Self-Monitoring, Self-Diagnosing Systems

What Happened?

Brian Williams
June 12th, 2017

Slide Contributions:
Steve Levine
David Wang
Brian Williams
Some Related Papers

imageScienceTargets(Rover1, Rover2)
Parallel{
    Sequence{
        [5,10] Rover1.goto(p4);
        [5,10] Rover1.goto(p5);
        Choice{
            [2,5] Rover1.imageTargets1(); [5,10] Rover1.goto(p3);
            [5,10] Rover1.goto(p3); [2,5] Rover1.imageTargets1();
        }
    }
    Sequence{
        [5,10] Rover2.goto(p1);
        Choice{
            [5,10]Rover2.imageTargets2(); [2,5] Rover2.goto(p2);
            [2,5] Rover2.goto(p2); [5,10]Rover2.imageTargets2();
        }
        [5,10] Rover2.goto(p3);
    }
}

Program A ::= 
    prim_action(args) | 
    [lb, ub] A | 
    Sequence {A1; A2; ... } | 
    Parallel {A1; A2; ... } | 
    Choose { [with reward R1] A1; [with reward] R2 A2; ... }
Recall: Program Executives Monitor Goals and Make Online Choices

- **Plan**: Make optimal, consistent program choices.
- **Execute**: Dispatch and monitor resulting plan.

**Executive**

- **Task Planning** (make choices)
- **Plan Execution**
  - Monitoring
    - sub-goals
    - activities
    - resources
  - T2 (Williams)
  - Activity Scheduling & Dispatch
  - T3 (Hunsberger)

6/12/2017 T2 self-monitoring, self-diagnosing systems
Recall: A “Suspicious” Executive Closes the Loop on Goals

- **Monitors**: detects as soon as sub-goals fail, actions break.
- **Adapts**: fixes the problem.
Recall: Self-Monitoring Systems

Detects failing actions and plans, due to:

- Anomalies in relevant state (sub-goals).
- Faulty hardware.
We can create different programming models by combining different methods.

Flexible Time

+ Execution monitoring

Flexible Time (w Uncertainty)

and

+ Diagnosis and Reconfiguration

Discrete Choice and Search
Outline

• Programs that Monitor State

• Programs that Self-Diagnose (opt)
Outline

• Programs that Monitor State
  – Sub-goal monitoring
  – Model-based diagnosis and mode estimation
  – Programs that Self-Diagnose (opt)
Volcano eruption!

Program (Plan) is a sequence of actions in RMPL

```java
method run() {
    sequence {
        uav.launch();
        uav.fly(base_station);
        uav.pick_up(med_kit);
        uav.fly(hikers);
        uav.drop_off(med_kit);
    }
}
```

Actions have specified preconditions & effects
class Main{
    QuadCopter uav;
    Location base-station;
    People climbers;
    MovableObject med-kit;
    ...

    method run() {
        sequence{
            uav.launch();
            uav.fly(base_station);
            uav.pick_up(med_kit);
            uav.fly(climbers);
            uav.drop_off(med_kit)}

    }

Program A ::= 
    prim_action(args) | 
    [lb, ub] A | 
    Sequence {A1; A2; ...} | 
    Parallel {A1; A2; ...}
QuadCopter Action Model in RMPL

class QuadCopter {
    Roadmap location;
    Boolean flying;
    Boolean loaded;

    primitive method Launch()
        flying == no => flying == yes;

    primitive method land()
        flying == yes => flying == no;

    primitive method pickup(MoveableObject object)
        ((flying == yes) && (loaded == no) && (object.location == location))
        => loaded == yes && (object.on == self);

    primitive method drop_off(MoveableObject object)
        ((flying == yes) && (loaded == yes) && object.on == self)
        => ((loaded == no) &&(object.location == location) && object.on == none);

    #MOTION_PRIMITIVES(location, fly, flying==yes)
}
Volcano eruption!

What could possibly go wrong??

Launch Q

Q fly to Base Station

Q pick up med kit

Q fly to hikers

Q give med kit to climbers
Can set expectations from action preconditions & effects

Launch Q
Q fly to Base Station
Q pick up med kit
Q fly to hikers
Q give med kit to climbers
Can set expectations from action preconditions & effects

Q pick up med kit

time
Can set expectations from action preconditions & effects

Q pick up med kit

- Q near med kit
- Q has empty cargo bay
- Q has med kit!
- Q’s cargo bay full

Q has empty cargo bay

Q near med kit

Q has med kit!
Preconditions tell executive what’s relevant

Launch Q

Q fly to Base Station

Q pick up med kit

Q fly to hikers

Q give supplies to climbers

Q in the air

Q near med kit

Q in the air

Q has med kit

Q has empty cargo bay

Q near climbers

Q drops med kit!
Preconditions tell executive what’s relevant

Action monitoring checks preconditions just before action executed.

(e.g., RosPlan)
How do we **anticipate** failures?

**Causal-link monitoring** checks preconditions as soon as desired effect is first established.

- But for **how long** should each **precondition** hold?
When are preconditions established?

Causal-link monitoring checks preconditions as soon as desired effect is first established.
When are preconditions established?

Causal-link monitoring checks preconditions as soon as desired effect is first established.
When are preconditions established?

Causal-link monitoring checks preconditions as soon as desired effect is first established.
When are preconditions established?

Causal-link monitoring checks preconditions as soon as desired effect is first established.
When are preconditions established?

Causal-link monitoring checks preconditions as soon as desired effect is first established.
When are preconditions established?

Causal-link monitoring checks preconditions as soon as desired effect is first established.
When are preconditions established?

Causal-link monitoring checks preconditions as soon as desired effect is first established.
These are called “causal links”

**Causal link**: identifies how an action’s effect is **consumed** by a later action.
Causal Link

\(<a_{\text{producer}}, p, a_{\text{consumer}}>\)

– Proposition \(p\)
– **Producer**: action with \(p\) as **effect**
– **Consumer**: action with \(p\) as **precondition**

– **Producer** must precede **consumer**
– No action interferes in between
Links tell executive what’s *relevant* to plan success at any time.
Links tell executive what’s *relevant* to plan success at any time.

Launch Q

Q fly to Base Station

Q pick up med kit

Q fly to hikers

Q give supplies to climbers

Q drops med kit!

The UK decides to Brexit!
How do we do perform causal link execution monitoring?

Offline:
• *Extract* causal links from plan & action model

Online:
• Continuously *monitor “active”* causal links against sensor measurements
Active links:

< ... , Q near med kit, Q pick up med kit>
< ... , Q in the air, Q pick up med kit>
< ... , Q has empty cargo bay, Q pick up med kit>

Q in the air

Q has med kit

Q near med kit

Q pick up med kit

Q fly to hikers

Q gives supplies to climbers

Q near climbers

Q has empty cargo bay

time
Active links:

< ... , Q near med kit, Q pick up med kit>
< ... , Q in the air, Q pick up med kit>
< ... , Q has empty cargo bay, Q pick up med kit>

Active links:
Active links:

< ... , Q near med kit, Q pick up med kit>
< ... , Q in the air, Q pick up med kit>
< ... , Q has empty cargo bay, Q pick up med kit>
Active links:
< ... , Q in the air, Q pick up med kit>
Active links:

< ... , Q in the air, Q pick up med kit>  
<Q pick up med kit, Q has med kit, Q give supplies>
Active links:

< ... , Q in the air, Q pick up med kit>

<Q pick up med kit, Q has med kit, Q give supplies>
Active links:
< ... , Q in the air, Q pick up med kit> 
<Q pick up med kit, Q has med kit, Q give supplies>

S:
Q has med kit, 
Q in the air, 
...
Active links:
< ... , Q in the air, Q pick up med kit>
<Q pick up med kit, Q has med kit, Q give supplies>

S:
Not (Q has med kit),
Q in the air,
...

Q in the air
Q near med kit
Q pick up med kit
Q has med kit
Q fly to hikers
Q give supplies to climbers
Q has empty cargo bay
Q near climbers

6/12/2017
T2 self-monitoring, self-diagnosing systems
Key takeaways

• Need to monitor plan sub-goals during execution, to anticipate failures.

• Causal links:
  – Encode what needs to be true, and when
  – Offline (check completeness) and online (monitor)

• See Appendix for details.
Outline

• Programs that Monitor State
  – Sub-goal monitoring
  – Model-based diagnosis and mode estimation

• Programs that Self-Diagnose (opt)
Motivation: Mission Loss
Highlights the Challenge of Robustness

- Mars Observer
- Clementine
- Mars Climate Orbiter
- Mars Polar Lander

courtesy of JPL
Mars Observer Failure

6/12/2017

T2 self-monitoring, self-diagnosing systems
1. Commanded by giving goals
2. Reasons from commonsense models
3. Closes loop on goals
4. Diagnoses, repairs and re-plans.

[Williams & Nayak, AAAI 95; Muscettola et al, AIJ 00]
Executive assigns EngineA = Thrusting, and Livingstone.

Estimate Modes

Oxidizer tank

Fuel tank

Deduces that valves are closed, thrust off, and engine is healthy

Selects valves in the backup engine B that will achieve thrust, and plans needed actions.

Prog: EngineB = Thrusting

Reconfigure Modes

Estimate Modes

Reconfigure Modes

Selects valves To open, and plans actions to open valves

Deduces that a valve failed - stuck closed
Model: Probabilistic Constraint Automata (PCA)

component modes…

described by finite domain constraints on variables…

guarded deterministic and probabilistic transitions

cost / reward & prior distribution

[Williams & Nayak AAAI 95, Williams et al. IEEE Proc. 01]

Engine Model

Camera Model

one per component … operating concurrently
Mode Estimation = Belief State Update

Given prior commands and observations:
1. **What?** Infer distribution on states (modes).
2. **How?** Infer most likely state (*mode*) trajectories.

Approximate by enumerating most likely states or trajectories.
Best-first Tree Search at Each Time Step

- Path search through component mode transitions, while checking constraints.
  - E.g., Enumerate best using conflict-directed A*
Detecting Subtle Failures: Hybrid Mode Estimation

Hybrid:
- Identify discrete failure mode.
- Interpret observations using continuous model.
Model – Hybrid HMM

• **Discrete modes** and transitions between them

\[
\begin{align*}
    x_{t+1} &= f_{\text{nominal}}(x_t, u_t, t) + v_{\text{nominal}}(t) \\
    y_{t+1} &= g_{\text{nominal}}(x_{t+1}, u_t) + \omega_{\text{nominal}}(t)
\end{align*}
\]

\[
\begin{align*}
    x_{t+1} &= f_{\text{failed}}(x_t, u_t, t) + v_{\text{failed}}(t) \\
    y_{t+1} &= g_{\text{failed}}(x_{t+1}, u_t) + \omega_{\text{failed}}(t)
\end{align*}
\]

• **Continuous dynamics** corresponding to each mode
Kalman Filters Track Trajectories

\[ k = 0 \quad k = 1 \quad k = 2 \]

\[ \begin{align*}
ok & \quad 0 \\
ok & \quad 1 \\
failed & \quad 0 \\
failed & \quad 1 \\
ok & \quad 0 \\
ok & \quad 1 \\
failed & \quad 0 \\
failed & \quad 1 \\
\end{align*} \]

\[
\tilde{p}(x_{c,2} \mid x_{d,0:2}^{(i)}, y_{c,1:k})
\]

\[
p(x_{d,0:2}^{(i)} \mid y_{c,1:k}) = 0.01
\]
Outline

• Programs that Monitor State
  – Sub-goal monitoring
  – Model-based diagnosis and mode estimation
    • Static mode estimation
    • Optimal CSPs and conflict-directed A*

• Programs that Self-Diagnose (opt)
Pedagogical Example: Polycell

<table>
<thead>
<tr>
<th>OR</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AND</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XOR</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Model-based Diagnosis

Input: Observations of a system with symptomatic behavior, and a model \( \Phi \) of the system.

Output: Diagnoses that account for the symptoms.
How Should Diagnoses Account for Novel Symptoms?

Consistency-based Diagnosis: Given symptoms, find diagnoses that are consistent with symptoms.

Suspending Constraints: For novel faults, make no presumption about faulty component behavior.

[deKleer & Brown, 83]
[Davis, 84]
[Geneserth, 84]
Multiple Faults: Identify all Combinations of Consistent “Unknown” Modes

And(I):
- I=G:
  \[ \text{Out}(i) = \text{In}1(i) \text{ AND } \text{In}2(i) \]
- I=U:
  True


• Candidate: Assignment of G or U to each component.
Multiple Faults: Identify all Combinations of Consistent “Unknown” Modes

And(I):

- I=G:
  Out(i) = In1(i) AND In2(i)
- I=U:
  True

Diagnosis = \{A1=G, A2=U, A3=G, X1=G, X2=U\}

- Candidate: Assignment of G or U to each component.
- Diagnosis: Candidate consistent with model and observations.
Consistency-based Diagnoses

![Diagram of a consistency-based diagnosis system]

A1 -> X1 -> X2
A2 -> Y -> X1
A3 -> Z -> X2

Symptom

A: 1, B: 1, C: 1, D: 1, E: 1
F: 1, G: 1

6/12/2017

T2 self-monitoring, self-diagnosing systems
Consistency-based Diagnoses

\{A1= U\}
Consistency-based Diagnoses

A1

\{A1 = U\}

\{X1 = U\}
Consistency-based Diagnoses

What, in addition to A2, might be faulty?
Consistency-based Diagnoses

What, in addition to A2, might be faulty?

{A1 = U}

{X1 = U}

{A2 = U, X2 = U}
What, in addition to A2, might be faulty?

- These diagnoses are minimal.
- All extensions are also diagnoses.

⇒ Kernel Diagnoses
Incorporating Failure Modes: Mode Estimation

Inverter(I):
• I=G: Out(i) = not(In(i))
• I=S1: Out(i) = 1
• I=S0: Out(i) = 0
• I=U: True

Good, Faulty and Unknown Modes

Sherlock
[de Kleer & Williams, IJCAI 89]
Example Mode Estimate

Diagnosis: [A=S1, B=G, C=U]

Sherlock
[de Kleer & Williams, IJCAI 89]
Diagnoses: (42 of 64 candidates)

**Fully Explained Failures**
- \([A=G, B=G, C=S0]\)
- \([A=G, B=S1, C=S0]\)
- \([A=S0, B=G, C=G]\)

**Partially Explained**
- \([A=G, B=U, C=S0]\)
- \([A=U, B=S1, C=G]\)
- \([A=S0, B=U, C=G]\)

**Fault Isolated, But Unexplained**
- \([A=G, B=G, C=U]\)
- \([A=G, B=U, C=G]\)
- \([A=U, B=G, C=G]\)
Mode Estimation Problem

Given:
- Mode, State, Observable Variables:
- Model:
  
  And(I):
  
  G(I):
  
  Out(i) = In1(i) AND In2(i)
  
  U(I):

  • All behaviors associated with modes.
  • All components have “unknown mode” U, which enables no constraint.

Return: All diagnoses

\[ D_{\Phi,obs} \equiv \{ X \in D_X \mid \exists Y \in D_Y \text{st } \text{Obs} \land \Phi(X,Y) \} \]
**Problem:** Due to the unknown mode, there tends to be an exponential number of mode estimates.

**Idea(s):**
1. Compute probability of each mode estimate.
2. Enumerate most likely mode estimates $X_i$, based on probability.

Prefix (k) (Sort \{X_i\} by decreasing $P(X_i \mid O)$)
Probabilistic, Mode Estimation

Inputs:
• Mode X, State Y, and Observation (O subset Y) variables, with finite domains.
• Model Φ(X;Y).
• Observations O = o.
• Prior distribution P(X).
• Observation function P(o | X, Φ(X;Y)) = P(o | X).

Outputs:
• Exact: P(X | o) Posterior, given observations.
• Approximate: k most likely mode estimates X_i,
  [Prefix (k) ( Sort {X_i} by decreasing P(X_i | o))].
A Simple Approximation for Probabilistic, Mode Estimation

\[ P(X \mid o) = \frac{P(o \mid X)P(X)}{P(o)} = \alpha P(o \mid X)P(X) \]

\[ P(o) = \frac{1}{\alpha} = \sum_{c_i \in C} P(o \mid c_i)P(c_i) \quad \text{for candidates C consistent with } o \]

1. Assume modes are \textit{a priori} independent:

\[ P(X) = \prod_{X_i \in X} P(X_i) \]

2. Assume consistent \textbf{observations} are equally likely for a given mode assignment:

\[ P(o \mid X) = \begin{cases} 0 & \text{if } \Phi \land o \land X \text{ is inconsistent} \\ 1/n & \text{else } n = \left| \{ o_i \mid \Phi \land o_i \land X \text{ is consistent} \} \right| \end{cases} \]
A Simple Approximation for Probabilistic, Mode Estimation

\[ P(X|o) = \frac{P(o|X)P(X)}{P(o)} = \alpha P(o|X)P(X) \]

\[ P(o) = \frac{1}{\alpha} = \sum_{c_i \in C} P(o|c_i)P(c_i) \quad \text{for candidates C consistent with } o \]

1. Assume modes are *a priori* independent:

\[ P(X) = \prod_{X_i \in X} P(X_i) \]

2. Assume consistent *interpretations* are equally likely, given observations, model and mode assignment.

⇒ Model Counting: Use exhaustive DPLL to count models.
Leading mode estimates before and after output observed.

Top 6 of 64 = 98.6% of P
Outline

• Programs that Monitor State
  – Sub-goal monitoring
  – Model-based diagnosis and mode estimation
    • Static mode estimation
    • Optimal CSPs and conflict-directed A*

• Programs that Self-Diagnose (opt)
Optimal CSP

Mode Estimation: k most likely mode estimates $X_i$,
Prefix (k) (Sort $\{X_i\}$ by decreasing $P(X_i \mid o)$ )

---

Given OCSP= $<X, f, CSP>$

- $X$ are decision variables, with finite domain $D_X$.
- $f: D_X \rightarrow \mathbb{R}$ is a utility function.
- CSP is a set of constraints over variables $<X;Y>$.

Find leading $\operatorname{arg\ max} f(X)$

$$
\begin{align*}
X \in D_X \\
\text{s.t.} \quad \exists Y \in D_Y \cdot C(X,Y)
\end{align*}
$$
Example: Encode Constraints in Propositional State Logic

\[ Y \in \{1, 0\} \]

\[ I \in \{G, S1, S0, U\} \]

**Inverter(I):**
- \( I=G: \) \( \text{Out}(I) = \text{NOT} \\text{In}(I) \)
- \( I=S1: \) \( \text{Out}(I) = 1 \)
- \( I=S0: \) \( \text{Out}(I) = 0 \)

\[ Y=1 \lor Y=0 \]

\[-[Y=1 \land Y=0] \]

\[ I=G \rightarrow [\text{In}(I)=1 \iff \text{Out}(I)=0] \]

\[ I=S1 \rightarrow \text{Out}(I)=1 \]

\[ I=S0 \rightarrow \text{Out}(I)=0 \]

\[ I=G \rightarrow [\text{In}(I)=1 \iff \text{Out}(I)=0] \]

\[ [ \neg(I=G) \lor \neg(\text{In}(I)=1) \lor \text{Out}(I)=0 ] \land \]

\[ [ \neg(I=G) \lor \neg(\text{Out}(I)=0) \lor \text{In}(I)=1 ] \]

Alternative constraints:
- Simple temporal networks
- Linear programs
- Global constraints
Model-based Diagnosis as Conflict-directed Best First Search

When you have eliminated the impossible, whatever remains, however improbable, must be the truth.

- Sherlock Holmes. The Sign of the Four.

1. Generate most likely candidate.
2. Test candidate.
3. If Inconsistent, learn reason for inconsistency (a conflict).
4. Use conflicts to leap over similarly infeasible options to the next best candidate.
Compare Most Likely Candidate to Observations

It is most likely that all components are okay.
The red component modes *conflict* with the *model* and *observations*. 

6/12/2017

T2 self-monitoring, self-diagnosing systems
Leap to the Next Most Likely Candidate that Resolves the Conflict

The next candidate must remove the conflict.
New Candidate Exposes Additional Conflicts

Pressure 1 = nominal
Pressure 2 = nominal

Acceleration = zero

Another conflict, try removing both.
Final Candidate Resolves all Conflicts

Implementation: Conflict-directed A* search.
Solver: OpSat

\[ C_\Phi \equiv \{ X \in D_X | C(X,Y) \text{ satisfiable} \} \]

**Generate (best-first):**
Enumerate every assignment to \( X \), using tree search

**Model:** \( C(X,Y) \)

**Candidate \( C_i \):**
Full assignment to \( X \)

**Consistent?**
(True / False) conflicting assignments

**Test:**
Check if satisfying assignment to \( Y \) exists for: \( C(X,Y) \)

Full assignments to \( X \) s.t. \( C(X,Y) \) satisfiable
Search Tree of Mode Assignments

6/12/2017

T2 self-monitoring, self-diagnosing systems
Constraint-based A*

\[ P_{X_i=G} \gg P_{X_i=U} \]
\[ P_{\text{single}} \gg P_{\text{double}} \]
\[ P_{A2=U} > P_{A1=U} > P_{A3=U} > P_{X1=U} > P_{X2=U} \]
Optimal CSP Candidate Generation using Constraint-based A*  

Initial State:  
- Empty assignment \{\}.

Child Expansion:  
- Select unassigned variable \( y_i \).
- Each child adds an assignment from \( D_i \).

Cutoff:  
- \( C(X,Y) \) is inconsistent.

Goal Test:  
- Full assignment to \( X \).
- \( \exists Y \in D_Y . C(X,Y) \)
Conflicts Indicate How to Remove Symptoms

Symptom:
F is observed 0, but predicted to be 1 if A1, A2 and X1 are okay.

Conflict 1: Learn \{A1=G, A2=G, X1=G\} is inconsistent.
→ One of A1, A2 or X1 must be broken (diagnosis of conflict).

Conflict: Partial assignment to mode variables X, inconsistent with $\Phi \land O=o$. 

6/12/2017 T2 self-monitoring, self-diagnosing systems
Symptom: G is observed 1, but predicted 0.

→ One of A1, A3, X1 or X2 must be broken.
Generate Kernel Diagnoses by Set Covering

\{A_1=G, A_2=G, X_1=G\} \quad \text{Conflict 1.}
\{A_1=G, A_3=G, X_1=G, X_2=G\} \quad \text{Conflict 2.}
\{A_1=U, A_2=U, X_1=U\} \quad \text{Diagnoses of Conflict 1.}
\{A_1=U, A_3=U, X_1=U, X_2=U\} \quad \text{Diagnoses of Conflict 2.}

Kernel Diagnoses =

1. Compute cross product.
2. Remove supersets.
   • Old subset New.
   • New subset Old.

“Smallest” sets of modes that remove all conflicts.
Generate Kernel Diagnoses by Set Covering

Kernel Diagnoses = \{A1=U\}

\{A1=G, A2=G, X1=G\} \hspace{1cm} \text{Conflict 1.}
\{A1=G, A3=G, X1=G, X2=G\} \hspace{1cm} \text{Conflict 2.}
\{A1=U, A2=U, X1=U\} \hspace{1cm} \text{Diagnoses of Conflict 1.}
\{A1=U, A3=U, X1=U, X2=U\} \hspace{1cm} \text{Diagnoses of Conflict 2.}

1. Compute cross product.
2. Remove supersets.
   • Old subset New.
   • New subset Old.

“Smallest” sets of modes that remove all conflicts.
Generate Kernel Diagnoses by Set Covering

\{A_1=G, A_2=G, X_1=G\} \quad \text{Conflict 1.}
\{A_1=G, A_3=G, X_1=G, X_2=G\} \quad \text{Conflict 2.}
\{A_1=U, A_2=U, X_1=U\} \quad \text{Diagnoses of Conflict 1.}
\{A_1=U, A_3=U, X_1=U, X_2=U\} \quad \text{Diagnoses of Conflict 2.}

Kernel Diagnoses = \{A_1=U, A_3=U\}
\{A_1=U\}

1. Compute cross product.
2. Remove supersets.
   - Old subset New.
   - New subset Old.

“Smallest” sets of modes that remove all conflicts.
Kernel Generation Tree

Constituent Kernels

\{X_1=U, A_1=U, A_2=U\}

\{X_1=U, X_2=U, A_1=U, A_3=U\}

cover conflicts

X_1=U
A_1=U
A_2=U

X_2=U
A_3=U

X_1=U
A_1=U
A_2=U \land X_2=U
A_2=U \land A_3=U
Conflict-directed A* Search Tree

Idea 1: Expand kernel tree best-first.
Idea 2: Extend to full assignment.

{X1=U, A1=U, A2=U}

{X1=U, X2=U, A1=U, A3=U}

Idea 1: Expand kernel tree best-first.
Idea 2: Extend to full assignment.
Conflict-directed A* Child Expansion

Conflict
\neg (A2=G \land A1=G \land X1=G)

If Unresolved Conflicts:
- Select unresolved conflict.
- Each child adds a conflict diagnosis (constituent kernel).

If All Conflicts Resolved:
- Select unassigned variable \( y_i \).
- Each child adds an assignment from \( D_i \).
1st Round:

cover conflicts

assign variables

Best kernel

A1=G
A1=U

A2=G
A2=U

A3=G
A3=U

X1=G
X1=U

X2=G
X2=U

Best candidate

Conflict Diagnoses

∅

Test Candidate 1:

1. Test Candidate 1:

Best kernel

P_{X_i=G} >> P_{X_i=U}

P_{single} >> P_{double}

P_{A2=U} > P_{A1=U}

> P_{A3=U} > P_{X1=U}

> P_{X2=U}

6/12/2017

T2 self-monitoring, self-diagnosing systems
1st Round: cover conflicts

[Diagram showing variables A1, A2, A3, X1, X2 with their assignments G or U, and best kernel highlighting the best candidate.

Conflict Diagnoses

- $P_{X_1=G} >> P_{X_1=U}$
- $P_{single} >> P_{double}$
- $P_{A_2=U} > P_{A_1=U}$
- $P_{A_3=U} > P_{X_1=U}$
- $P_{X_2=U}$

Test Candidate 1:

Conflict 1: $\{A_1=G, A_2=G, X_1=G\}$
2nd Round:

cover conflicts

\[ \{X1=U, A1=U, A2=U\} \]

Best kernel

Test Candidate 2:

Best candidate

Conflict Diagnoses

\[ P_{X1=G} >> P_{X1=U} \]

\[ P_{single} >> P_{double} \]

\[ P_{A2=U} > P_{A1=U} \]

\[ > P_{A3=U} > P_{X1=U} \]

\[ > P_{X2=U} \]
2nd Round:

cover conflicts

assign variables

X1=U
A1=U
A2=U

Best kernel

A1=G
A1=U
A3=G
A3=U
X1=G
X1=U
X2=G
X2=U

Test Candidate 2:

Best candidate

Conflict 2: {A1=G, A2=G, X1=G, X2=G}

Conflicting Diagnoses

{X1=U, A1=U, A2=U}

6/12/2017

T2 self-monitoring, self-diagnosing systems
3rd Round:

```
X1=U  A1=U  A2=U
```

**Conflict Diagnoses**

\[ \{X1=U, A1=U, A2=U\}\]

\[ \{X1=U, X2=U, A1=U, A3=U\}\]

**Test Candidate 3:**

Consistent!

- All conflicts discovered.
- All remaining candidates are diagnoses.

6/12/2017 T2 self-monitoring, self-diagnosing systems
Increasing Cost

A*

Feasible

Infeasible
Conflict-directed A*

Increasing Cost

Feasible

Infeasible
Conflict-directed A*

Increasing Cost

Conflict 1

Infeasible

Feasible
Conflict-directed A*
Conflict-directed A*

Increasing Cost

Conflict 1

Conflict 2

Infeasible

Feasible
Conflict-directed A*

Increasing Cost

<table>
<thead>
<tr>
<th></th>
<th>Conflict 1</th>
<th>Infeasible</th>
<th>Conflict 2</th>
<th>Feasible</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6/12/2017  T2 self-monitoring, self-diagnosing systems  104
Conflict-directed A*

Increasing Cost

Conflict 3

Conflict 1

Conflict 2

Infeasible

Feasible
Conflict-directed A*

Increasing Cost

Conflict 1
Infeasible
Conflict 2
Feasible

Conflict 3
Conflict-directed A*

- Each feasible subregion described by a kernel assignment.

⇒ Approach: Use conflicts to search for kernel assignment containing the best cost candidate.
Enumerating Likely Mode Estimates

Mode Estimation and Diagnosis (Satisfiability)

Use **conflicts** to prune large numbers of inconsistent interpretations.

Optimal Search (A*)

Uses a **heuristic** to enumerate solutions best-first.

Conflict-directed A*
[Williams and Ragno 2003]
Use OpSat to . . .

Reconfigure Modes

Execute Programs with Choice

open six valves

Livingstone
[Williams & Nayak, AAAI 96]

Kirk
[Kim, Williams & Abramson, IJCAI 01]
Outline

• Programs that Monitor State
  – Sub-goal monitoring
  – Model-based diagnosis and mode estimation

• Programs that Self-Diagnose (opt)
Personal Transportation System

- The Interactive Trip Advisor
The Problem

• As humans, we often plan to do too much.

• . . . our study plans, working schedules, and travel itineraries are often over-subscribed.

  – ‘I want to see the new movie premiere tonight!’

  – ‘Sorry, you cannot. You need submit your project by mid night, and it will keep you busy until 11pm.’
Approach

View human robot collaboration as a form of collaborative diagnosis.

1. Identify user goals that are over-subscribed, within a temporal plan (QSP).

2. Explain their source of inconsistency, using conflicts.

3. Use constraint suspension and kernels to suggest goals to drop.

- Use continuous relaxation to weaken rather than drop user goals.

- Generalize OpSat to disjunctive linear programs.
Enterprise

- **Dialog Management**: Provides natural language interface
- **Uhura**: Collaboratively diagnoses and resolves goal failures
- **Kirk**: Plans mission and contingencies
- **p-Sulu**: Navigates within risk bounds

**User**

**Natural Language**

**PTS**

**Control Commands**

6/12/2017
I want to go to Boeing in 35 minutes. I want to stop at Tacoma airport for 20 to 30 minutes. I’ve applied for a slot at Tacoma. That’s all.
Kirk searches for an executable travel plan

Enterprise

User

Natural Language

Dialog Management

Provides natural language interface

Uhura

Collaboratively diagnoses and resolves goal failures

Kirk

Plans mission and contingencies.

p-Sulu

Navigates within risk bounds

 PTS

6/12/2017
Uhura Collaborates with User to Diagnose and Repair Plan Failure

User

I have a problem, navigating around the storm will delay us.

PAV

[Yu & Williams, IJCAI 13]
Uhura Collaborates with User to Diagnose and Repair Plan Failure

[Yu & Williams, IJCAI 13]

Preferred Minimal Constraint Relaxation

Can you shorten your stay at Tacoma to 15 minutes.
Uhura Collaborates with User to Diagnose and Repair Plan Failure

No, I cannot do that.

[Yu & Williams, IJCAI 13]
Uhura Collaborates with User to Diagnose and Repair Plan Failure

[Yu & Williams, IJCAI 13]

Next Preferred Minimal Constraint Relaxation

6/12/2017

T2 self-monitoring, self-diagnosing systems
Uhura Collaborates with User to Diagnose and Repair Plan Failure

User

Leaves Home
[10, 15]

Arrives Tacoma
[20, 30]

Leaves Tacoma
[10, 15]

Arrives Boeing

[0, 35]

40

[0, 25]

Another Conflict!

[0, 25]

30

Next Preferred Minimal Constraint Relaxation

[Yu & Williams, IJCAI 13]

T2 self-monitoring, self-diagnosing systems
Uhura Collaborates with User to Diagnose and Repair Plan Failure

Can you delay your take off slot at Tacoma by 5 minutes and your arrival slot at Boeing by 5 minutes? I cannot fly too fast.

[Yu & Williams, IJCAI 13]
Uhura Collaborates with User to Diagnose and Repair Plan Failure

Next Preferred Minimal Constraint Relaxation

[0,35] 40 [0,25] 30


Arrive Boeing

[0,25]

30

Yes, that’s fine.
Best-first, Conflict-directed Relaxation

• **Generate:**
  – Enumerates *minimal relaxations* in *best-first order*.
  – Generalizes *conflict-directed A* to *continuous conflicts*.
  – Framed as a *disjunctive linear program* with only *positive literals*.

• **Test:**
  – Scheduler with conflict extraction.
Goals can be relaxed in many ways
Self-Monitoring, Self-Diagnosing Systems

- **Self-monitoring** is essential to languages and architectures used to build robust robotic systems.
- **Causal link monitoring** enables robots to anticipate and adapt to sub-goal failures.
- **Probabilistic mode estimation** enable robots to diagnose component faults that lead to action failure.
- **Conflict-direct search** solves large decision problems by learning why hypotheses fail.
- **Conflict-directed relaxation** can be used to collaboratively resolve inconsistent plans and programs.
Questions?