g(\mathbf{Y}) \tag{4.7}$$

where g is the cost function from Definition 4.2 and j is the cost associated with violating the constraints in $\mathbf{C}$:

$$j(\mathbf{Y}) = \begin{cases} \infty, & \text{if all extensions of } \mathbf{Y} \text{ violate a constraint in } \mathbf{C} \\ 0, & \text{else} \end{cases} \tag{4.8}$$

Using this formulation of the cost function, it is easy to see how the candidate generator can incorporate knowledge of j (in the form of conflicts) into a heuristic function.

CDA*'s candidate generation process behaves similarly to a constraint based A* search with heuristic function h':

$$h'(y) = \begin{cases} \infty, & \text{if } \exists \gamma \in \Gamma s.t. \gamma \subset y \\ h(y), & else \end{cases} \tag{4.9}$$

where h is the original heuristic of cost g. From this formulation of the heuristic, we can see that the conflicts can be viewed as representing areas of the search space where the original heuristic is incorrect by an infinite amount.

The A*BC algorithm attempts to extend the inclusion of information from the testing phase into the generation phase beyond the simple binary information conveyed by conflicts. We do this by using conflicts to also record areas of the search space where the heuristic is weak and incorrect by some finite amount. These new conflicts are called *bounding conflicts* and the next section goes through an example of A*BC running on the example problem, showing how bounding conflicts are useful.

4.4.2 A*BC in a Nutshell

Now that we have shown how CDA*, a state of the art algorithm, would solve the example OCSP, we turn to describing the behavior of our proposed algorithm, A*BC. First, note how CDA* quickly learned the important piece of information that if x_1 is false, x_2 cannot be true. This allowed CDA* to skip over some search nodes that a non-conflict directed A* search would have normally expanded. However, note that CDA* spent quite a bit of time exploring the children of the $x_2 = F$ nodes, even though the full assignments in that region have a relatively high cost compared to the surrounding nodes. These areas were explored because the heuristic function of estimating the cost to go to always be zero is extremely optimistic in these areas.

A*BC is an algorithm that learns in which areas of the search space the heuristic function is particularly weak and then actively steers the search away from those areas. This will be accomplished by learning *bounding conflicts* while searching. Each bounding conflict describes a region of the search space using a partial assignment and a better estimate of the cost to go than the heuristic function h. These bounding conflicts are then used while searching to skip over the areas that would be explored only because the heuristic function was too optimistic.

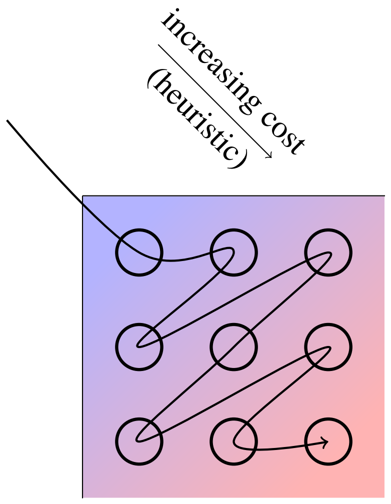
Figure 4-3 provides a conceptual picture of how the A*BC algorithm explores the search space. First, Figure 4-3a shows how an A^* search would explore the search space. It would start at the node with the lowest heuristic value and touch every node with a lower heuristic cost than the goal (lower-right corner of the search space). A*BC begins to explore at the node with the lowest heuristic, just like A^*(Figure 4-3b). However, it learns that the error between the true cost and heuristic cost is over a specified error threshold Δ. A*BC then generalizes the reason for the weak heuristic and learns it as a bounding conflict. This is shown in Figure 4-3c by the red area superimposed on the search space.

Now A*BC begins avoiding regions of the search space covered by bounding conflicts. This can also be seen in Figure 4-3c. The next best node according to the heuristic supplied to A*BC is covered by a bounding conflict, so A*BC skips it and explores the next best node not in a bounding conflict.

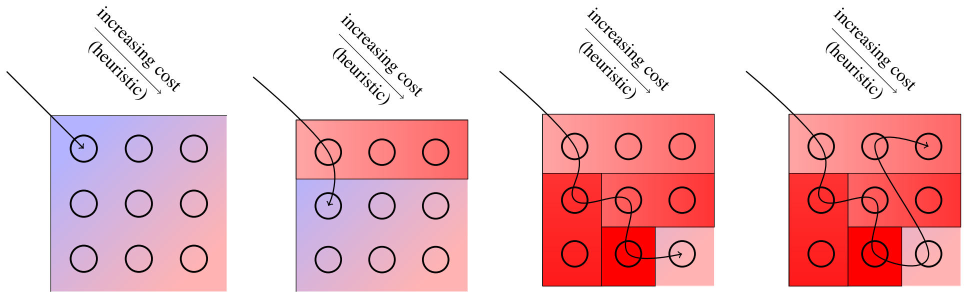
The search continues in this way until the optimal solution is found in Figure 4-3d. At this point in the search, the entire search space is explored or covered by bounding conflicts, however, A*BC may be requested to provide the second optimal solution. To do this, A*BC must relax a bounding conflict and start exploring in a part of the search space that was previously skipped. This is depicted in Figure 4-3e as A*BC begins to explore the area covered by the bounding conflict that has the lowest heuristic value (the top bounding conflict with the lightest shading of red).

This need to eventually explore areas with weak heuristics is what drives us to define a new type of conflict and algorithm other than the typical conflict directed algorithms. These typical conflict directed algorithms learn areas of the search space that are inconsistent and assume that any inconsistent part of the search space will never become consistent. This assumption means that these algorithms have no way to relax conflicts and start exploring the spaces that were skipped over previously.

Algorithm 4.2 provides an algorithmic description of A*BC that behaves as described. Now that we have described the concepts of A*BC, we examine A*BC's performance on



(a)



(b) (c) (d) (e)

Figure 4-3: This figure shows a conceptual view of the A* with Bounding Conflicts (A*BC) search algorithm. The circles represent nodes in a discrete search space and the color of the background represents the heuristic cost of the node, with red being higher than blue. Unlike [9], in this problem **all** states are feasible, and no hard constraints exist. Assume that the bottom right node is the true optimal node. A traditional A* search (top) will visit every single node with a heuristic cost lower than the heuristic cost of the optimal. However, in an A*BC search, a candidate will be generated, and its value estimated to a tighter bound. If the bound tightens significantly, the new bound is generalized to similar states, and the enumeration of those states is deferred until later in the search process. This is achieved by using bounding conflict learning to improve the heuristic estimate employed by A*. A*BC maintains completeness by exploring regions denoted by bounding conflicts when no better search node exists (bottom right).

Algorithm 4.2: A*-WITH-BOUNDING-CONFLICTS

```
Input: Optimal Constraint Satisfaction Problem, OCSP
Output: Returns the leading minimal cost solutions to OCSP
1  begin
2      $\Gamma[OCSP] \longleftarrow \{\ \}$;
3      Solutions[OCSP] $\longleftarrow \{\ \}$;
4      Nodes[OCSP] $\longleftarrow$ MAKE-QUEUE;
5      Visited[OCSP] $\longleftarrow \{\ \}$;
6      $\Delta \longleftarrow \Delta_0$;
7      repeat
8          current-node $\longleftarrow$ REMOVE-BEST(Nodes[OCSP]);
9          if MANIFEST-CONFLICT?(current-node, OCSP) then
10             UPDATE-NODE-COST(current-node, OCSP);
11         else if GOAL-NODE?(current-node, OCSP) then
12             Visited[OCSP] $\longleftarrow$ Visited[OCSP] $\cup$ current-node;
13             Solutions[OCSP] $\longleftarrow$ Solutions[OCSP] $\cup$ current-node;
14         else
15             Visited[OCSP] $\longleftarrow$ Visited[OCSP] $\cup$ current-node;
16             new-nodes $\longleftarrow$ CHILDREN-RESOLVING-CONFLICTS(current-node,
               OCSP);
17             foreach new $\in$ new-nodes do
18                 if EXTRACT-CONFLICT?(new-node, OCSP) then
19                     new-conflicts $\longleftarrow$ EXTRACT-CONFLICTS(new-node, OCSP);
20                     $\Gamma[OCSP] \longleftarrow$ new-conflicts $\cup\ \Gamma[OCSP]$;
21                 else
22                     Nodes[OCSP] $\longleftarrow$ ENQUEUE(Nodes[OCSP], new, $f$);
23                 end
24             end
25         end
26         $\Delta \longleftarrow$ UPDATE-DELTA-AND-RELAX(OCSP);
27     until Nodes[OCSP] is empty or TERMINATE?(OCSP);
28     return Solutions[OCSP];
29 end
```

the example design problem. The key takeaway from this will be to see how A*BC learns that any node containing the partial assignments $\{x_2 = F, x_4 = F\}$ or $\{x_2 = F, x_4 = T\}$ have a weak heuristic and are skipped until every better node has been expanded. For this example, we set the error bound Δ to be 10.

A*BC starts exactly the same as CDA* as it begins to explore the left side of the tree. The first difference occurs when the first full assignments are placed on the queue to be expanded. Both $\{x_1 = F, x_2 = F, x_3 = F, x_4 = F\}$ and $\{x_1 = F, x_2 = F, x_3 = F, x_4 = T\}$ are the placed on the queue to be expanded (Figure 4-4). The algorithm then determines that for each of these search nodes, the difference between the heuristic cost and the true cost (15 and 31, respectively) is greater than Δ The algorithm then learns a bounding conflict that says any assignment that contains $\{x_2 = F, x_4 = F\}$ will have a cost of at least 15 and any assignment containing $\{x_2 = F, x_4 = T\}$ will have a cost of at least 31.

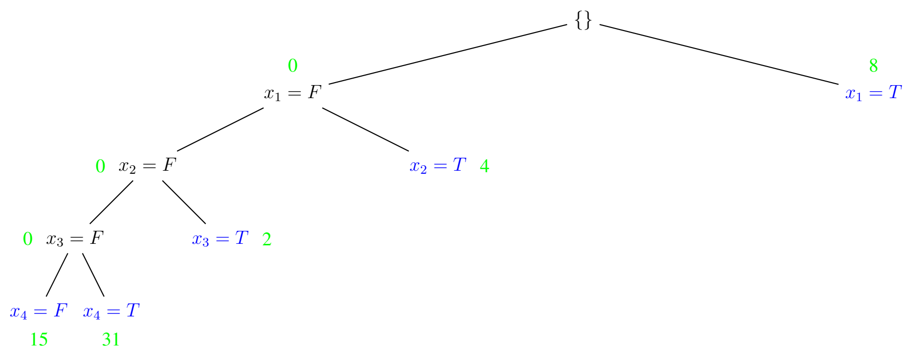


Figure 4-4: The first part of the A*BC search in the example problem. The blue nodes represent the fringe of the search. The g values of nodes are in green. At this point, the A*BC algorithm determines that every assignment containing $\{x_2 = F, x_4 = F\}$ will have a cost of at least 15 and every assignment containing $\{x_2 = F, x_4 = T\}$ will have a cost of at least 31.

Now that A*BC has learned this bounding conflict, the search begins to differ from CDA*'s search. Now, A*BC will look at the next node on the queue ($\{x_1 = F, x_2 = F, x_3 = T\}$) and extend it such that it resolves one of the learned conflicts. In this example, it resolves the $\{x_2 = F, x_4 = T\}$ conflict because it has the highest cost. It resolves the conflict by extending $\{x_1 = f, x_2 = F, x_3 = T\}$ to $\{x_4 = F\}$. This is shown in Figure 4-5.

The search then continues at the next best node on the queue. In this case, that is

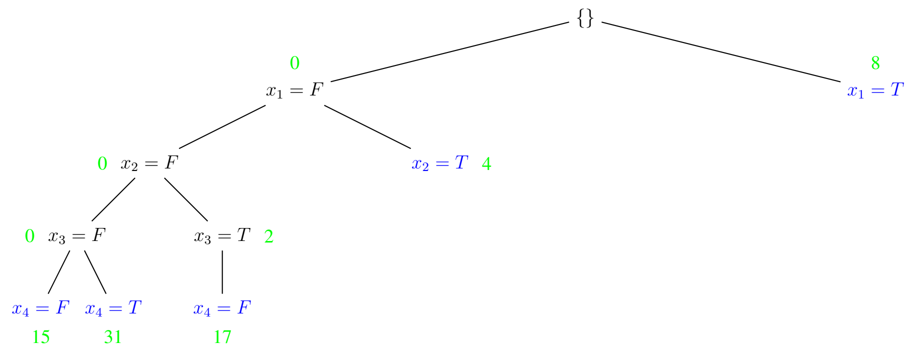


Figure 4-5: The second part of the A*BC search in the example problem. The blue nodes represent the fringe of the search. The g values of nodes are in green. The bounding conflict relating to $setx_2 = F, x_4 = T$ is resolved by expanding the best fringe node $\{x_1 = F, x_2 = F, x_3 = T\}$ to include $x_4 = F$.

the node corresponding to $\{x_1 = F, x_2 = T\}$. A*BC determine that this partial assignment already resolves every known conflict because it contains $\{x_2 = T\}$ and proceeds as normal until, as with CDA*, $\{x_1 = F, x_2 = T, x_3 = F, x_4 = F\}$ is determined to be the full assignment that minimizes the cost function. And, as before, the conflict $\{x_1 = F, x_2 = T\}$ is learned because of the hard constraints on the system. This step is shown in Figure 4-6.

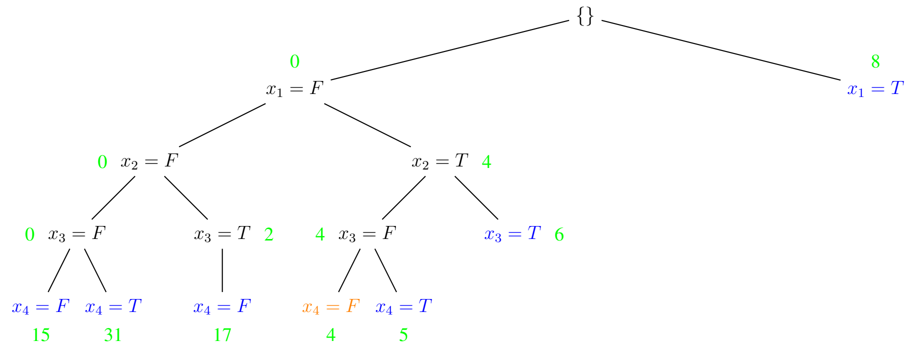


Figure 4-6: The third part of the A*BC search in the example problem. The blue nodes represent the fringe of the search. The g values of nodes are in green. The full assignment $\{x_1 = F, x_2 = T, x_3 = F, x_4 = F\}$ (in orange) is determined to minimize the cost function. However, it violates the system's hard constraints, so the conflict $\{x_1 = F, x_2 = T\}$ is learned.

As with CDA*, a number of the current fringe nodes are then determined to be infeasible because of the learned feasibility conflict. A*BC then examines the node corresponding to $\{x_1 = T\}$. This node resolves the learned feasibility conflict, but does not resolve any of the learned bounding conflicts. Instead of splitting on the x_2 variable as CDA* does, A*BC splits on the unresolved bounding conflict $\{x_2 = F, x_4 = T\}$. The result of this split is shown in Figure 4-7.

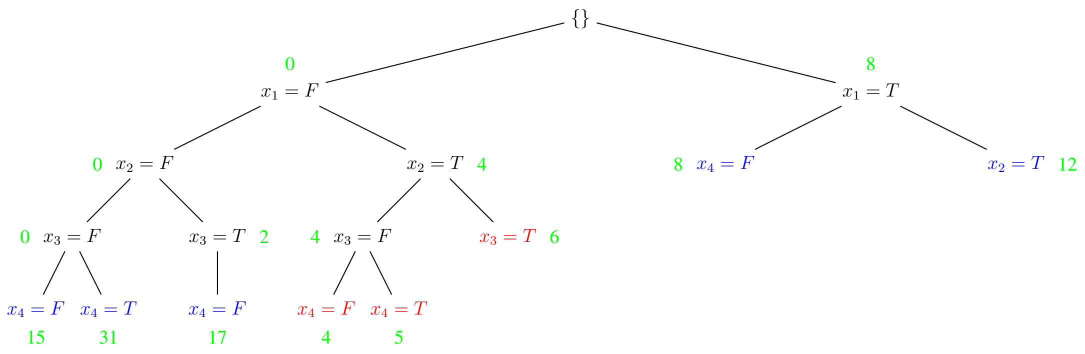


Figure 4-7: The fourth part of the A*BC search in the example problem. The blue nodes represent the fringe of the search. The g values of nodes are in green. The node corresponding to the partial assignment $\{x_1 = T\}$ is expanded to resolve the bounding conflict corresponding to $\{x_2 = F, x_4 = T\}$ by adding nodes corresponding to $\{x_1 = T, x_4 = F\}$ and $\{x_1 = T, x_2 = T\}$ to the search tree.

Now the next best node is determined to be $\{x_1 = T, x_4 = F\}$. However, this node does not resolve the bounding conflict corresponding to $\{x_2 = F, x_4 = F\}$, so that conflict is split on. The only way to resolve the conflict is to extend the partial assignment to include $\{x_2 = T\}$. This step is shown in Figure 4-8.

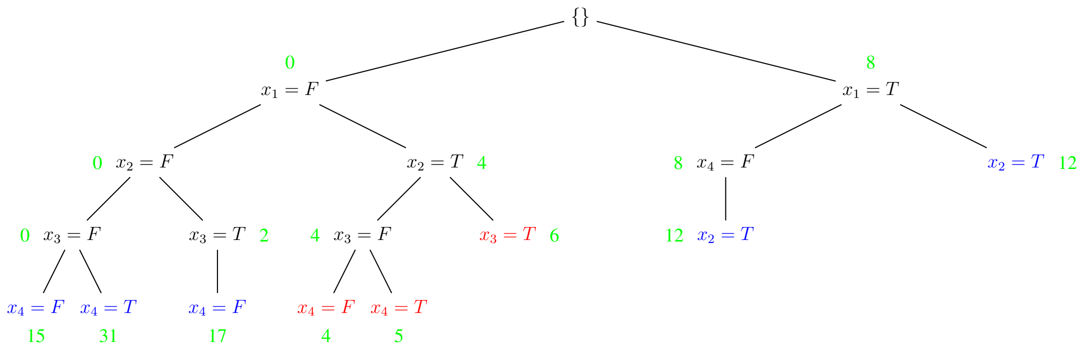


Figure 4-8: The fifth part of the A*BC search in the example problem. The blue nodes represent the fringe of the search. The g values of nodes are in green. The node corresponding to the partial assignment $\{x_1 = T, x_4 = F\}$ is expanded to resolve the bounding conflict corresponding to $\{x_2 = F, x_4 = F\}$ by adding the search node corresponding to $\{x_1 = T, x_4 = F, x_2 = T\}$ to the search tree.

Now, the best node on the fringe is the one corresponding to $\{x_1 = T, x_4 = F, x_2 = T\}$. This resolves every known feasibility and bounding conflict, so it is expanded by splitting on the only unassigned variable, x_3. This step is shown in Figure 4-9. The best node in the search tree now corresponds to the full assignment $\{x_1 = T, x_4 = F, x_2 = T, x_3 = F\}$. This full assignment satisfies all the hard constraints on the system, so it is determined to be the optimal solution of the example problem.

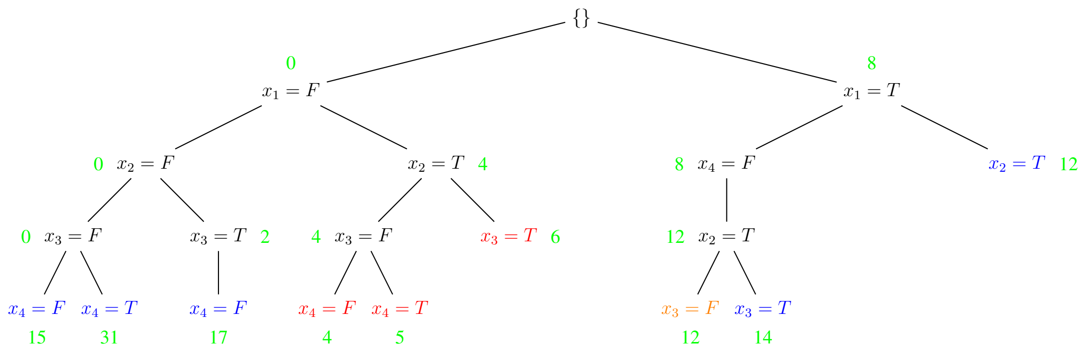


Figure 4-9: The sixth part of the A*BC search in the example problem. The blue nodes represent the fringe of the search. The g values of nodes are in green. The node corresponding to the partial assignment $\{x_1 = T, x_4 = F, x_2 = T\}$ is expanded by splitting on the only unassigned variable, x_3. The optimal solution to the example problem is shown in orange.

Now we are able to compare the performance of A*BC vs. CDA*. A*BC touched only 18 nodes with 5 nodes on the fringe at the end compared to CDA*'s 26 and 10 respectively. This increase in search efficiency is due to the fact that A*BC learned bounding conflicts early and then used them to skip over parts of the search space later on.

4.4.3 Bounding Conflicts

This section begins the formal description of the A*BC algorithm. We begin by describing the fundamental unit of the algorithm: bounding conflicts.

There are two main types of conflicts found in existing literature. The first is a feasibility conflict and is used in many constraint satisfaction and optimization algorithms to increase search efficiency. This is the type of conflict CDA* uses and it represents areas of the search space that are infeasible because they violate a conflict. This type of conflict is not useful for A*BC to use to learn areas with weak heuristics because those areas of the search space are still feasible. What is needed is a way to represent conflicts that record stronger heuristics as opposed to infeasibilities.

This is where the second type of conflicts comes into play. The concept of a *valued nogood* was introduced by Dago and Verfaillie in [2] as a way to improve a branch and bound search for valued constraint satisfaction problems. These valued nogoods record partial assignments and a lower bound on the optimal cost of any full assignment that contains the partial assignment.

In this thesis, we adopt the the definition of valued nogoods from [2] as the definition of a bounding conflict. The renaming is to maintain consistency with CDA* and further emphasize that the bounding conflict provides a better bound on the cost function.

Definition 4.5 (Bounding Conflict)**.** *Let $\mathbf{X}$ be a set of variables and let $c(x)$ be a cost function on the variables in $\mathbf{X}$. A bounding conflict γ is defined by the tuple $\langle z, \delta \rangle$ where z is a partial assignment to $\mathbf{X}$ such that any assignment y that extends z, ($z \subset y$) has a cost of at least δ, in other words, $g(y) \geq \delta$.*

Any assignment that extends the partial assignment z in conflict γ is said to manifest conflict γ and any assignment that does not extend z is said to resolve the conflict γ.

Note that conflicts are a specialized case of bounding conflicts where $\delta = \infty$.

4.4.4 Bounding Conflict Resolution

As discussed in Section 4.4.2, the A*BC algorithm alternates between resolving conflicts and searching based on the provided heuristic, both in best first order. The bounding conflict resolution phase is entered when the algorithm detects that the current best node, with value f does not resolve one or more known bounding conflicts with a bound greater than $f + \Delta$.

Because the node fails to resolve a bounding conflict with a bound greater than $f + \Delta$, it's possible that some extension will end up in a region of the search space that's known to have high error in the heuristic. In order to prevent this from happening, A*BC tries to resolve all bounding conflicts with a bound greater than $f + \Delta$. A*BC does this in a method similar to CDA* using constituent kernels.

A bounded kernel with bound δ' is a partial assignment to the problem variables such that all known bounding conflicts with bound δ' are resolved.

Definition 4.6 (Bounded Kernel)**.** *Let* Γ *represent a set of bounding conflicts* $\{\gamma_1, \gamma_2, \ldots\}$*.* k_Γ *is a bounded kernel with bound* δ' *if every extension* y *to* k_Γ *resolves every bounding conflict in* Γ *with a bound* $\delta \geq \delta'$*.*

Definition 4.7 (Constituent Bounded Kernel)**.** *Let* γ *be a bounding conflict with partial assignment* $z[\gamma]$ *and bound* $\delta[\gamma]$*.* k_γ *is a constituent bounded kernel with bound* δ *if for every extension* y *to* k_γ*,* y *resolves* γ *and no proper subset of* y *is also a constituent bounded kernel of* γ*.*

Using this definition, the FIND-CONSTITUENT-KERNELS algorithm can be designed to extract the set of all constituent kernels from a bounding conflict database. Such an algorithm is sketched in Algorithm 4.3. The ADD-TO-MINIMAL-SETS algorithm (Algorithm 4.4) is used to ensure that no constituent kernel is duplicated or is a subset of another constituent kernel.

Algorithm 4.3: FIND-CONSTITUENT-KERNELS

```
Input: Set of Bounding Conflicts $\Gamma$
begin
    constituents $\longleftarrow \{\ \}$;
    foreach $\gamma \in \Gamma$ do
        $K_\gamma \longleftarrow \{\}$;
        foreach $(x_i = v_{ij} \in z[\gamma])$ do
            $K_\gamma \longleftarrow K_\gamma \cup \{\{x_i = k_{ik}\} | v_{ik} \in D_{x_i}, v_{ik} \neq v_{ij}\}$;
        end
        constituents $\longleftarrow$ADD-TO-MINIMAL-SETS(constituents, $K_\gamma$);
    end
    return constituents;
end
```

A bounded kernel is a minimal set covering of the constituent bounded kernels for each conflict. Because a bounded kernel contains an element from each constituent bounded kernel, it is a partial assignment that is guaranteed to resolve all conflicts.

If a full assignment resolves conflict γ, it must contain at least one of the constituent bounded kernels of γ. This insight can then be used to construct an algorithm for testing if an assignment resolves all the known conflicts. This is done by calculating the set of

Algorithm 4.4: ADD-TO-MINIMAL-SETS

```
Input: A set $Set$ and an element to add to set $S$
begin
    foreach $E \in Set$ do
        if $E \subset S$ then
            return $Set$;
        else if $S \subset E$ then
            $Set \longleftarrow$ remove $E$ from $Set$;
        end
    end
    return $Set \cup \{S\}$;
end
```

constituent bounded kernels and then ensuring that the assignment being queried contains a constituent bounded kernel for each conflict. This behavior is described in Algorithm 4.5.

Algorithm 4.5: RESOLVES-CONFLICTS?

```
Input: node, OCSP
begin
    foreach $K_\gamma \in$ CONSTITUENT-KERNELS(OCSP) do
        if State[node] does not contain a kernel in $K_\gamma$ then return False;
    end
    return True;
end
```

Conversely, an algorithm to determine if an assignment manifests a conflict is described in Algorithm 4.6.

Algorithm 4.6: MANIFEST-CONFLICT?

```
Input: Search node to test, node, and OCSP
begin
    foreach $\gamma \in \Gamma$ do
        if $z[\gamma] \subset$ State(node) then return True;
    end
    return False;
end
```

4.4.5 Node Expansion

As seen in Section 4.4.2, the primary idea of A*BC is to expand search nodes in such a way that the search is actively steered away from areas known to have particularly bad bounding function heuristics. The CHILDREN-RESOLVING-CONFLICTS algorithm is responsible for determining how nodes are expanded. The key idea is to first determine if the the node to be expanded resolves all the known conflicts. If the parent node resolves all the known conflicts, then its children are guaranteed to as well, so CHILDREN-RESOLVING-CONFLICTS chooses to split on an unassigned variable. Otherwise, a conflict is chosen to split on and resolve. Algorithm 4.7 describes the CHILDREN-RESOLVING-CONFLICTS algorithm.

Algorithm 4.7: CHILDREN-RESOLVING-CONFLICTS

```
1  begin
2      new-nodes ⟵{ };
3      if RESOLVES-CONFLICTS?(current-node, OCSP) then
4          new-nodes ⟵EXPAND-VARIABLE(current-node, OCSP);
5      else
6          new-nodes ⟵EXPAND-CONFLICT(current-node, OCSP);
7      end
8      foreach new ∈ new-nodes do
9          if State(new) ∈ Visited[OCSP] then
10             new-nodes ⟵remove new from new-nodes;
11         end
12     end
13     return new-nodes;
14 end
```

As seen in Section 4.4.2, the A*BC search on the example problem would first try to expand the root node, consisting of the empty assignment. On line 3, the algorithm determines that all conflicts are resolved (as there are no conflicts yet), and then expands the children on line 4 using EXPAND-VARIABLE. Lines 8 - 12 then ensure that none of the children have been previously expanded.

The EXPAND-VARIABLE algorithm expands nodes exactly like A* does, it chooses a variable to split on and then returns all possible assignments to that variable. It is described by Algorithm 4.8. When given the node representing the empty assignment, this algorithm first determines that it is not a complete assignment. On line 5, it then selects a variable to split on. In the example, it chooses y_1 as it is the first and all the variables have the same size domain. Then on line 6, the algorithm determines all the extensions to the state of the node. In this case, the extensions are $y_1 = True$ and $y_1 = False$. Last, lines 8 - 10 generate nodes to represent the extensions.

Algorithm 4.8: EXPAND-VARIABLE

```
Input: node, $OCSP$
1  begin
2      if node is full assignment then
3          return { };
4      else
5          $x_i$ ⟵an unassigned variable in State[node] with smallest domain;
6          extensions ⟵$\{x_i = v_{ij} | v_{ij} \in D_i[OCSP]\}$;
7          children ⟵{ };
8          foreach extension ∈ extensions do
9              children ⟵children ∪ MAKE-NODE(extension, node);
10         end
11         return children;
12     end
13 end
```

CHILDREN-RESOLVING-CONFLICTS also generates children by splitting on a conflict as opposed to a variable.

4.4.6 Conflict Extraction

Ultimately, the efficiency of the A*BC algorithm depends on the ability to extract bounding conflicts. This task is provided by the EXTRACT-CONFLICTS algorithm and its implementation is application specific. Any implementation of EXTRACT-CONFLICTS is assumed to satisfy two requirements. First, the bounding conflict must be correct and have no extension to the partial assignment with utility greater than the bounds in the conflict. Second, the bound on the bounding conflict (δ) must be higher than the conflict boundary Δ.

4.4.7 Conflict Relaxation

It is possible that the optimal solution (or next optimal solution) is within one of the areas marked by a bounding conflict. This can arise when the bounding function is poor over the

Algorithm 4.9: EXPAND-CONFLICT

```
Input: node, OCSP
begin
    if RESOLVES-CONFLICTS?(node, OCSP) then
        return { };
    end
    else
        $K_\gamma \longleftarrow$ The smallest set in CONSTITUENT-KERNELS(OCSP) such that no
        kernel in the set is contained in State[node];
        extensions
        $\longleftarrow \{x_i = v_{ij} | \{x_i = v_{ij}\} \in K_\gamma, x_i = v_{ij}$ is consistent with State[node]$\}$;
        children $\longleftarrow$ { };
        foreach extension in extensions do
            children $\longleftarrow$ children $\cup$ MAKE-NODE(extension, node);
        end
        return children;
    end
end
```

entire state space, resulting in bounding conflicts being continuously extracted. In order to maintain completeness in these cases, bounding conflicts must be periodically relaxed. This section describes when and how the bounding conflicts must be relaxed.

A bounding conflict γ contains the quantity $\delta[\gamma]$ which describes the optimistic best cost for any full assignment within the subspace defined by γ. The head of A*BC's node list is the node with the lowest optimistic cost. If the f value of the head of the queue ever exceeds $\delta[\gamma]$, then it is possible that $z[\gamma]$ will expand to a goal node with lower cost than the head of the queue. When this happens, γ must be relaxed so that A*BC can start exploring that subspace.

In order to relax conflict γ, it must first be removed from the conflict database. This action ensures that the nodes already on the search queue will be allowed to explore the previously blocked off region. However, it is possible that all the nodes on the search queue have been generated to resolve the conflict being relaxed. If this is the case, then the nodes on the queue will never explore the newly opened subspace! In order to ensure that the newly opened subspace is explored, then $z[\gamma]$ is also added to the queue to be expanded.

Algorithm 4.10: UPDATE-DELTA-AND-RELAX

```
  Input: $OCSP$
  Output: new $\Delta$
1 begin
2     foreach $\gamma \in \Gamma[OCSP]$ do
3         if $f[OCSP]$(Head(Nodes[OCSP])) $\geq \delta[\gamma]$ then
4             $\Gamma[OCSP] \longleftarrow$Remove $\gamma$ from $\Gamma[OCSP]$;
5             push $z[\gamma]$ onto Nodes$[OCSP]$;
6         end
7     end
8     return UPDATE-CONFLICT-BOUNDARY($OCSP$);
9 end
```

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

State Estimation of cPHAs Using A*BC

This chapter describes *Bones*, our implementation of an approximate hybrid state estimator that uses the A*BC algorithm to perform the best first enumeration over operating modes. We first describe the problem statement and provide an overview of classical recursive filtering. Next, we provide the recursive hybrid filtering update equations. Then, we describe the search problem the estimator must solve at every time step. Last, we provide the heuristic used by A*BC and the method for extracting bounding conflicts.

5.1 Bones Overview

Bones is a recursive filter for hybrid discrete/continuous systems. The inputs to Bones are:

- a model of the system being estimated in the form of a cPHA (Definition 2.2), $\mathcal{CA}$,
- a time series of control inputs to the cPHA, $\mathbf{u}_{(1:T)}$,
- and a time series of observations made of the cPHA, $\mathbf{y}_{c,(1:T)}$.

The outputs of Bones are:

- a probability distribution over the mode of the system, $p(\mathbf{X}_{d,(T)}|\mathbf{y}_{c,(1:T)})$
- and a probability distribution over the continuous state of the system for every mode in the previous distribution, $p(\mathbf{X}_{c,(T)}|\mathbf{x}_{d,(T)}, \mathbf{y}_{c,(1:T)}) \quad \forall \mathbf{x}_{d,(T)} \in \mathbf{X}_{d,(T)}$.

While Bones outputs estimates for time T, internally it tracks the probability mass function for complete mode trajectories $p(\mathbf{X}_{d,(0:T)}|\mathbf{y}_{c,(1:T)})$ and the corresponding continuous estimates $p(\mathbf{X}_{c,(T)}|\mathbf{x}_{d,(0:T)}, \mathbf{y}_{c,(1:T)}) \quad \forall \mathbf{x}_{d,(0:T)} \in \mathbf{X}_{d,(0:T)}$. This is done to produce a more accurate approximation of the continuous probability estimates by preventing the continuous estimates for any mode trajectory ending in the same mode from being mixed together.

At every time step, Bones takes the probability distributions from the previous time step $(t-1)$ and uses them to calculate the probability distributions at time step t. This is accomplished by searching for the system transitions from the modes in $\mathbf{X}_{d,(0:t-1)}$ that result in the largest probability mass for the successor modes in $p(\mathbf{X}_{d,(0:t)}|\mathbf{y}_{c,(0:t)})$ using a generate and test approach. First, mode candidates are generated that maximize $p(\mathbf{X}_{d,(0:t)}|\mathbf{y}_{c,(0:t-1)})$. Then, a hybrid estimate is calculated

5.2 Review of Bayesian Filtering

Recursive Bayesian filtering is the underlying principal used by a large number of state estimation algorithms. At its core is the idea that as time progresses, new information can be incorporated into the belief state of the system incrementally. Updating the belief state between two time slices is done in a two step fashion. The first step is known as *propagation* where the belief state is projected forward in time using the system dynamics. The second step is known as *assimilation* and incorporates observations of the system to update the belief state.

If $\mathbf{X}_{(k)}$ represents the system's state at time k and $\mathbf{y}_{(k)}$ represents observations of the system at time k, the recursive propagation relation can be generally described by:

$$b_{\bullet}(\mathbf{X}_{(k)}) = p(\mathbf{X}_{(k)}|\mathbf{y}_{(0:k-1)}) = \int_{\mathbf{X}_{(k-1)}} p(\mathbf{X}_{(k)}|\mathbf{x}_{(k-1)}) b(\mathbf{x}_{(k-1)}) \tag{5.1}$$

Where $b_{\bullet}(\mathbf{X}_{(k)})$ represents the belief state of the system before the observation at time k is assimilated. The assimilation equation can be generally represented by:

$$b(\mathbf{X}_{(k)}) = p(\mathbf{X}_{(k)}|\mathbf{y}_{(0:k)}) = \frac{p(\mathbf{y}_{(k)}|\mathbf{X}_{(k)})b_{\bullet}(\mathbf{X}_{(k)})}{p(\mathbf{y}_{(k)})} = \frac{p(\mathbf{y}_{(k)}|\mathbf{X}_{(k)})b_{\bullet}(\mathbf{X}_{(k)})}{\int_{\mathbf{X}_{(k)}} p(\mathbf{y}_{(k)}|\mathbf{x}_{(k)})b_{\bullet}(\mathbf{x}_{(k)})} \quad (5.2)$$

There exist special forms of these equations for purely discrete and purely continuous systems. Hybrid systems involve both discrete and continuous variables and interactions between them, as such we will review the special forms of the Bayesian filter equations which are used by a hybrid filter.

5.2.1 Discrete Bayesian Filtering

As discussed in Chapter 2, the modes of a PHA and their transitions can be modeled with a discrete hidden Markov model (HMM). The states of the HMM are the modes of the component $\mathcal{X}_d$ and the transitions are contained in T. Figure 5-1 shows an example HMM for a component with three modes and unguarded transitions.

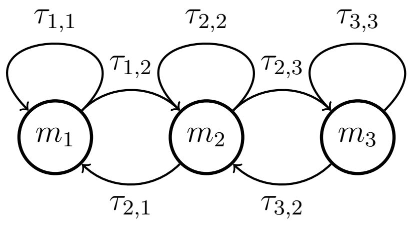


Figure 5-1: Hidden Markov model with three states and the associated transition probabilities.

The specialized form of Equation 5.1 for discrete system propagation is given by Equation 5.3. For every state, the probability of transitioning into that state from every other state is calculated and then summed to determine the probability of entering that state from any other state.

$$b_{(\bullet k)}[m_i] = \sum_{m_j \in \mathcal{X}_d} P_{\mathcal{T}}(m_i|u_{d,(k-1)}, m_j) b_{(k-1)}[m_j] \quad (5.3)$$

$P_{\mathcal{T}}(m_i|u_{d,(k-1)}, m_j)$ represents the probability of transitioning into mode m_i from mode m_j given the control input $u_{d,(k-1)}$.

After the propagation step, the observations from the system are assimilated to form the posterior distribution at time k. The specialized form of Equation 5.2 for this process is described by Equation 5.4.

$$b_{(k)}[m_i] = \frac{b_{(\bullet k)}[m_i] P_{\mathcal{O}}(y_{(k)}|m_i)}{\sum_{m_j \in \mathcal{X}_d} b_{(\bullet k)}[m_j] P_{\mathcal{O}}(y_{(k)}|m_j)} \tag{5.4}$$

$P_{\mathcal{O}}(y_{(k)}|m_i)$ represents the likelihood of observing $y_{(k)}$ in mode m_i.

5.2.2 Continuous Bayesian Filtering

A standard way of estimating the state of a system with continuous state is to use a Kalman filter or one of its variants (notably the extended Kalman filter (EKF) or unscented Kalman filter (UKF)). In these systems, the evolution of state over time is modeled using differential equations that can be integrated, resulting in difference equations of the form:

$$\mathbf{x}_{(k)} = f_{DE}(\mathbf{x}_{(k-1)}, \mathbf{u}_{(k-1)}) + \nu_x \tag{5.5}$$

$$\nu_x \sim \mathcal{N}(0, \mathbf{Q}) \tag{5.6}$$

where $\mathbf{x}_{(k)}$ is a vector describing the state of the system at time k and ν_x is a zero-mean Gaussian disturbance with covariance $\mathbf{Q}$.

Without loss of generality, we will assume an EKF is used to estimate continuous state in the rest of this chapter. The EKF also uses a two step propagation then assimilation approach. The standard EKF equations for propagation are:

$$\hat{\mathbf{x}}_{c,(\bullet k)} = f_{DE}(\hat{\mathbf{x}}_{c,(k-1)}, \mathbf{u}_{(k-1)}) \tag{5.7}$$

$$\mathbf{A}_{(k-1)} = \left. \frac{\partial f}{\partial \mathbf{x}_c} \right|_{\hat{\mathbf{x}}_{c,(k-1)}, \mathbf{u}_{(k-1)}} \tag{5.8}$$

$$\mathbf{P}_{(\bullet k)} = \mathbf{A}_{(k-1)} \mathbf{P}_{(k-1)} \mathbf{A}_{(k-1)}^T + \mathbf{Q} \tag{5.9}$$

where $\hat{\mathbf{x}}_{c,(\bullet k)}$ represents the mean of the estimate before data assimilation at time k, $\hat{\mathbf{x}}_{c,(k-1)}$ represents the mean of the estimate at time $(k-1)$, $\mathbf{P}_{(k-1)}$ represents the estimate covariance at time $(k-1)$, $\mathbf{P}_{(\bullet k)}$ represents the estimate covariance before data assimilation at time k, and $\mathbf{Q}$ represents the effect of Gaussian random noise on the state variables.

After the propagation step, the estimated observations and covariance in the observation estimate are calculated using the equations:

$$\mathbf{r}_{(k)} = \mathbf{y}_{c,(k)} - g(\hat{\mathbf{x}}_{c,(\bullet k)}, \mathbf{u}_{(k)}) \tag{5.10}$$

$$\mathbf{C}_{(k)} = \left.\frac{\partial g}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{c,(\bullet k)}, \mathbf{u}_{(k)}} \tag{5.11}$$

$$\mathbf{S}_{(k)} = \mathbf{C}_{(k)} \mathbf{P}_{\bullet k} \mathbf{C}_{(k)}^T + \mathbf{R} \tag{5.12}$$

where $\mathbf{r}_{(k)}$ represents the residual between the predicted and observed observations, $\mathbf{S}_{(k)}$ represents the covariance in the predicted sensor readings, and $\mathbf{R}$ represents the covariance of the Gaussian noise affecting the sensors.

Once the predicted observations, the observation residual, and observation covariance are calculated, the data is assimilated into the state estimate. The standard EKF equations for data assimilation are:

$$\mathbf{K}_{(k)} = \mathbf{P}_{(\bullet k)} \mathbf{C}_{(k)}^T \mathbf{S}_{(k)}^{-1} \tag{5.13}$$

$$\hat{\mathbf{x}}_{c,(k)} = \hat{\mathbf{x}}_{c,(\bullet k)} + \mathbf{K}_{(k)} \mathbf{r}_{(k)} \tag{5.14}$$

$$\mathbf{P}_{(k)} = \left(\mathbf{I} - \mathbf{K}_{(k)} \mathbf{C}_{(k)}\right) \mathbf{P}_{(\bullet k)} \tag{5.15}$$

5.3 State Estimation as Best-First Enumeration

Hofbaur and Williams [5] showed that the update equations for the belief state of a cPHA consist of a propagation equation where the belief state at the previous time step is projected forward in time using the cPHA model (Equation 5.16) and a data assimilation equation where the latest system observations are incorporated into the belief state (Equation 5.17).

$$h_{(\bullet t)}[\hat{\mathbf{x}}_i] = P_T(m_i | \hat{\mathbf{x}}_{j,(t-1)}, \mathbf{u}_{d,(t-1)}) h_{(t-1)}[\hat{\mathbf{x}}_j] \tag{5.16}$$

$$h_{(t)}[\hat{\mathbf{x}}_i] = \frac{h_{(\bullet t)}[\hat{\mathbf{x}}_i] P_O(\mathbf{y}_{c,(t)} | \hat{\mathbf{x}}_{i,(t)}, \mathbf{u}_{c,(t)})}{\sum_j h_{(\bullet t)}[\hat{\mathbf{x}}_j] P_O(\mathbf{y}_{c,(t)} | \hat{\mathbf{x}}_{j,(t)}, \mathbf{u}_{c,(t)})} \tag{5.17}$$

These update equations show that, unlike IMM techniques, the belief states containing the same operating mode are not mixed together during the propagation step. Instead, these equations are used to track trajectories of the system's operating mode. For even moderate numbers of components and time steps, the total number of possible trajectories the system can take becomes intractable to calculate h for every trajectory. To deal with this, Hofbaur and Williams provided an algorithm that uses an A* search to find the hybrid state estimates that maximize h at every time step.

In Equation 5.16, P_T represents the probability of transitioning into mode m_i given the state of the system. In a cPHA, the mode transitions for each component are conditionally independent given the state of the system, so the transition function is simply a multiplication of the transition probabilities for each component.

$$P_T(m_i|\hat{\mathbf{x}}_{j,(t-1)}, \mathbf{u}_{d,(t-1)}) = \prod_k P_T(m_{i,k}|\hat{\mathbf{x}}_{j,(t-1)}, \mathbf{u}_{d,(t-1)}) \tag{5.18}$$

Where $m_{i,k}$ is the mode of the k^{th} component in estimate i.

In Equation 5.17, P_O represents the observation likelihood. Based on prior research, [5], calculates the observation likelihood using a non-normalized Gaussian distribution using the sensor covariance $\mathbf{S}_{(k)}$ and residual $\mathbf{r}_{(k)}$ from a Kalman filter. The reason for not normalizing is to prevent the estimator from being biased toward modes with lower measurement noise. P_O can be written as:

$$P_O(\mathbf{y}_{c,(t)}|\hat{\mathbf{x}}_{i,(t)}, \mathbf{u}_{c,(t)}) = e^{-\mathbf{r}_{(k)}^T \mathbf{S}_{(k)}^{-1} \mathbf{r}_{(k)}} \tag{5.19}$$

5.4 Search for Best Modes

At a high level, Bones can be viewed as a generate and test algorithm. It relies on a candidate generator (A*BC) to output a candidate mode trajectory that looks promising based on an upper bound to the trajectory's posterior probability mass. Bones then tests the candidate to determine its true posterior probability mass. Information from this test, such as a set of modes that are unlikely to produce the observations, is then fed back into the generator to make the generator more efficient when producing candidates. This process is shown in Algorithm 5.1.

Algorithm 5.1: BONES

```
1  begin
2      $\theta \longleftarrow \emptyset$;
3      $\Theta \longleftarrow \emptyset$;
4      $h \longleftarrow \infty$;
5      Initialize A*BC search;
6      while Candidates remaining && $|\Theta| < k$ do
7          $\hat{\mathbf{x}}_d, h \longleftarrow$ NEXT-BEST-CANDIDATE($\mathcal{CA}$);
8          $\hat{\mathbf{x}}_c, \hat{\mathbf{P}}, \hat{b} \longleftarrow$ BONES-TEST($\hat{\mathbf{x}}_d$, $\mathcal{CA}$);
9          $\hat{\mathbf{x}} \longleftarrow \langle \hat{\mathbf{x}}_d, \hat{\mathbf{x}}_c, \hat{\mathbf{P}} \rangle$;
10         $\theta \longleftarrow$ INSERT-AND-SORT($\theta, \hat{\mathbf{x}}, \hat{b}$);
11         while $\hat{b}$[HEAD($\theta$)] $\geq h$ do
12             $\Theta \longleftarrow$ APPEND($\Theta$, POP($\theta$));
13         end
14         if EXTRACT-BOUNDING-CONFLICT?($h, \hat{b}$) then
15             EXTRACT-BOUNDING-CONFLICT($\hat{\mathbf{x}}, h, \hat{b}$);
16         end
17     end
18 end
```

This algorithm uses A*BC as the candidate generator, and the test algorithm, shown in Algorithm 5.2 is simply running a Kalman filter on the generated mode estimate and calculating the observation likelihood.

Algorithm 5.2: BONES-TEST

```
   Input: $\hat{\mathbf{x}}_d$, $\hat{\mathbf{x}}_{c,(k-1)}, \mathbf{P}_{(k-1)}, \mathbf{y}_c$, $\mathcal{CA}$
1  begin
2      $\mathbf{Q} \longleftarrow$ GET-Q-MATRIX($\hat{\mathbf{x}}_d$, $\mathcal{CA}$);
3      $\mathbf{R} \longleftarrow$ GET-R-MATRIX($\hat{\mathbf{x}}_d$, $\mathcal{CA}$);
4      $\hat{\mathbf{x}}_{c,(k)}, \mathbf{P}_{(k)} \longleftarrow$ KALMAN-FILTER($\hat{\mathbf{x}}_{c,(k-1)}, \mathbf{P}_{(k-1)}, \mathbf{Q}, \mathbf{R}, \mathbf{y}_c$);
5      $\mathbf{r}, \mathbf{S} \longleftarrow$ Kalman residual and covariance;
6      $b \longleftarrow e^{-\mathbf{r}^T \mathbf{S}^{-1} \mathbf{r}}$;
7      return $\hat{\mathbf{x}}_{c,(k)}, \mathbf{P}_{(k)}, b$;
8  end
```

5.5 Extracting Conflicts

It is worthwhile at this point to briefly mention how one can extract conflicts in the hybrid estimation case. The simplest method, and the one used by this paper, is to repeatedly remove a component mode assignment from the mode vector and use the BONES-TEST algorithm to determine if the bound Δ is still violated. If the bound is still violated, the assignment that was removed is put back into the total mode assignment, otherwise it is left out. This guarantees that a minimal conflict will be found. However, this algorithm for extracting conflicts is not particularly efficient and other approaches may exist that can better exploit the structure of the problem.

Chapter 6

Results and Summary of Contributions

In this chapter, we discuss preliminary results of applying the search and estimation algorithms described in this thesis to static estimation of a fluid system. Additionally, we review the contributions of the thesis and suggest areas of future research.

6.1 Results

The A*BC algorithm and best-first enumeration algorithm for static hybrid estimation described in Chapters 4 and 3, respectively, were implemented in Lisp and tested using a fluid system simulator provided by Johns Hopkins University Applied Physics Laboratory (APL). The simulated system contains four water pumps, six load devices, a flow sensor attached to each pump and load, and 26 valves. A picture of the real (non-simulated) system can be seen in Figure 6-1.

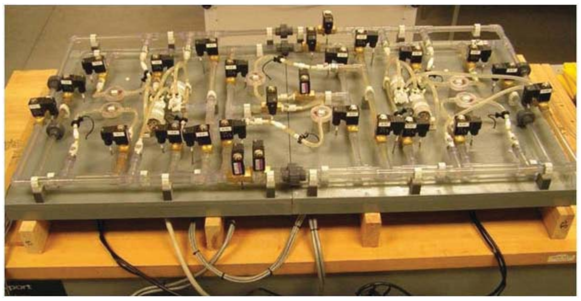


Figure 6-1: Real world analog of the simulated fluid system. Courtesy of APL.

The fluid simulator is written in Matlab and connects to the state estimator using a

TCP/IP connection. Both the estimator and simulator were executed on a 2.6 GHz dual-core Intel Core i5 processor with 8Gb of RAM. The testing process consisted of first defining a prior distribution over the modes of the fluid system that favored the configuration the system should be in, as it would be commanded by a separate controller, and assigned equal likelihood to each component failing. Then, a number of faults, single and double, were injected into the simulation. The estimator then used the observed flow sensor data to estimate the actual state of the system.

In all single fault scenarios, the static estimation algorithm identified the correct fault, within the limits of the observability of the system. Due to the low amount of sensors placed throughout the system, there are many different failure scenarios that result in the same observations. As such, several different failure scenarios were output by the state estimator, each with equal likelihood, and the true fault was contained within that set. 100% of the single fault scenarios were discovered with four seconds.

The results from the double fault scenarios were less promising. When different parts of the system were isolated from each other (i.e., pumps were separated from everything else except a single load by closed valves) and the faults happened in different parts, the algorithm again identified the correct faults 100% of the time.

However, when the system was not well isolated, the estimator sometimes never finished calculating an estimate within a reasonable amount of time (60s). It is unknown at this time if this was due to modeling errors, incorrect settings of the conflict boundary Δ in the A*BC algorithm, a bug in the implementation, or something else entirely.

6.2 Contributions and Future Research

This thesis presented a novel search algorithm, A^* with Bounding Conflicts, for solving Optimal Constraint Satisfaction Problems. A*BC uses a continuous analog of feasibility conflicts to learn a tighter heuristic on the true cost of the OCSP as it is searching. This learning process allows A*BC to delay expanding areas of the search space with a weak heuristic until much later when compared to a vanilla constraint-based A^* algorithm.

Additionally, this thesis described two uses of the A*BC algorithm: as a candidate generator for static hybrid estimation and as a candidate generator for dynamic hybrid estimation.

While A*BC performs extremely well on toy problems (as evidenced by the example in Chapter 4) and problems where the effects of changing the mode of a single component are limited (such as the fluid system simulation with isolated pump/load groups), it has not yet been demonstrated to improve the search efficiency on more complicated problems. Further research is needed in this area, especially with a simulator or other provider of truth data that has the ability to stochastically simulate the system, both in terms of process and sensing noise, and component failures.

Additionally, for large systems, the conflict extraction process can be extremely time consuming and inefficient, requiring in the worst case a number of calls to a Kalman filter or quadratic program solver equal to the number of components in the system. In the future, the conflict extraction process should be augmented with analysis systems that can separate the system being estimated into connected groups in order to focus the conflict extraction process. A possible system for doing this with cPHAs is described in [4].

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix A

Common Algorithms

Algorithm A.1: GET-Q-MATRIX

```
   Input: Mode $\mathrm{x}_d$, and cPHA $\mathcal{CA}$
1  begin
2      $\mathrm{Q} \longleftarrow []$;
3      foreach $x_d[i] = v_{i,j} \in \mathbf{x}_d$ do
4          $\mathrm{Q} \longleftarrow \begin{bmatrix} \mathbf{Q} & \mathbf{0} \\ \mathbf{0} & \mathcal{CA}[N_x[i]](v_{i,j}) \end{bmatrix}$;
5      end
6      return $\mathrm{Q}$;
7  end
```

Algorithm A.2: GET-R-MATRIX

```
   Input: Mode $\mathrm{x}_d$, and cPHA $\mathcal{CA}$
1  begin
2      $\mathrm{R} \longleftarrow []$;
3      foreach $x_d[i] = v_{i,j} \in \mathbf{x}_d$ do
4          $\mathrm{R} \longleftarrow \begin{bmatrix} \mathrm{R} & \mathbf{0} \\ \mathbf{0} & \mathcal{CA}[N_y[i]](v_{i,j}) \end{bmatrix}$;
5      end
6      return $\mathrm{R}$;
7  end
```

Algorithm A.3: GET-ALGEBRAIC-CONSTRAINTS

```
   Input: Mode $\mathbf{x}_d$, and cPHA $\mathcal{CA}$
1  begin
2      $C \longleftarrow []$;
3      foreach $x_d[i] = v_{i,j} \in \mathbf{x}_d$ do
4          $C \longleftarrow$ APPEND($C, \mathcal{CA}[F_{AE}[i]](v_{i,j})$);
5      end
6      return $C$;
7  end
```

Algorithm A.4: GET-OBSERVATION-CONSTRAINTS

```
   Input: Mode $\mathbf{x}_d$, and cPHA $\mathcal{CA}$
1  begin
2      $C \longleftarrow []$;
3      foreach $x_d[i] = v_{i,j} \in \mathbf{x}_d$ do
4          $C \longleftarrow$ APPEND($C, \mathcal{CA}[F_{OBS}[i]](v_{i,j})$);
5      end
6      return $C$;
7  end
```

Algorithm A.5: GET-DIFFERENCE-CONSTRAINTS

```
   Input: Mode $\mathbf{x}_d$, and cPHA $\mathcal{CA}$
1  begin
2      $C \longleftarrow []$;
3      foreach $x_d[i] = v_{i,j} \in \mathbf{x}_d$ do
4          $C \longleftarrow$ APPEND($C, \mathcal{CA}[F_{DE}[i]](v_{i,j})$);
5      end
6      return $C$;
7  end
```

Algorithm A.6: GET-STATIC-CONSTRAINTS

```
   Input: cPHA $\mathcal{CA}$
1  begin
2      return $\mathcal{CA}$[S];
3  end
```

Bibliography

[1] Henk A. P. Blom and Yaakov Bar-Shalom. The interacting multiple model algorithm for systems with markovian switching coefficients. In *IEE Transactions on Automatic Control*, August 1988.

[2] Pierre Dago and Gerard Verfaillie. Nogood recording for valued constraint satisfaction problems. In *In ICTAI*, pages 132–139, 1996.

[3] Cutset Decomposition and Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[4] Michael Hofbaur and Brian Williams. Hybrid estimation of complex systems. In *IEE Transactions on Systems, Man, and Cybernetics*, October 2004.

[5] Michael W. Hofbaur and Brian C. Williams. Mode estimation of probabilistic hybrid systems. In *In Intl. Conf. on Hybrid Systems: Computation and Control*, pages 253–266. Springer Verlag, 2002.

[6] S. Narasimhan and L. Brownston. Hyde–a general framework for stochastic and hybrid model-based diagnosis. In *Proc. 18th International Workshop on Principles of Diagnosis (DX'07), Nashville, USA*, pages 162–169. Citeseer, 2007.

[7] Martin Sachenbacher and Brian C. Williams. Conflict-directed a * search for soft constraints, 2006.

[8] Thomas Schiex and Grard Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *International Journal of Artificial Intelligence Tools*, 3:48–55, 1993.

[9] Brian C. Williams and Robert J. Ragno. Conflict-directed a* and its role in model-based embedded systems. *Discrete Appl. Math.*, 155:1562–1595, June 2007.