Optimal CSPs and Conflict-directed A*

courtesy of JPL

Brian C. Williams
16.412J/6.834J
February 22nd, 2005

R

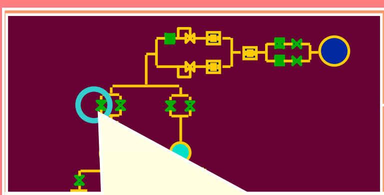
Mode Estimation:

Select a most likely set of component modes that are consistent with the model and observations

Mode Reconfiguration:

Select a least cost set of commandable component modes that entail the current goal, and are consistent

System Model



State estimates

ate goals

Tracks likely plant states

Tracks least cost goal states

arg min $P_t(Y | Obs)$

s.t. $\Psi(X,Y) \wedge O(m')$ is consistent

arg max $R_t(Y)$

s.t. $\Psi(X,Y)$ entails $G(X,Y)$

s.t. $\Psi(X,Y)$ is consistent

ions

Pl

illiams, 2

Outline

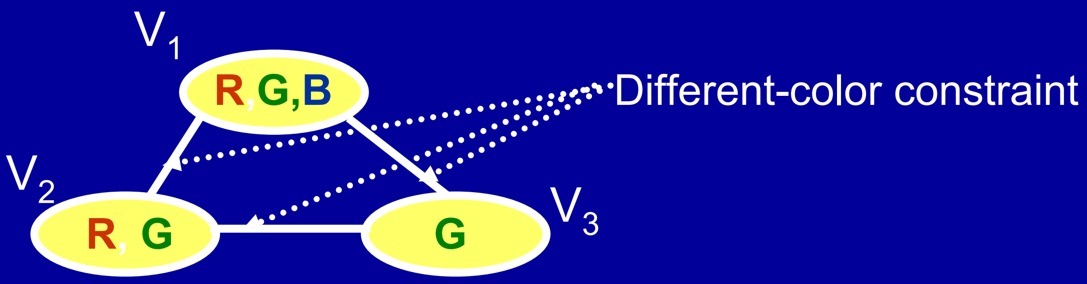
- Optimal CSPs
- Application to Model-based Execution
- Review of A*
- Conflict-directed A*
- Generating the Best Kernel
- Intelligent Tree Expansion
- Extending to Multiple Solutions
- Performance Comparison

Constraint Satisfaction Problem

CSP = <X, D_X,C>

- variables X with domain D_X
- Constraint C(X): $D_X \rightarrow$ {True,False}

Find X in D_X s.t. C(X) is True



Optimal CSP

OCSP= <Y, *g*, CSP>

- Decision variables Y with domain D_Y
- Utility function g(Y): $D_Y \rightarrow \Re$
- CSP is over variables <X,Y>

$$\text{Find Leading } \arg\max_{Y \in D_y} g(Y)$$

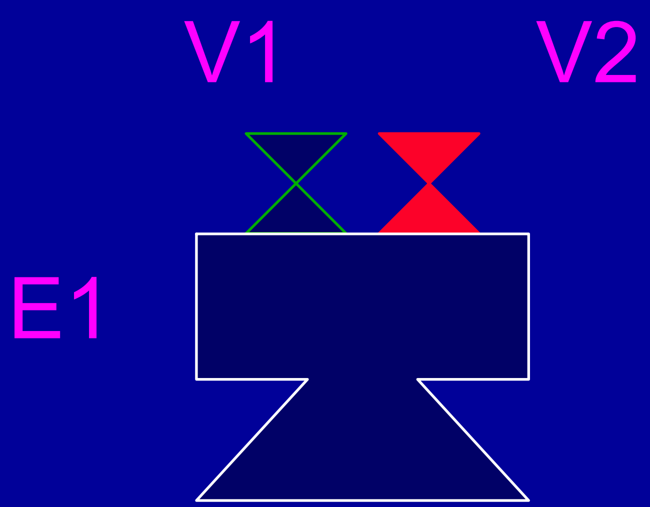
$$\text{s.t. } \exists X \in D_X \text{ s.t. } C(X,Y) \text{ is True}$$

➔ Frequently we encode C in propositional state logic

➔ g() is a multi-attribute utility function that is preferentially independent.

CSP Frequently in Propositional Logic

(mode(E1) = ok implies
(thrust(E1) = on if and only if flow(V1) = on and flow(V2) = on)) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)



Multi Attribute Utility Functions

$g(Y) = G(g_1(y_1), g_2(y_2), . . .)$

where

$G(u_1, u_2 \ldots u_n) = G(u_1, G(u_2 \ldots u_n))$

$G(u_1) = G(u_1, I_G)$

Example: Diagnosis

$g_i(y_i = mode_{ij}) = P(y_i = mode_{ij})$

$G(u_1, u_2) = u_1 \times u_2$

$I_G = 1$

Mutual Preferential Independence

Assignment δ_1 is preferred over δ2
if $g(\delta_1) < g(\delta_2)$

For any set of decision variables $W \subseteq Y$,
our preference between two assignments to W is independent of the assignment to the remaining variables $W - Y$.

Mutual Preferential Independence

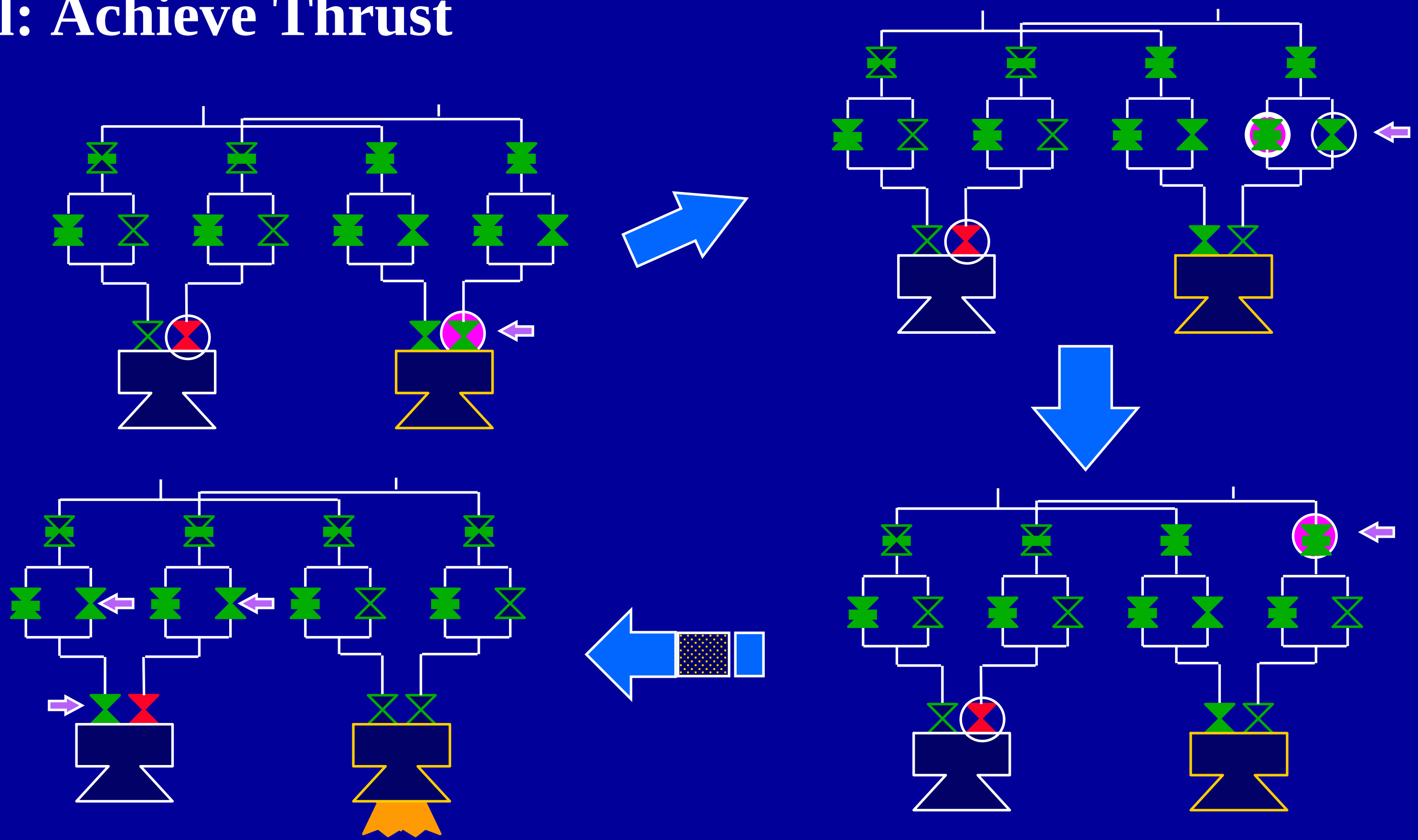
Example: Diagnosis

- If M1 = G is more likely than M1 = U,
- Then,
  {M1 = G, M2 = G, M3 = U, A1 = G, A2 = G}
- Is preferred to
  {M1 = U, M2 = G, M3 = U, A1 = G, A2 = G}

Reconfiguration via Conflict Learning

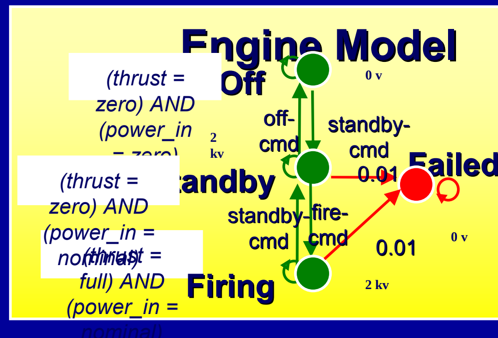
arg max Rt(Y)
s.t. Ψ(X,Y) entails G(X,Y)
s.t. Ψ(X,Y) is consistent

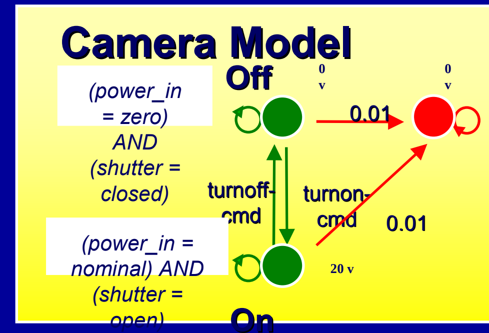
Goal: Achieve Thrust

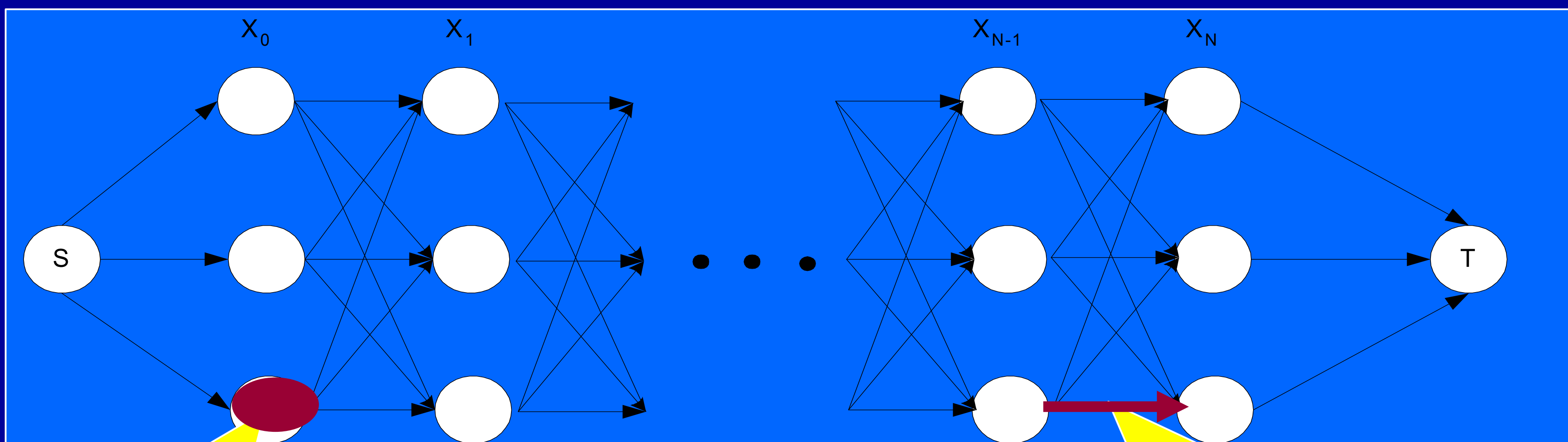


A ***conflict*** is an assignment to a ***subset*** of the control variables that entails the **negation** of the goal.

Approximate PCCA Belief State Update







- Assigns a value to each variable (e.g.,3,000 vars).
- Consistent with all state constraints (e.g., 12,000).

- A set of concurrent transitions, one per automata (e.g., 80).
- Previous & Next states consistent with source & target of transitions

Belief State Propagation

- Propagation Equation
  propagates the system dynamics

$$\mathbf{P}(s_j^{t+1}|o^{<0,t>},\mu^{<0,t>}) = \sum_{s_i^t \in S^t} \left(\mathbf{P}(s_j^{t+1}|s_i^t,\mu^t)\mathbf{P}(s_i^t|o^{<0,t>},\mu^{<0,t-1>})\right)$$

- Update Equation
  updates prior distribution with observations

$$\mathbf{P}(s_j^{t+1}|o^{<0,t+1>},\mu^{<0,t>}) = \frac{\mathbf{P}(s_j^{t+1}|o^{<0,t>},\mu^{<0,t>}) \cdot \mathbf{P}(o^{t+1}|s_j^{t+1})}{\sum_{s_i^{t+1} \in S^{t+1}} \mathbf{P}(s_i^{t+1}|o^{<0,t>},\mu^{<0,t>})\mathbf{P}(o^{t+1}|s_i^{t+1})}$$

Best-First Belief State Enumeration

- Enumerate next state priors in best first order
- Evaluate likelihood of partial states using optimistic estimate of unassigned variables.

$$f(n) = \sum_{s_i^t \in S^t} \left(\mathrm{P}(s_j^{t+1} | s_i^t, \mu^t) \mathrm{P}(s_i^t | o^{<0,t>}, \mu^{<0,t-1>}) \right)$$

$$= \sum_{s_i^t \in S^t} \left(\prod_{x_a \in s_j} \left(\mathrm{P}(x_a^{t+1} = v' | x_a^t = v, \mu^t) \right) \mathrm{P}(s_i^t | o^{<0,t>}, \mu^{<0,t-1>}) \right)$$

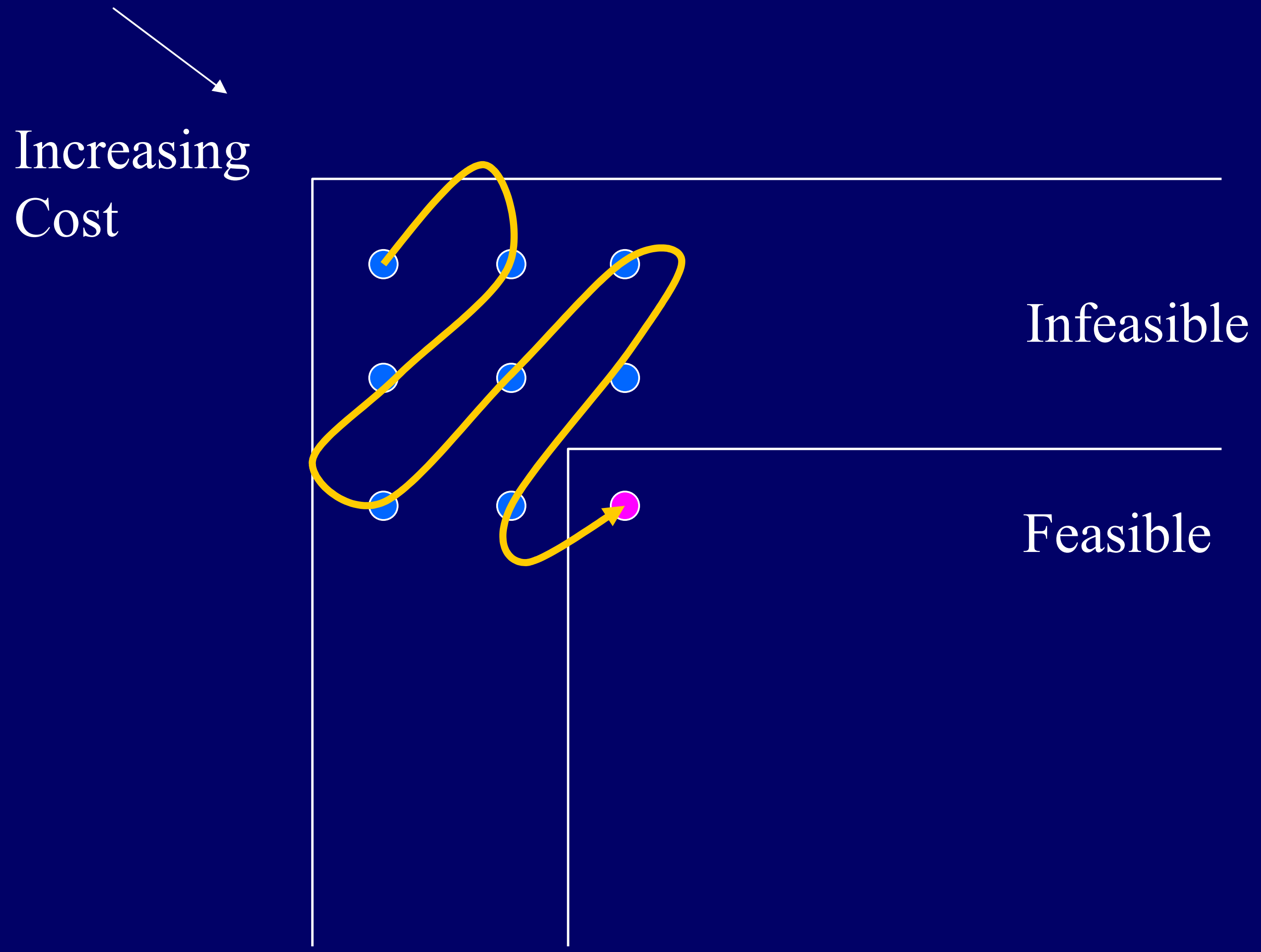
$$= \sum_{s_i^t \in S^t} \left(\prod_{x_g \in n} \left(\mathrm{P}(x_g^{t+1} = v' | x_g^t = v, \mu^t) \right) \prod_{x_h \notin n} \left(\max_{v' \in \mathbb{D}(x_h)} \mathrm{P}(x_h^{t+1} = v' | x_h^t = v, \mu^t) \right) \mathrm{P}(s_i^t | o^{<0,t>}, \mu^{<0,t-1>}) \right)$$

cost so far, g — optimistic estimate of the cost to go, h

Outline

- Optimal CSPs
- Application to Model-based Execution
- Review of A*
- Conflict-directed A*
- Generating the Best Kernel
- Intelligent Tree Expansion
- Extending to Multiple Solutions
- Performance Comparison

A*



A* Search: Search Tree

Problem: **State Space Search Problem**

- Θ — **Initial State**
- Expand(*node*) — **Children of Search Node = next states**
- Goal-Test(node) — **True if search node at a goal-state**

h — ***Admissible Heuristic -Optimistic cost to go***

Search Node: **Node in the search tree**

- **State** — **State the search is at**
- **Parent** — **Parent in search tree**

A* Search: State of Search

Problem: **State Space Search Problem**

- **Θ** — **Initial State**
- Expand(*node*) — **Children of Search Node = adjacent states**
- Goal-Test(node) — **True if search node at a goal-state**
- **Nodes** — **Search Nodes to be expanded**
- **Expanded** — **Search Nodes already expanded**
- **Initialize** — **Search starts at Θ, with no expanded nodes**

g(state) — **Cost to state**

h(state) — ***Admissible Heuristic-Optimistic cost to go***

Search Node: **Node in the search tree**

- **State** — **State the search is at**
- **Parent** — **Parent in search tree**

Nodes[Problem]:

- **Enqueue(*node, f*)** — **Adds node to those to be expanded**
- **Remove-Best(f)** — **Removes best cost queued node according to f**

A* Search

Function A*(*problem, h*)
returns the best solution or failure. Problem pre-initialized.
f(x) ← *g*[*problem*](x) + *h*(x)
loop do

Expand best first

***node* ← Remove-Best(Nodes[*problem*], f)**

***new-nodes* ← Expand(*node, problem*)**
for each *new-node* in *new-nodes*

then Nodes[*problem*] ← Enqueue(Nodes[*problem*], *new-node, f*)

end

A* Search

Function A*(*problem, h*)
returns the best solution or failure. Problem pre-initialized.
f(x) ← *g*[*problem*](x) + *h*(x)
loop do
if Nodes[*problem*] is empty then return failure
node ← Remove-Best(Nodes[*problem*], f)

new-nodes ← Expand(*node, problem*)
for each *new-node* in *new-nodes*

then Nodes[*problem*] ← Enqueue(Nodes[*problem*], *new-node, f*)
if Goal-Test[*problem*] applied to State(*node*) succeeds
then return *node*
end

Terminates when . . .

A* Search

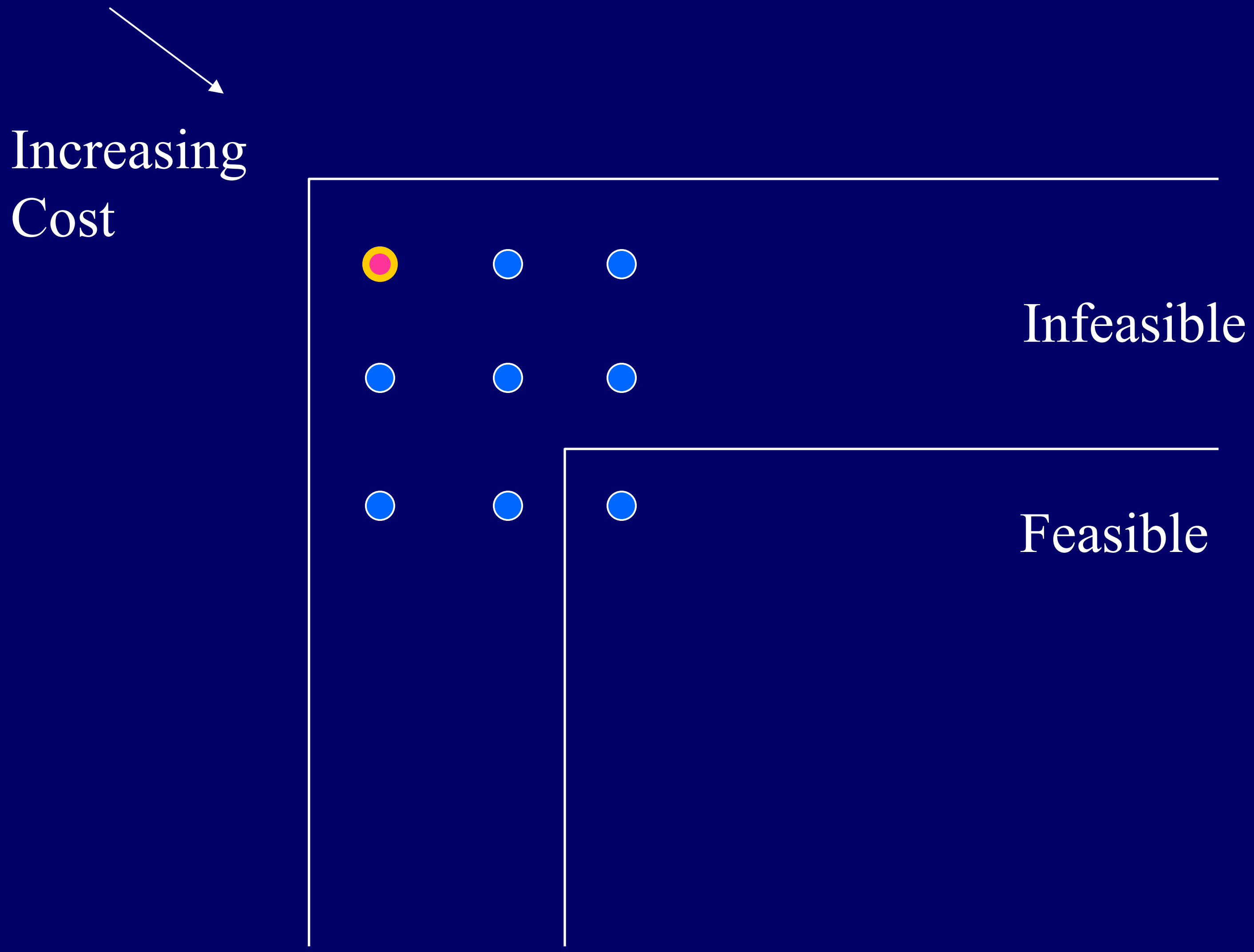
Function A*(*problem, h*)
returns the best solution or failure. Problem pre-initialized.
f(x) ← *g*[*problem*](x) + *h*(x)
loop do
if Nodes[*problem*] is empty **then return** failure
node ← Remove-Best(Nodes[*problem*], f)
state **← State(*node*)**
remove any n from Nodes[*problem*] such that State(n) = *state*
Expanded[*problem*] ← Expanded[*problem*] ∪ {*state*}
new-nodes ← Expand(*node, problem*)
for each *new-node* in *new-nodes*
unless State(*new-node*) is in Expanded[*problem*]
then Nodes[*problem*] ← Enqueue(Nodes[*problem*], *new-node, f*)
if Goal-Test[*problem*] applied to State(*node*) succeeds
then return *node*
end

Dynamic Programming Principle . . .

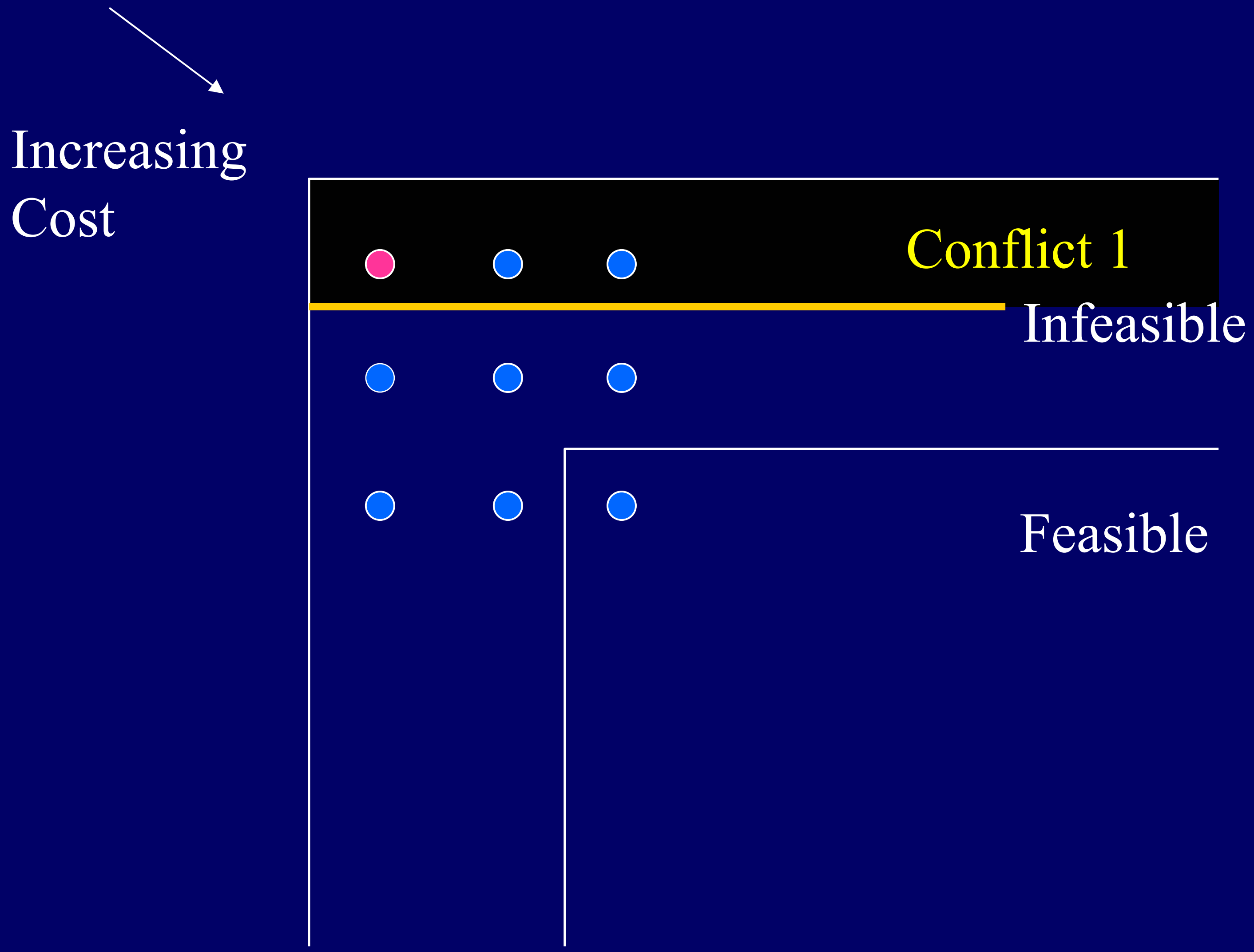
Outline

- Optimal CSPs
- Application to Model-based Execution
- Review of A*
- Conflict-directed A*
- Generating the Best Kernel
- Intelligent Tree Expansion
- Extending to Multiple Solutions
- Performance Comparison

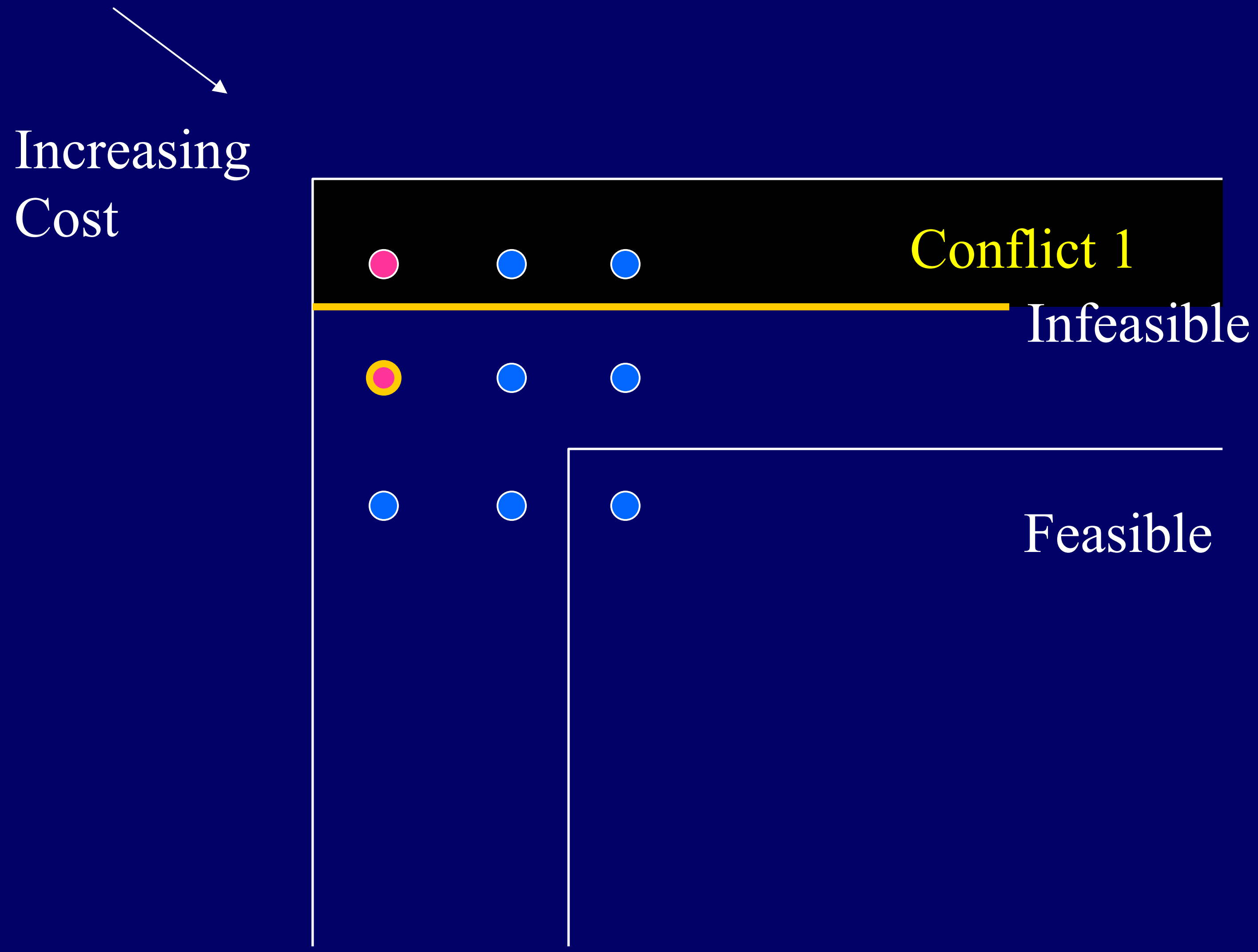
Conflict-directed A*



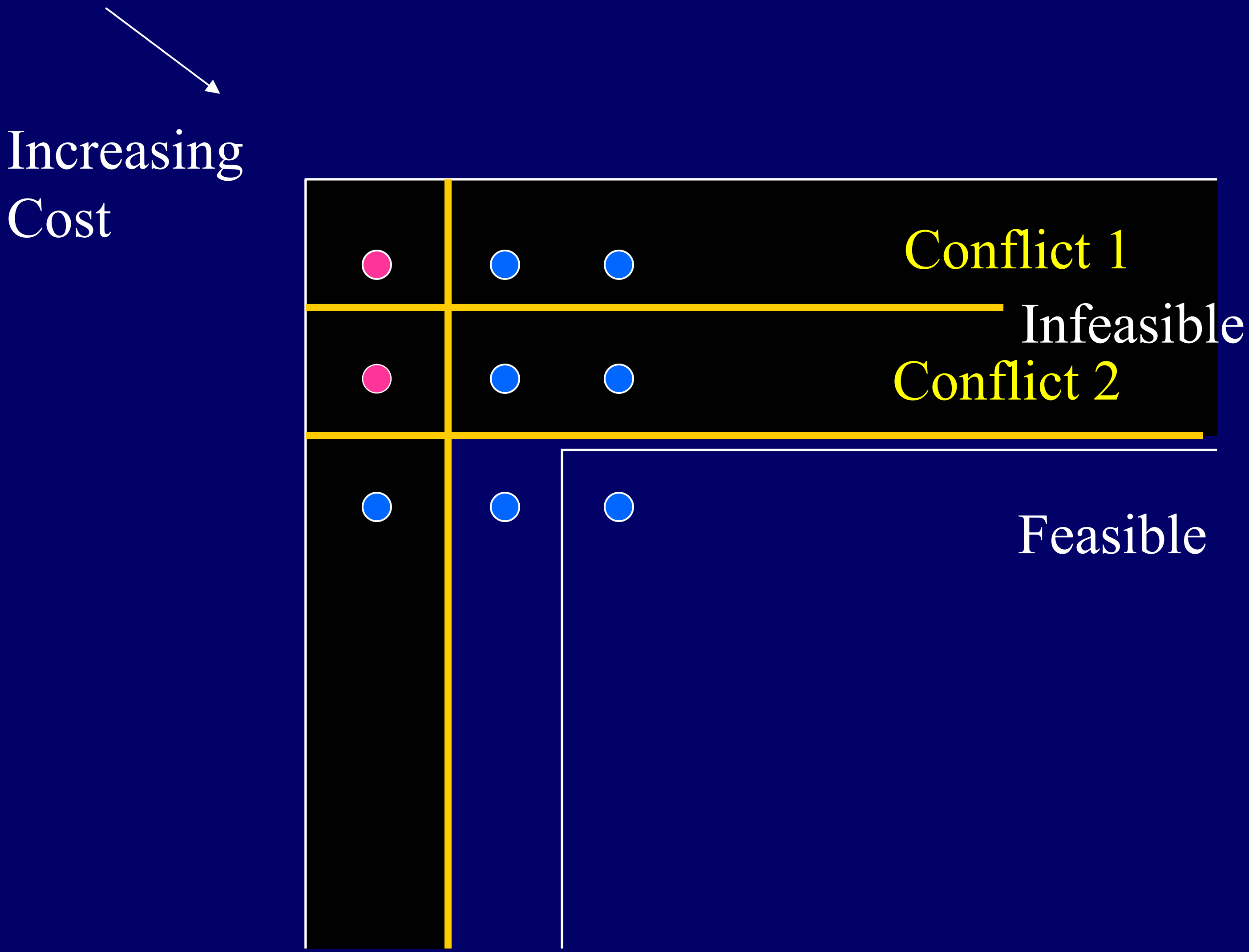
Conflict-directed A*



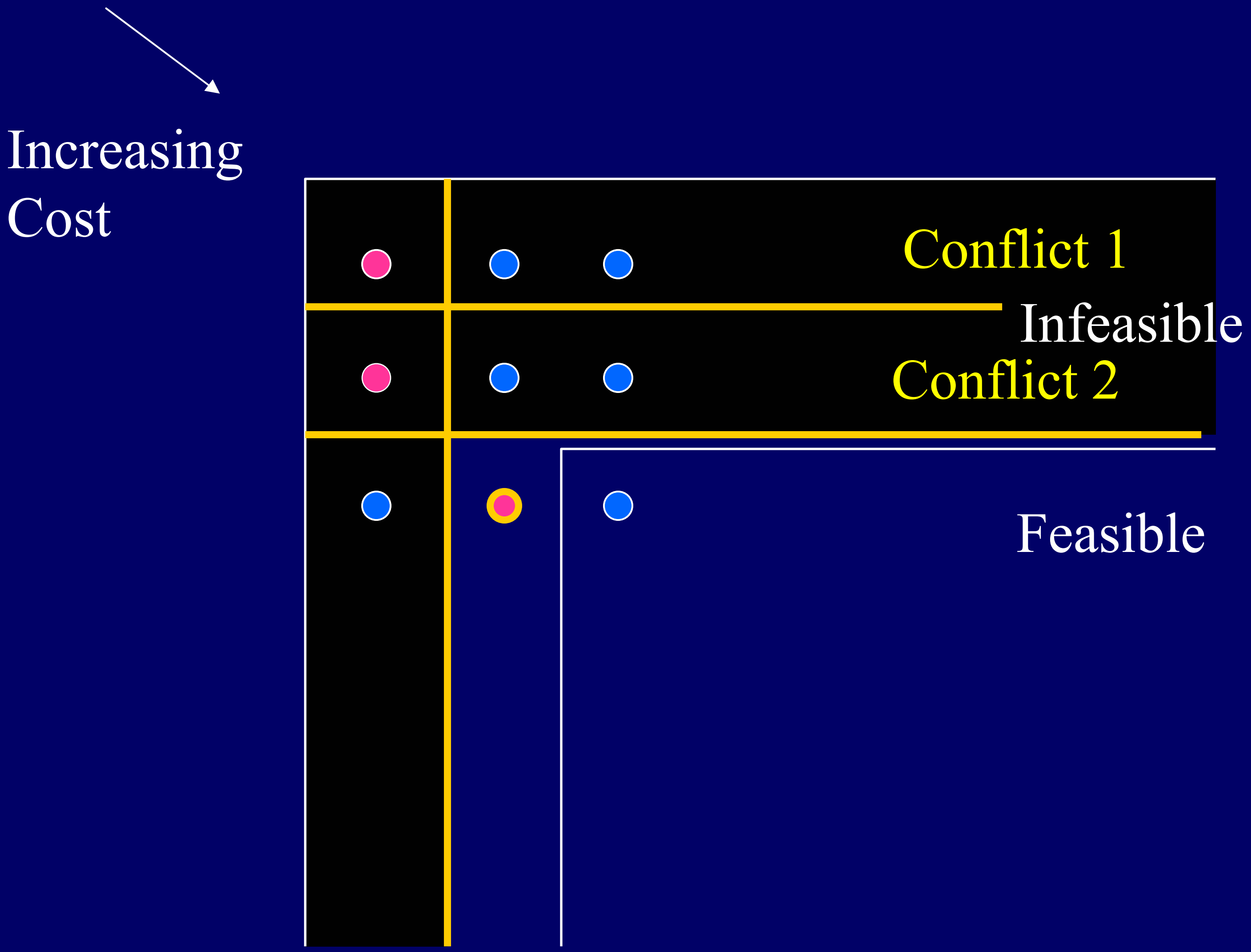
Conflict-directed A*



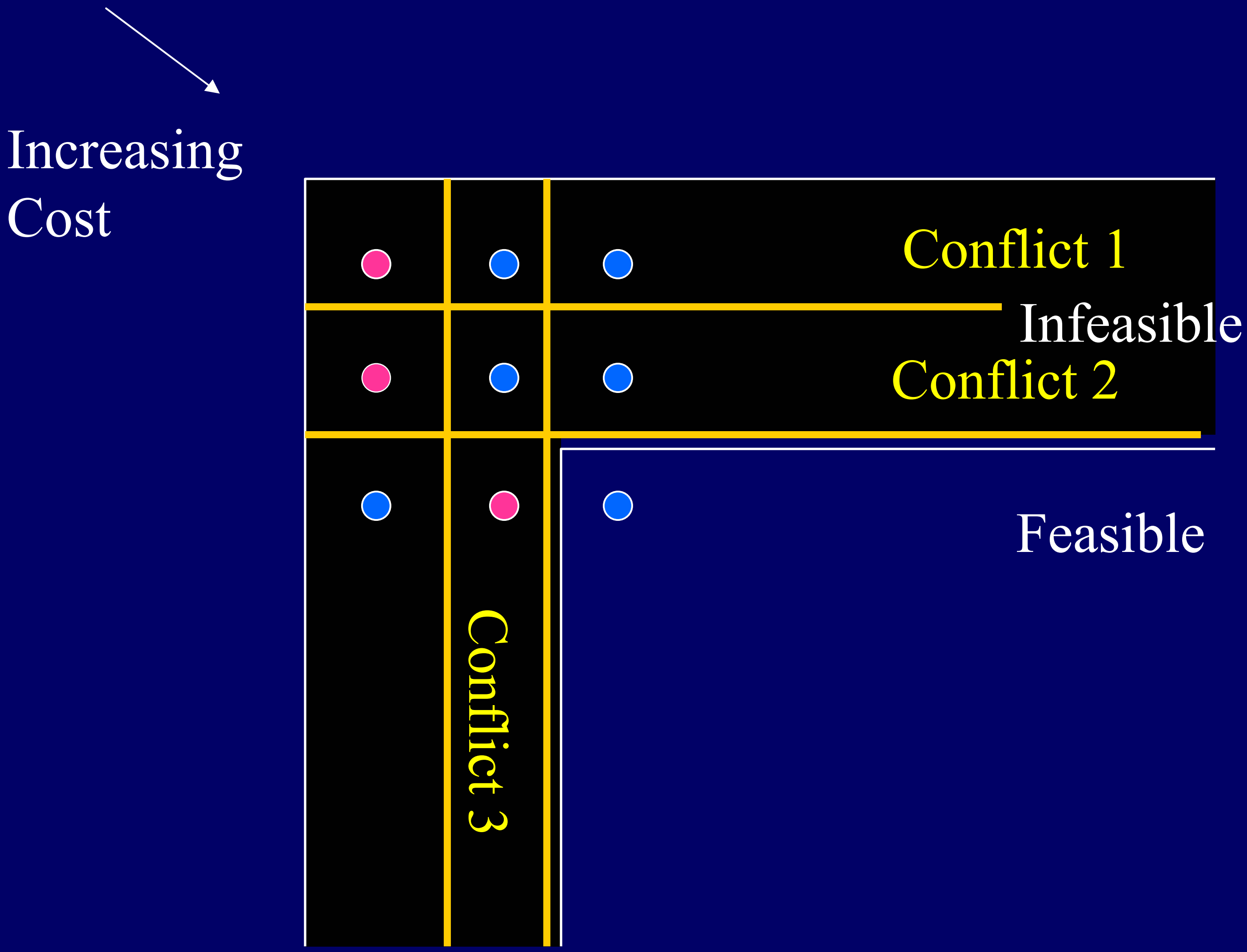
Conflict-directed A*



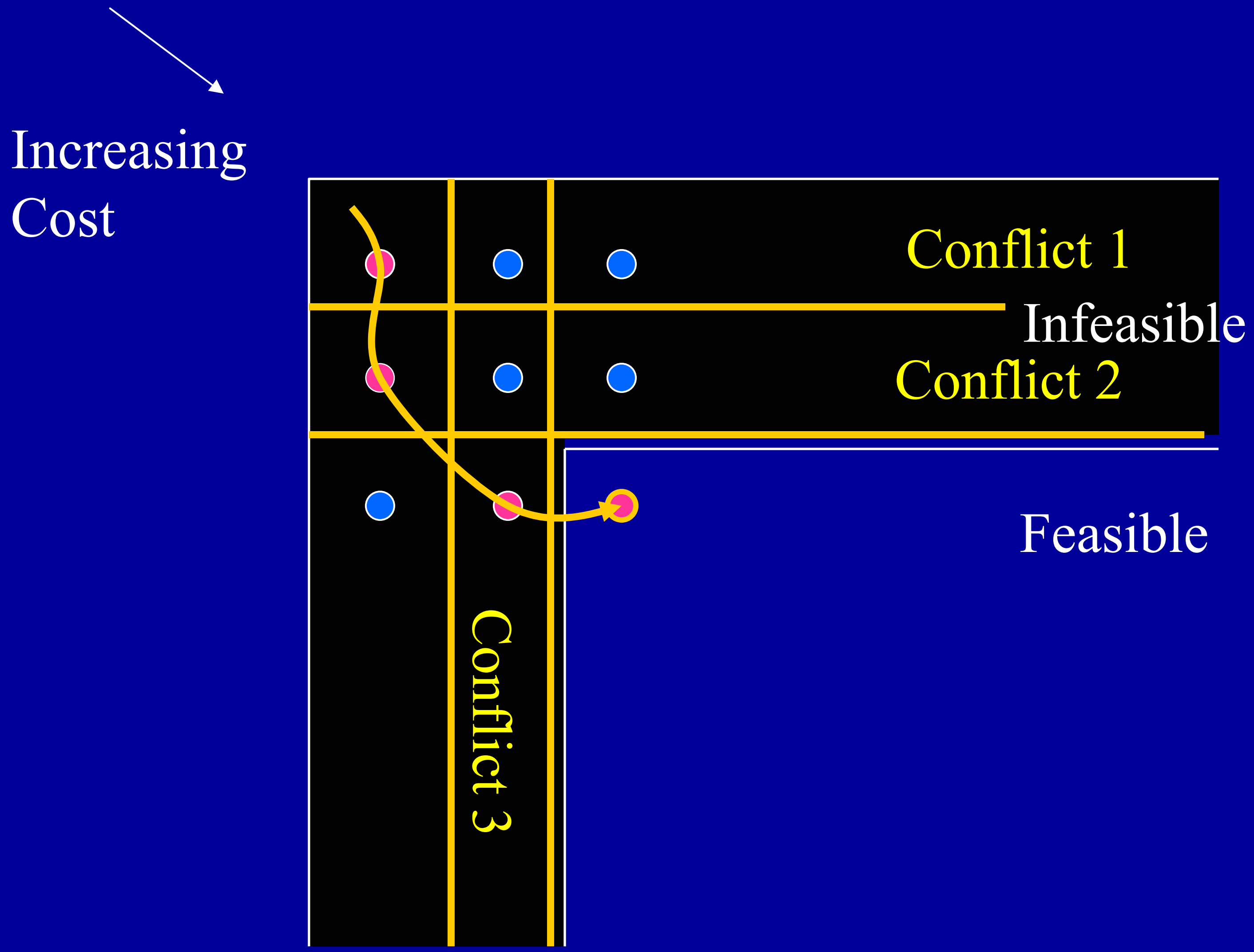
Conflict-directed A*



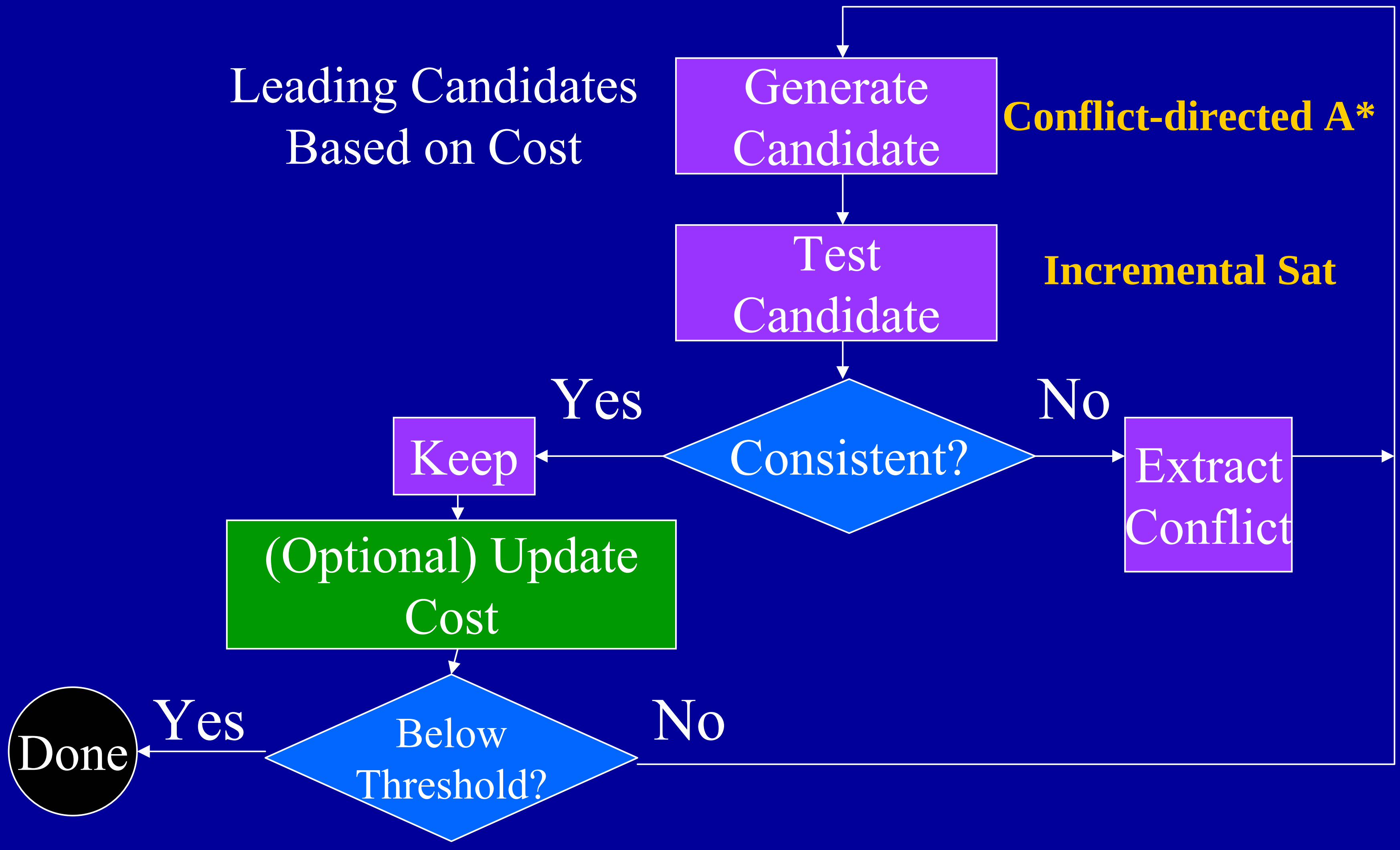
Conflict-directed A*



Conflict-directed A*



Solving Optimal CSPs Through Generate and Test



Conflict-directed A*

Function Conflict-directed-A*(*OCSP*)
returns the leading minimal cost solutions.
Conflicts[*OCSP*] ← {}
OCSP ← Initialize-Best-Kernels(*OCSP*)
Solutions[*OCSP*] ← {}
loop do
decision-state ← **Next-Best-State-Resolving-Conflicts(OCSP)**

new-conflicts ← **Extract-Conflicts(CSP[OCSP],** ***decision-state*****)**
Conflicts[OCSP]
← Eliminate-Redundant-Conflicts(Conflicts[OCSP] ∪ new-conflicts)
end

Conflict-guided Expansion

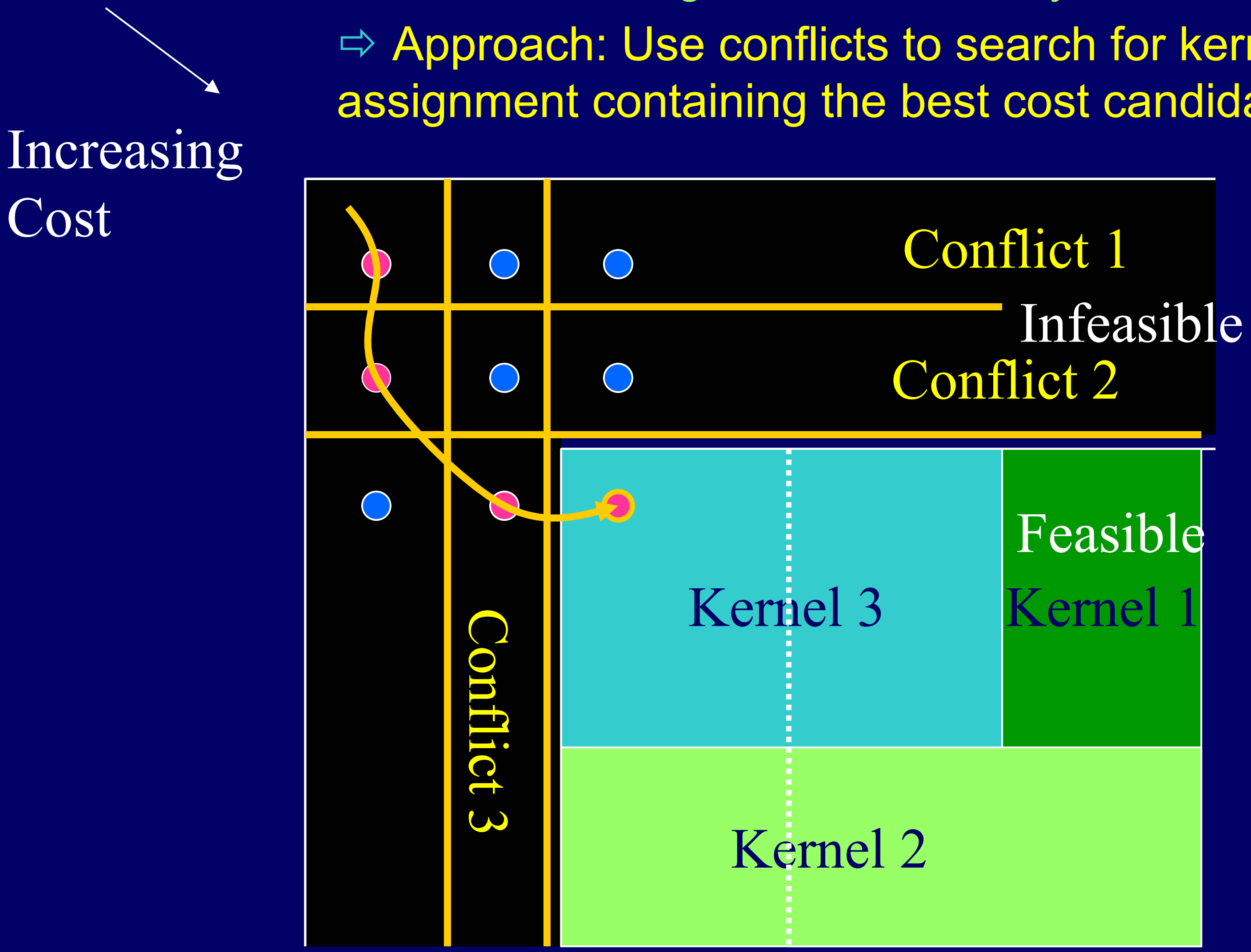
Conflict-directed A*

Function Conflict-directed-A*(*OCSP*)
returns the leading minimal cost solutions.
Conflicts[*OCSP*] ← {}
OCSP ← Initialize-Best-Kernels(*OCSP*)
Solutions[*OCSP*] ← {}
loop do
decision-state ← Next-Best-State-Resolving-Conflicts(OCSP)
if no decision-state returned or
Terminate?(*OCSP*)
then return Solutions[*OCSP*]
if Consistent?(CSP[*OCSP*], *decision-state)*
then add *decision-state* to Solutions[OCSP]
new-conflicts ← Extract-Conflicts(CSP[OCSP], *decision-state*)
Conflicts[OCSP]
← Eliminate-Redundant-Conflicts(Conflicts[OCSP] ∪ new-conflicts)
end

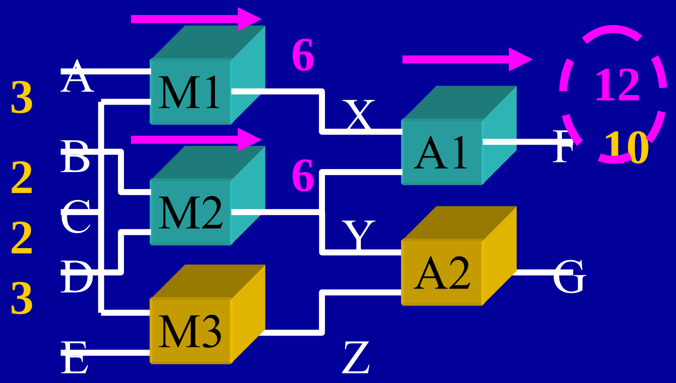
Conflict-directed A*

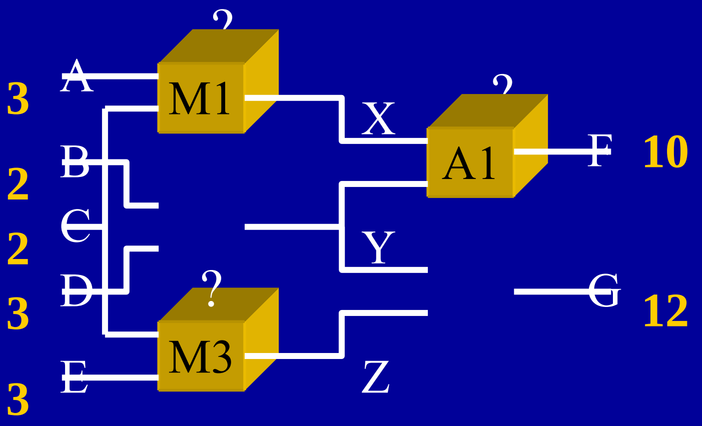
- Feasible subregions described by kernel assignments.

⇨ Approach: Use conflicts to search for kernel assignment containing the best cost candidate.



Mapping Conflicts to Kernels





Conflict C_i: A set of decision variable assignments that are inconsistent with the constraints.

Constituent Kernel: An assignment A that resolves a conflict C_i.

A entails □ $\neg C_i$

Kernel: A minimal set of decision variable assignments that resolves all known conflicts C.

A entails □ $\neg C_i$ for all C_i in C

Mapping conflict to constituent kernels

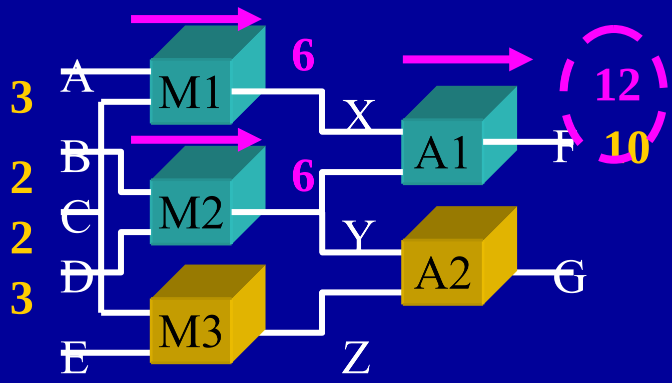
Conflict: {M1=G, M2=G, A1=G}

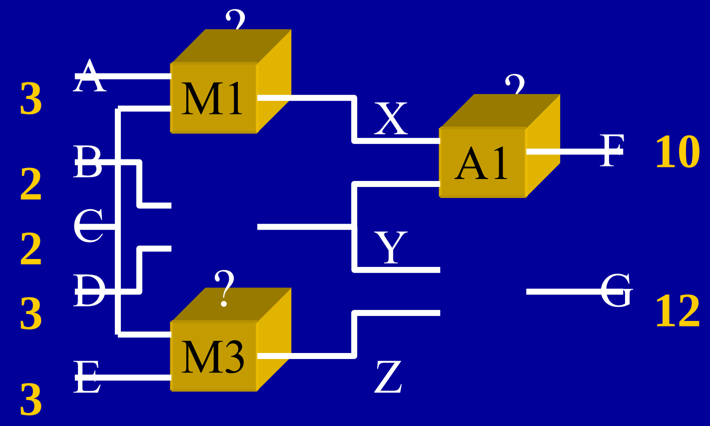
$\neg$(M1=G $\wedge$ M2=G $\wedge$ A1=G)

M1=U $\vee$ M2=U $\vee$ M3=U

Constituent Kernels: {M1=U, M2=U, A1=U}

Composing Constituents Kernels of Every Conflict





Constituent Kernel: An assignment A that resolves a conflict C_i.

A entails □ $\neg C_i$

Kernel: A minimal set of decision variable assignments that resolves all known conflicts C.

A entails □ $\neg C_i$ for all C_i in C

⇨ Constituent kernels map to kernels by minimal set covering

Extracting a kernel's best state

- Select best utility value for unassigned variables (Why?)

{M2=U}

↓

M1=? ∧ M2=U ∧ M3=? ∧ A1=? ∧ A2=?

↓

M1=G ∧ M2=U ∧ M3=G ∧ A1=G ∧ A2=G

Next Best State Resolving Conflicts

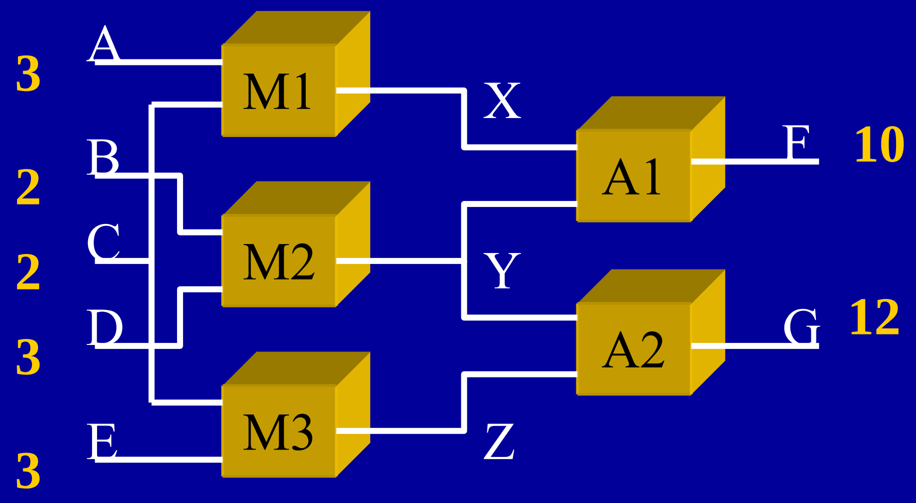
```
function Next-Best-State-Resolving-Conflicts(OCSP)
  best-kernel ← Next-Best-Kernel(OCSP)
  if best-kernel = failure
    then return failure
    else return kernel-Best-State[problem](best-kernel)
end

function Kernel-Best-State(kernel)
  unassigned ← all variables not assigned in kernel
  return kernel ∪ {Best-Assignment(v) | v ∈ unassigned}
End

function Terminate?(OCSP)
  return True iff Solutions[OCSP] is non-empty
```

Algorithm for only finding the first solution, multiple later.

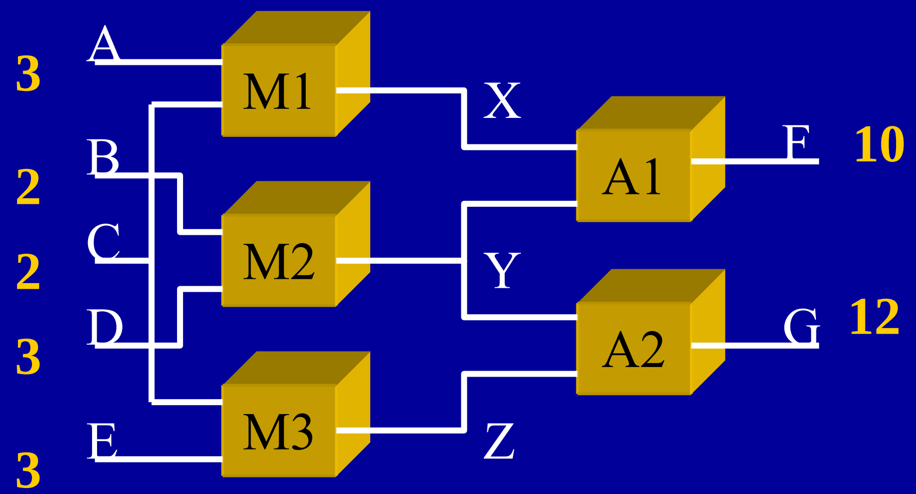
Example: Diagnosis



Assume Independent Failures:

- $P_{G(mi)} >> P_{U(mi)}$
- $P_{single} >> P_{double}$
- $P_{U(M2)} > P_{U(M1)} > P_{U(M3)} > P_{U(A1)} > P_{U(A2)}$

First Iteration



- Conflicts / Constituent Kernels
  - none
- Best Kernel:
  - {}
- Best Candidate:
  - ?

Extracting the kernel's best state

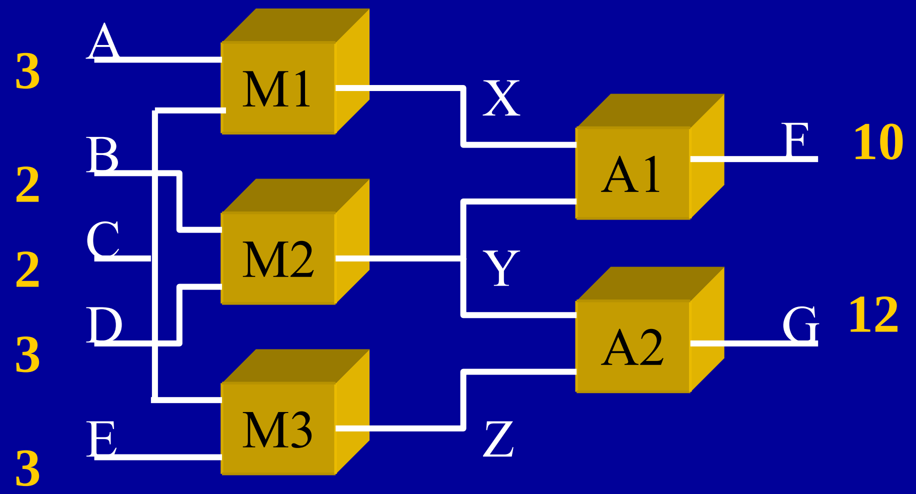
- Select best value for unassigned variables

{ }

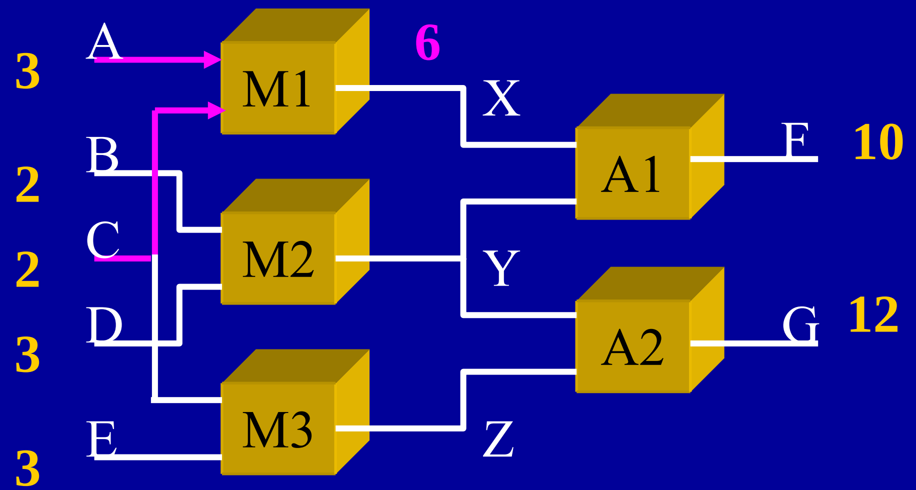
$M1=? \wedge M2=? \wedge M3=? \wedge A1=? \wedge A2=?$

$M1=G \wedge M2=G \wedge M3=G \wedge A1=G \wedge A2=G$

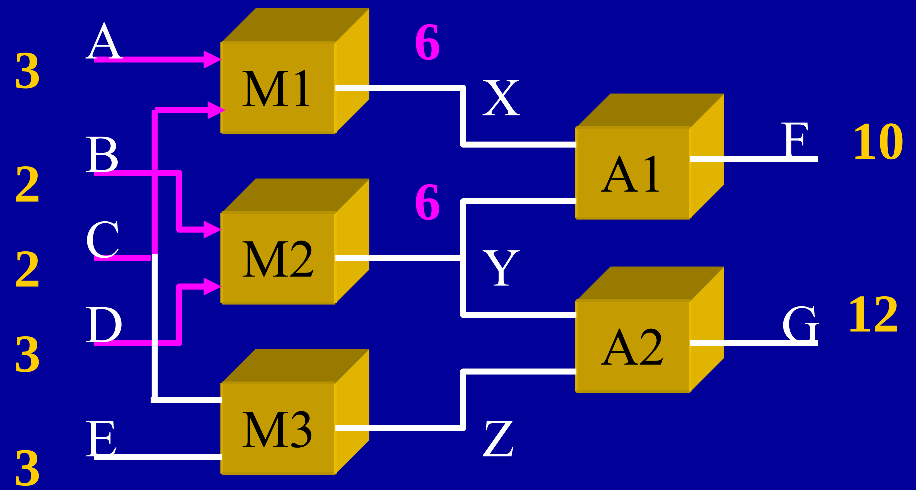
- Test: M1=G ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G



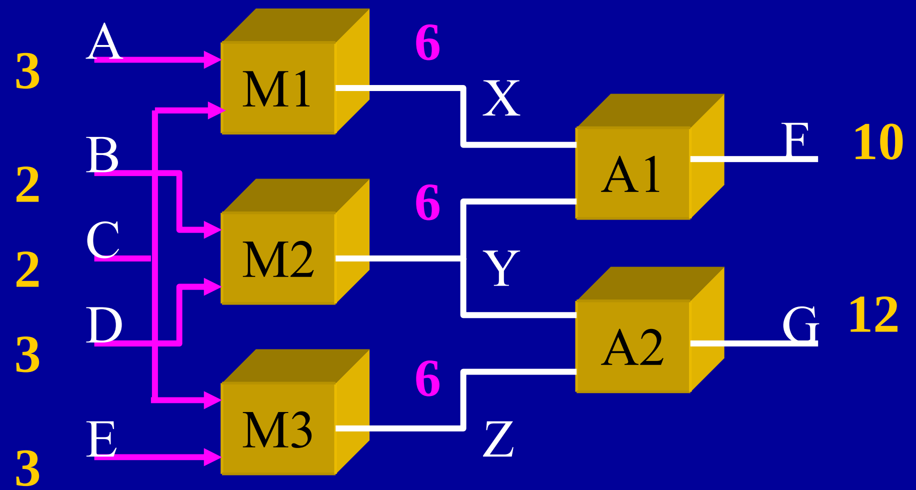
- Test: M1=G ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G



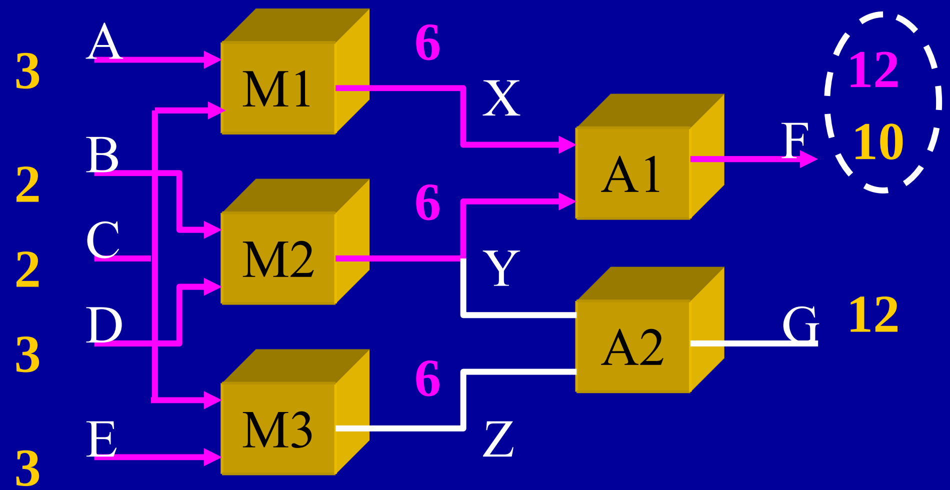
- Test: M1=G ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G



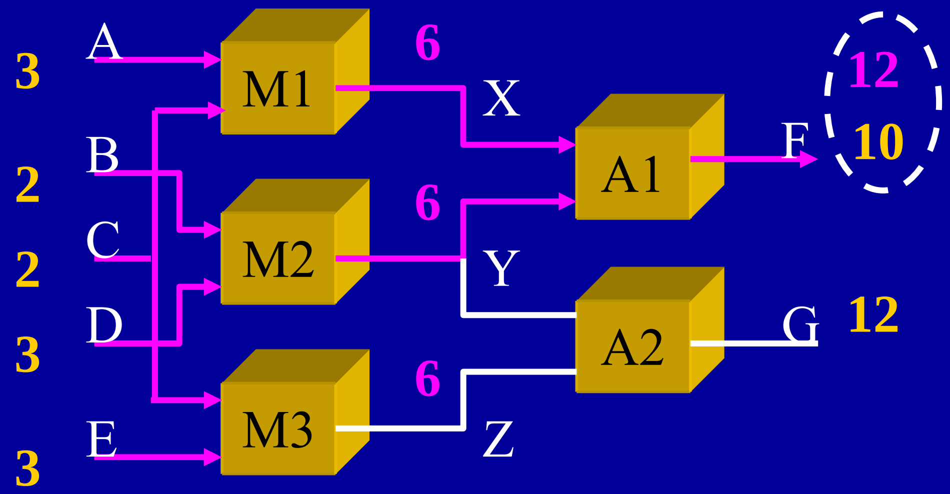
- Test: M1=G ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G



- Test: M1=G $\wedge$ M2=G $\wedge$ M3=G $\wedge$ A1=G $\wedge$ A2=G

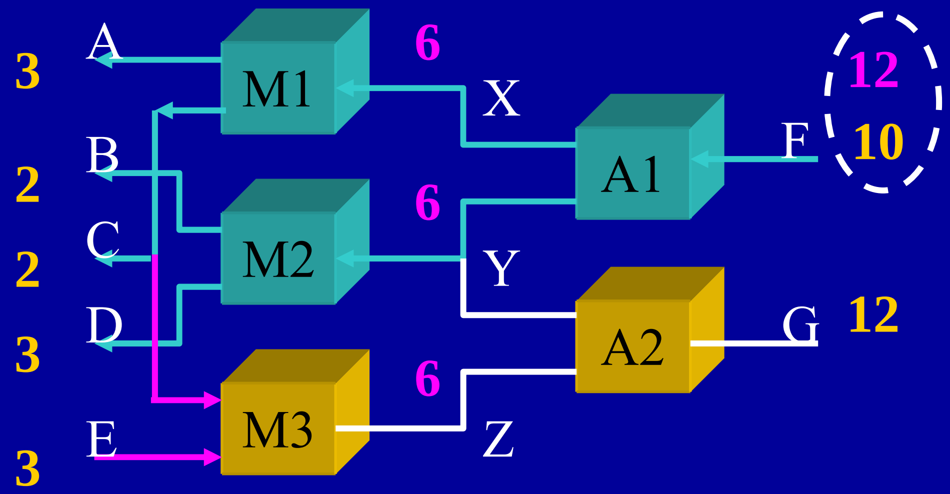


- Test: M1=G ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G



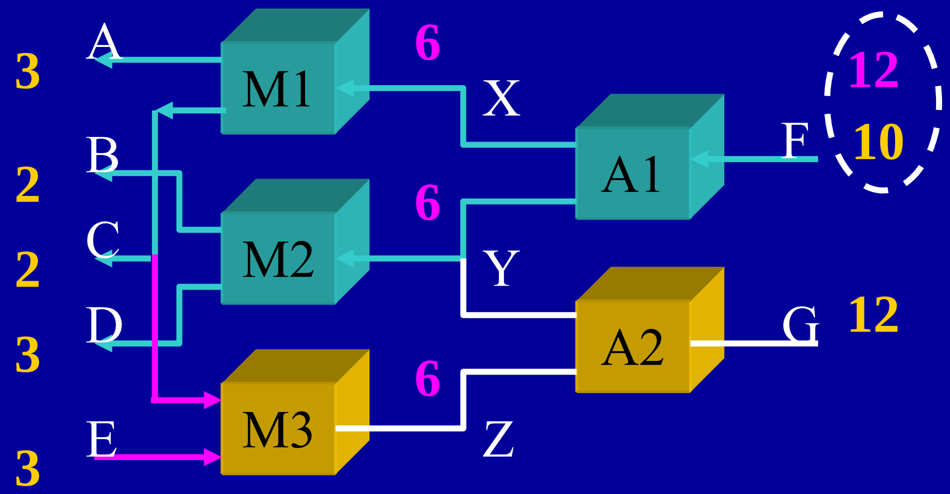
- Extract Conflict and Constituent Kernels:

- Test: M1=G ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G



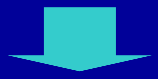
- Extract Conflict and Constituent Kernels:

- Test: M1=G ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G



- Extract Conflict and Constituent Kernels:

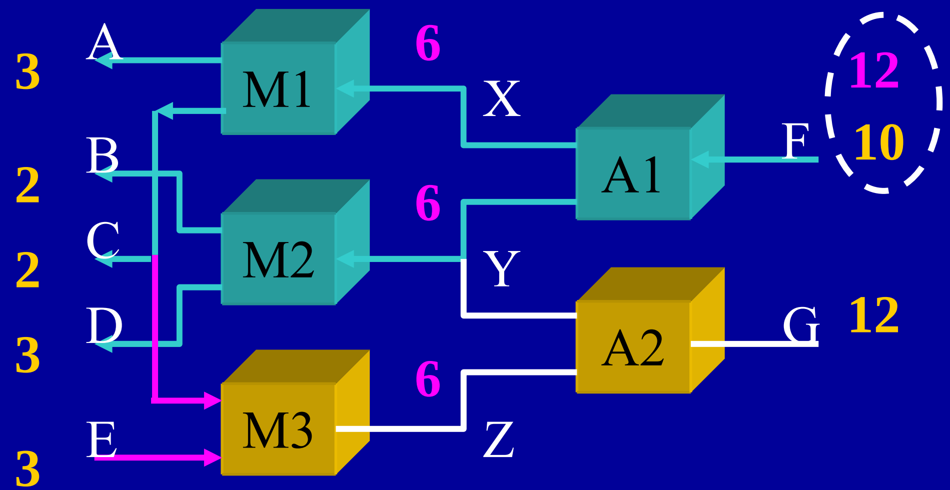
¬ [M1=G ∧ M2=G ∧ A1=G]



M1=U ∨ M2=U ∨ A1=U

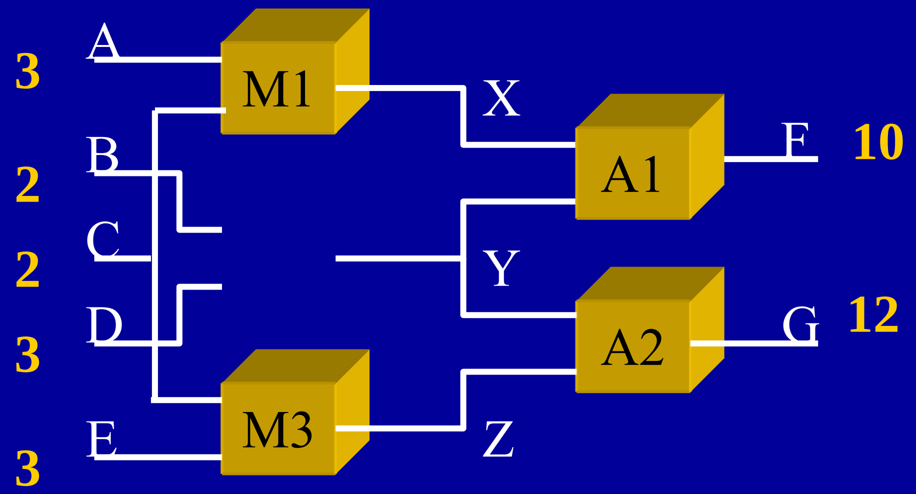
Second Iteration

- $P_{G(mi)} >> P_{U(mi)}$
- $P_{single} >> P_{double}$
- $P_{U(M2)} > P_{U(M1)} > P_{U(M3)} > P_{U(A1)} > P_{U(A2)}$

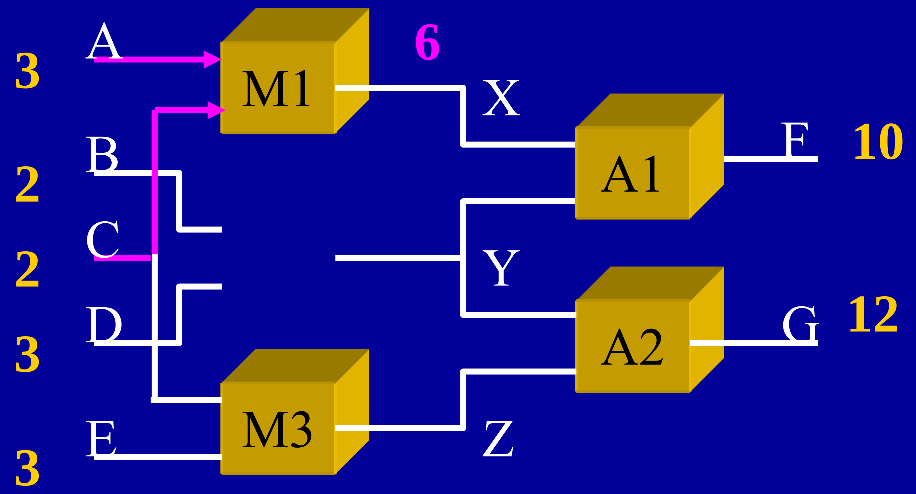


- Conflicts ⇨ Constituent Kernels
  - M1=U ∨ M2=U ∨ A1=U
- Best Kernel:
  - M2=U (why?)
- Best Candidate:
  - M1=G ∧ M2=U ∧ M3=G ∧ A1=G ∧ A2=G

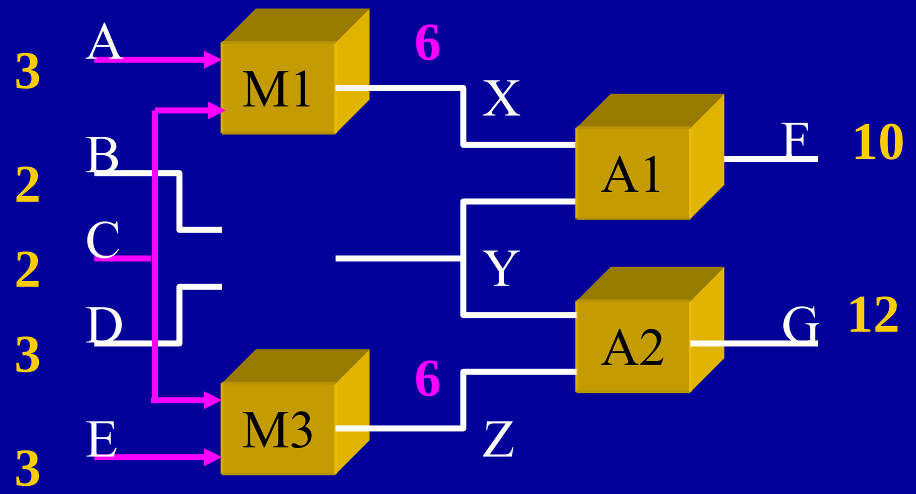
- Test: M1=G ∧ M2=U ∧ M3=G ∧ A1=G ∧ A2=G



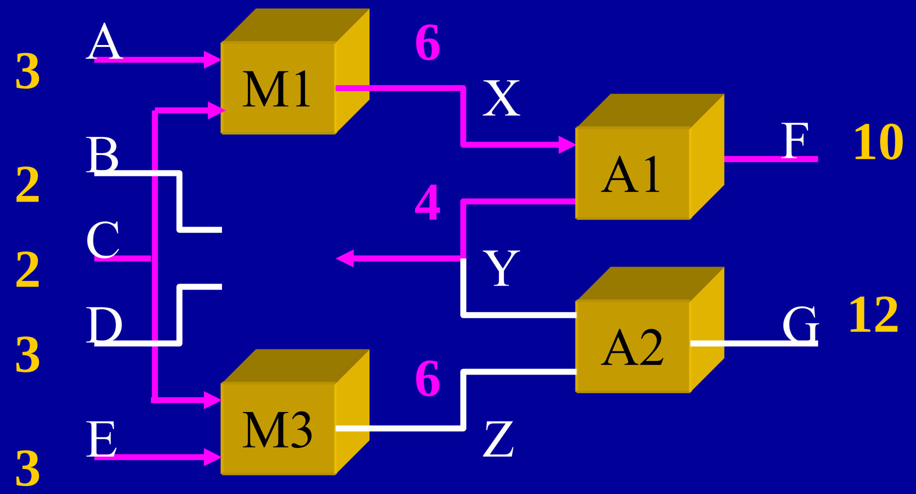
- Test: M1=G ∧ M2=U ∧ M3=G ∧ A1=G ∧ A2=G



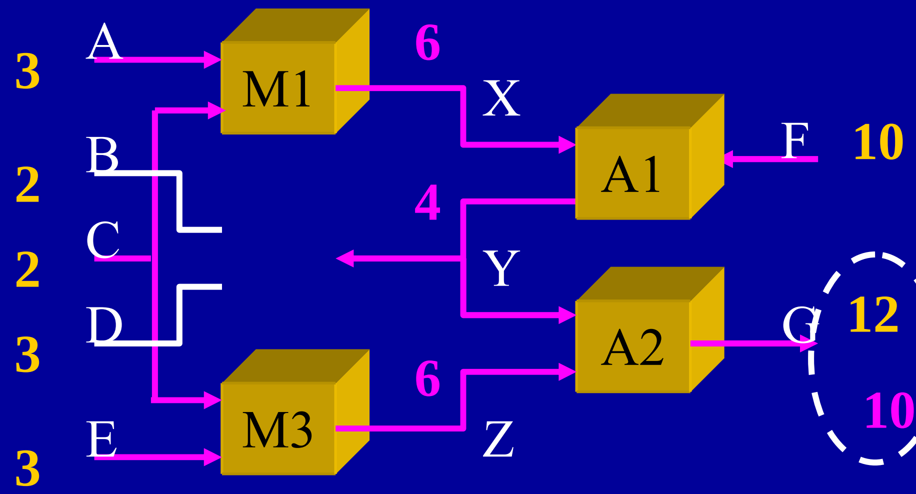
- Test: M1=G ∧ M2=U ∧ M3=G ∧ A1=G ∧ A2=G



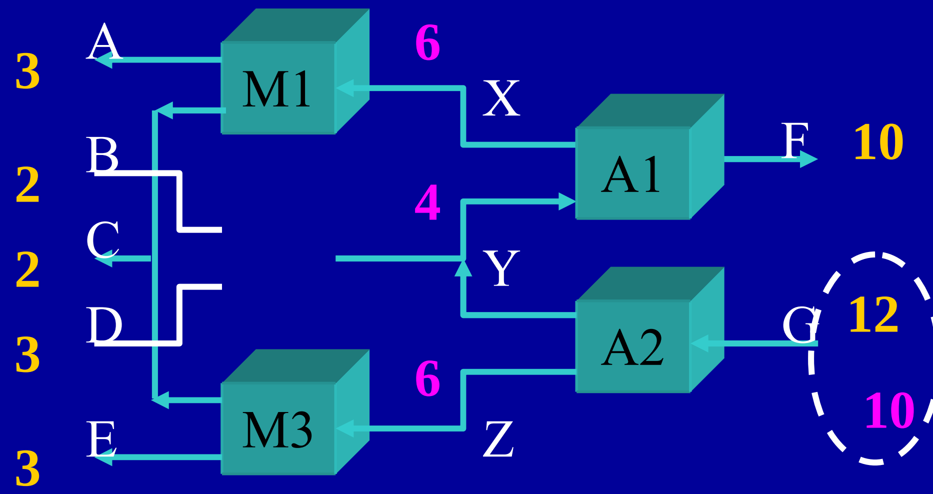
- Test: M1=G ∧ M2=U ∧ M3=G ∧ A1=G ∧ A2=G



- Test: M1=G ∧ M2=U ∧ M3=G ∧ A1=G ∧ A2=G



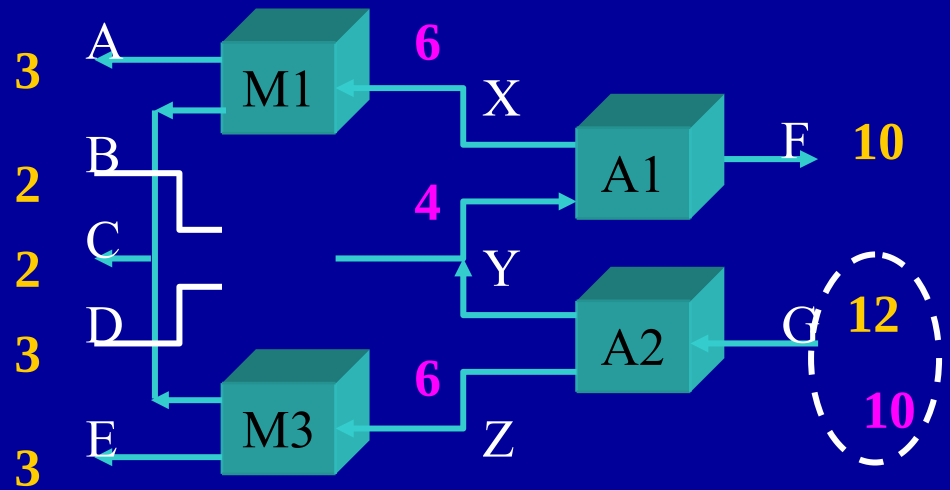
- Test: M1=G ∧ M2=U ∧ M3=G ∧ A1=G ∧ A2=G



- Extract Conflict:

  ¬ [M1=G ∧ M3=G ∧ A1=G ∧ A2=G]

- Test: M1=G ∧ M2=U ∧ M3=G ∧ A1=G ∧ A2=G



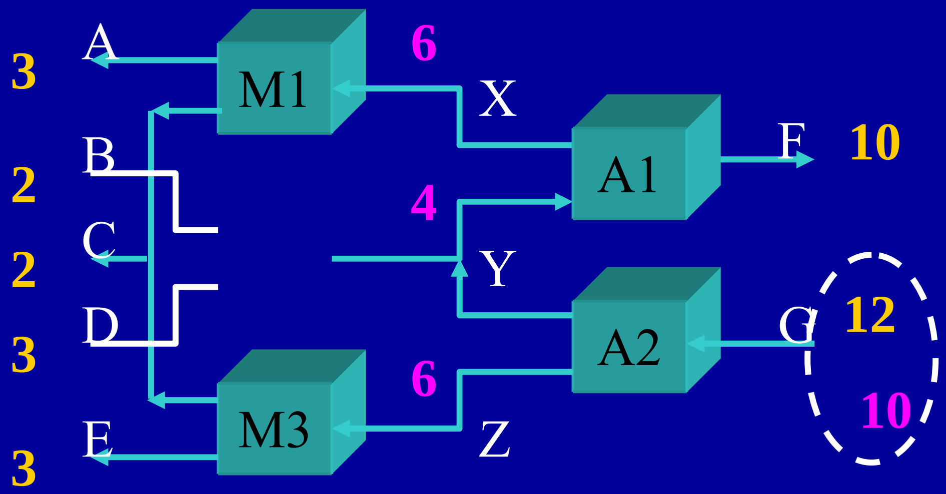
- Extract Conflict:

$\neg$ [M1=G ∧ M3=G ∧ A1=G ∧ A2=G]

M1=U ∨ M3=U ∨ A1=U ∨ A2=U

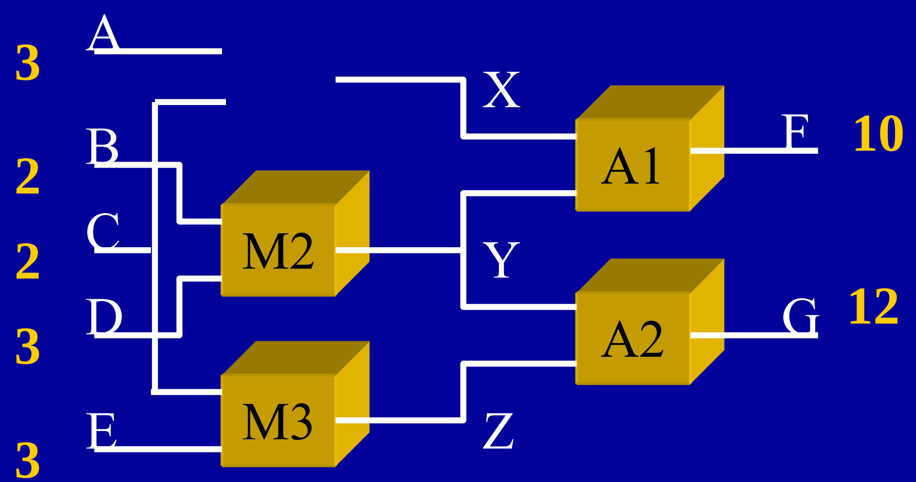
Second Iteration

- $P_{G(mi)} >> P_{U(mi)}$
- $P_{single} >> P_{double}$
- $P_{U(M2)} > P_{U(M1)} > P_{U(M3)} > P_{U(A1)} > P_{U(A2)}$

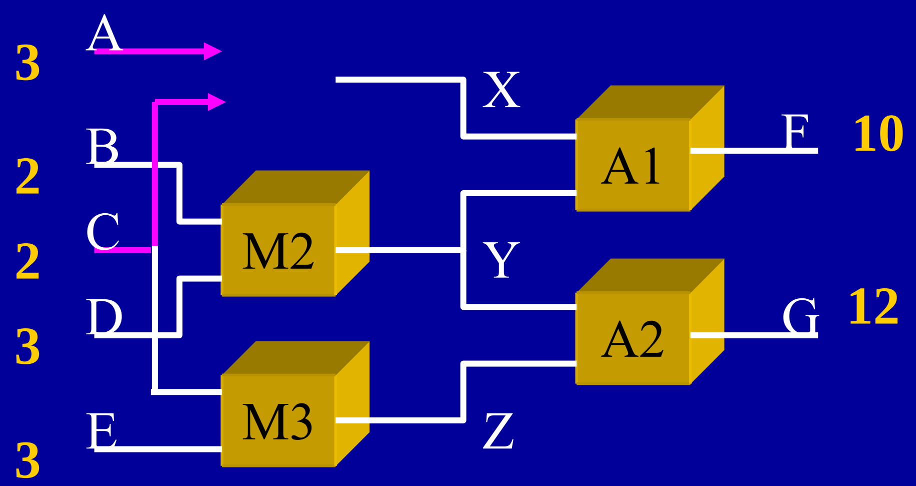


- Conflicts ⇨ Constituent Kernels
  - M1=U ∨ M2=U ∨ A1=U
  - M1=U ∨ M3=U ∨ A1=U ∨ A2=U
- Best Kernel:
  - M1=U
- Best Candidate:
  - M1=U ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G

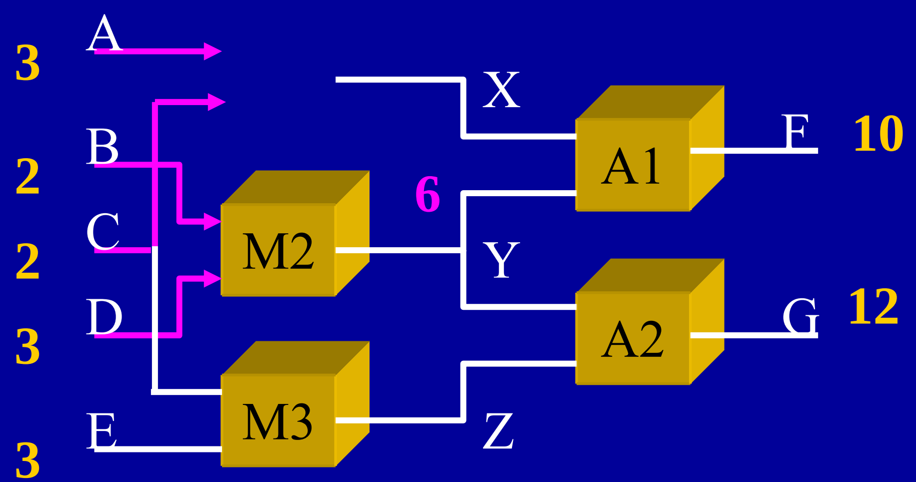
- Test: M1=U ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G



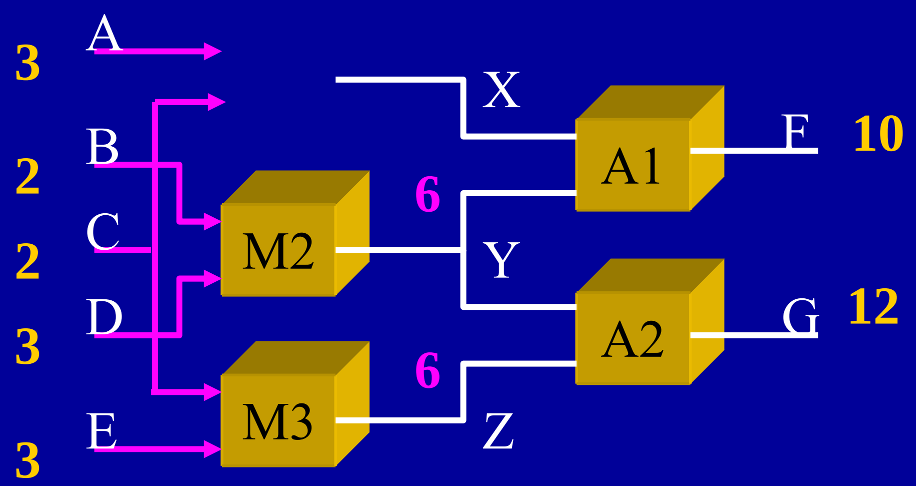
- Test: M1=U ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G



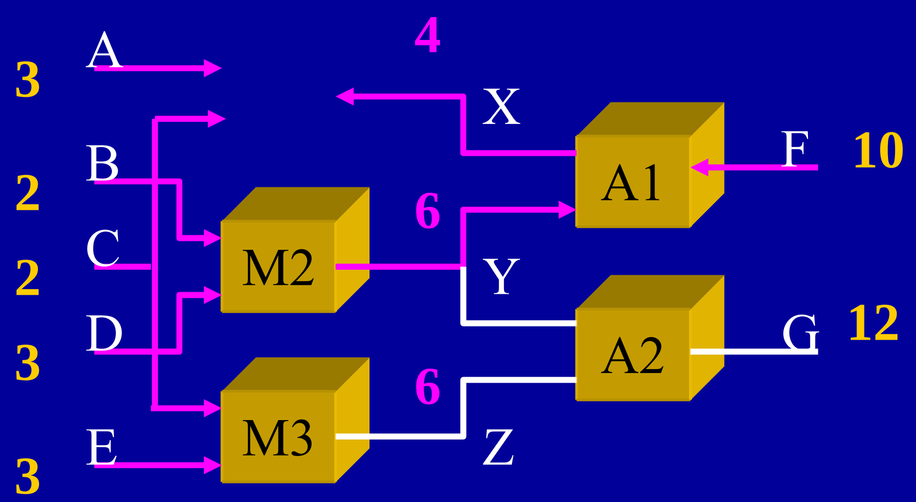
- Test: M1=U ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G



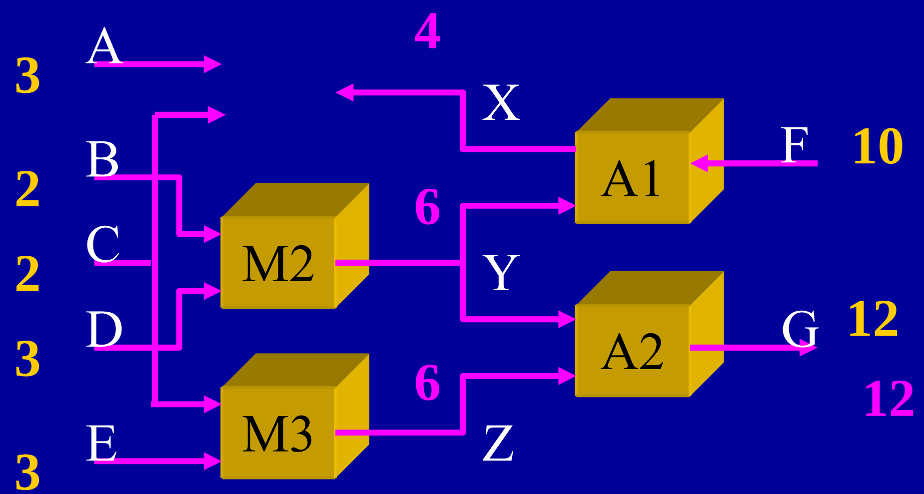
- Test: M1=U ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G



- Test: M1=U ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G



- Test: M1=U ∧ M2=G ∧ M3=G ∧ A1=G ∧ A2=G

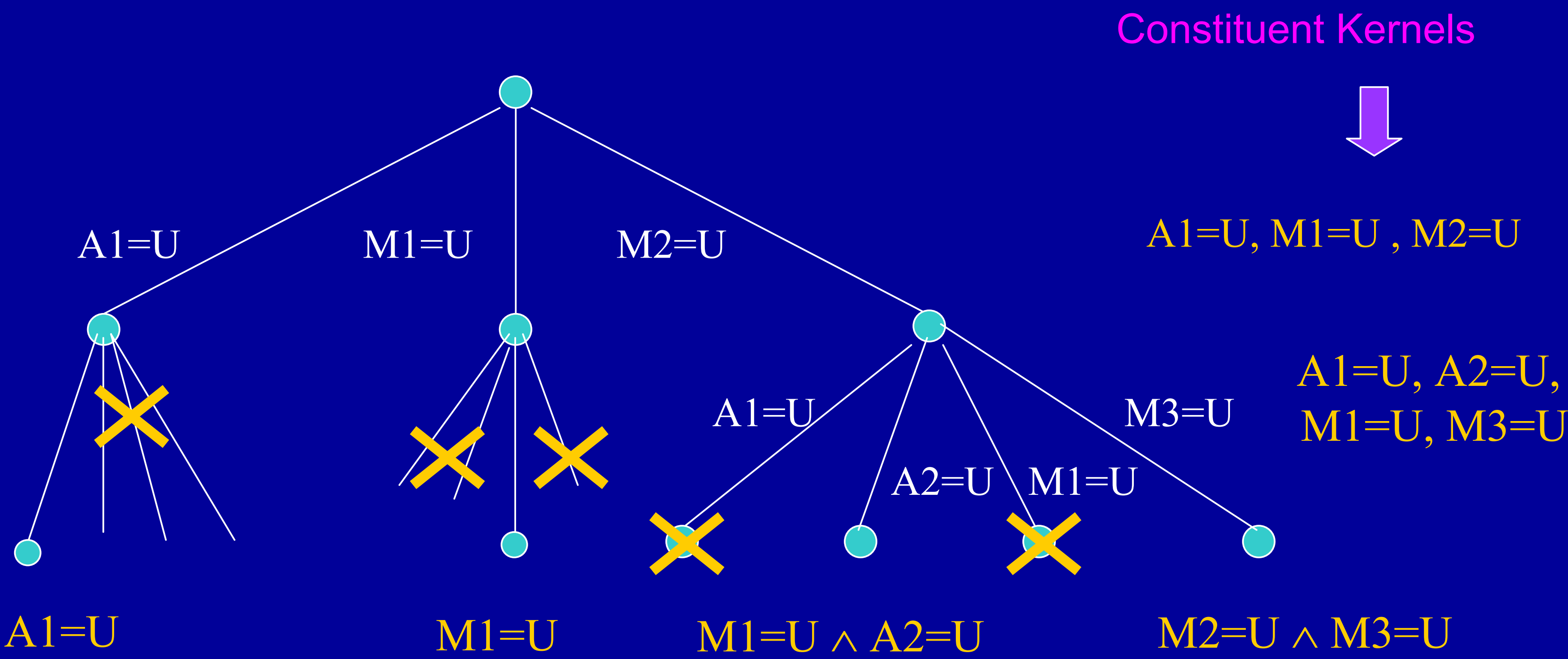


Consistent!

Outline

- Optimal CSPs
- Application to Model-based Execution
- Review of A*
- Conflict-directed A*
- Generating the Best Kernel
- Intelligent Tree Expansion
- Extending to Multiple Solutions
- Performance Comparison

Generating The Best Kernel of The Known Conflicts



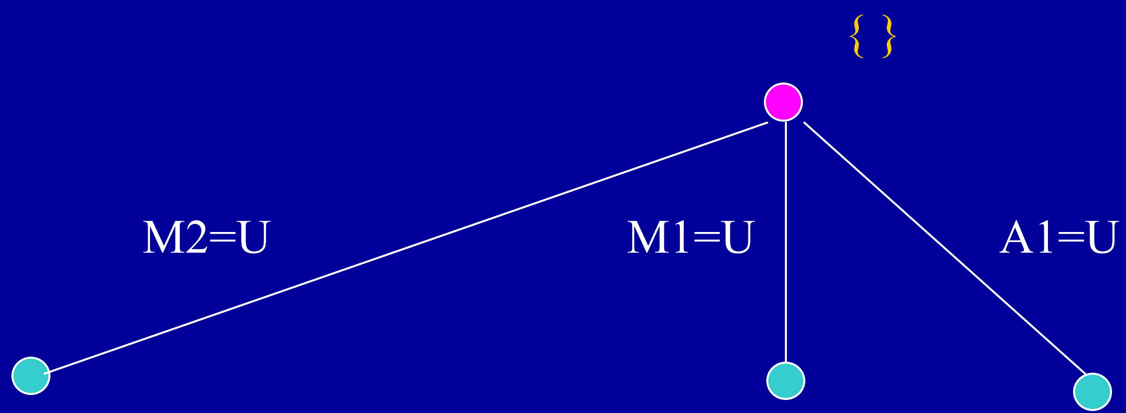
Insight:

- Kernels found by minimal set covering
- Minimal set covering is an instance of breadth first search.

Expanding a Node to Resolve a Conflict

Constituent kernels

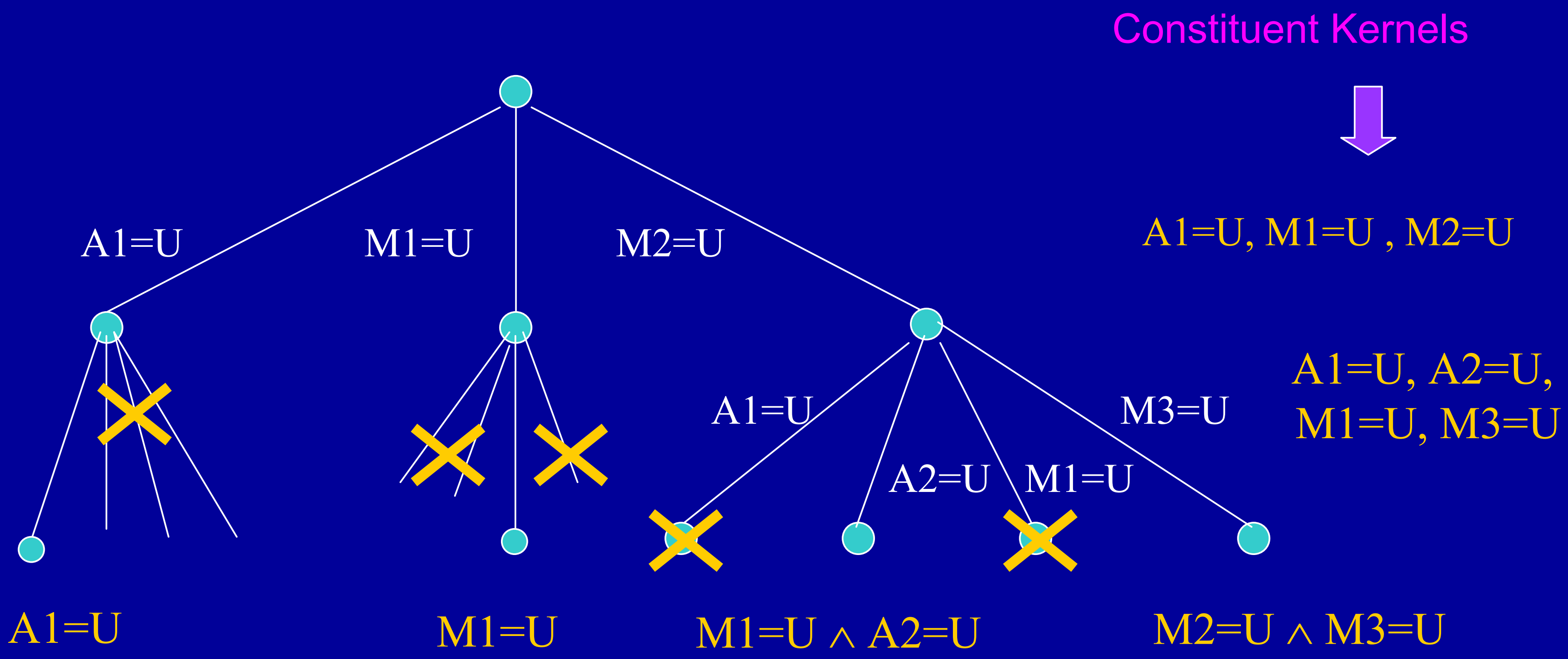
$M2{=}U \vee M1{=}U \vee A1{=}U$



To Expand a Node:

- Select an unresolved Conflict.
- Each child adds a constituent kernel.
- Prune child if state is
  - Inconsistent, or
  - subsumed by a known kernel (or another node's state).

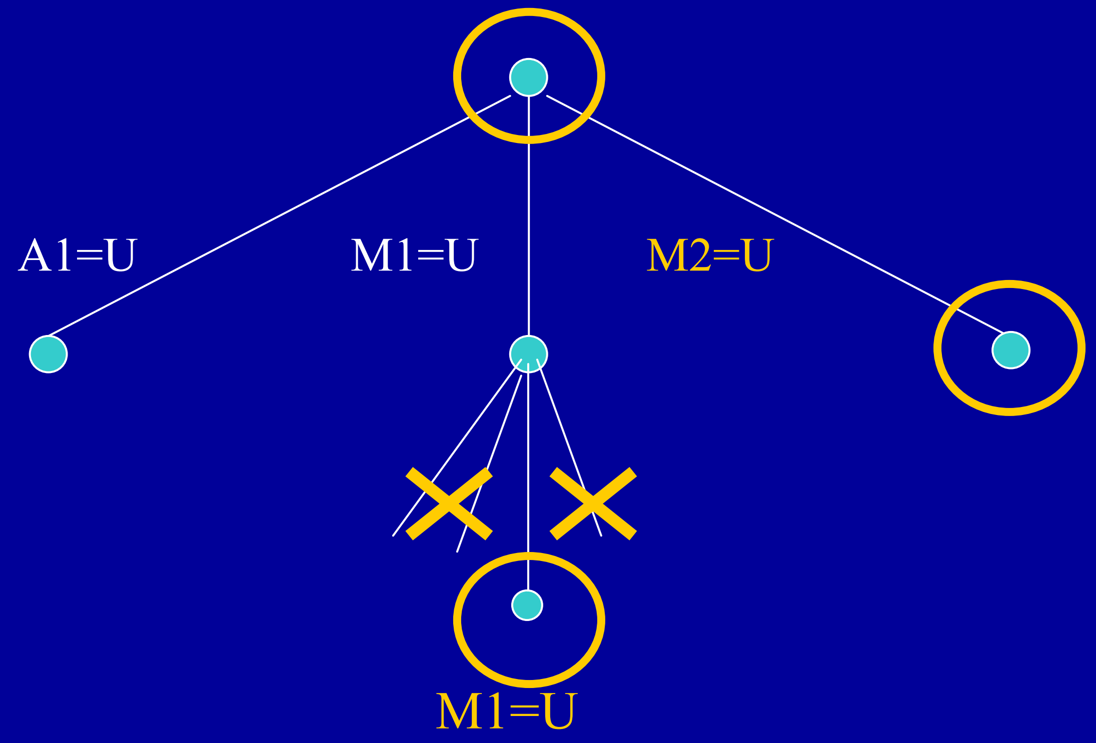
Generating The Best Kernel of The Known Conflicts



Insight:

- Kernels found by minimal set covering
- Minimal set covering is an instance of breadth first search.

Generating The Best Kernel of The Known Conflicts



Constituent Kernels

A1=U, M1=U , M2=U

A1=U, A2=U, M1=U, M3=U

Insight:

- Kernels found by minimal set covering
- Minimal set covering is an instance of breadth first search.

➔ To find the best kernel, expand tree in best first order.

Admissible h(α): Cost of best state extending partial assignment α

$f = g + h$

$M2=U \wedge M1=? \wedge M3=? \wedge A1=? \wedge A2=?$

$P_{M2=u} \times P_{M1=G} \times P_{M3=G} \times P_{A1=G} \times P_{A2=G}$

- Select best value of unassigned variables

Admissible Heuristic h

- Let $g = \langle G, g_i, Y \rangle$ describe a multi-attribute utility fn

- Assume the preference for one attribute x_i is independent of another x_k
  - *Called Mutual Preferential Independence*:

    For all $u, v \in Y$

    If $g_i(u) \geq g_i(v)$ then for all w

    $G(g_i(u), g_k(w)) \geq G(g_i(v), g_k(w))$

An Admissible h:

- Given a partial assignment, to $X \subseteq Y$
- h selects the best value of each unassigned variable $Z = X - Y$

$$h(Y) = G(\{g_{zi\_max} \mid z_i \in Z, \max_{v_{ij} \in D_{zi}} g_{zi}(v_{ij})\})$$

- A candidate always exists satisfying h(Y).

Terminate when all conflicts resolved

Function Goal-Test-Kernel (*node, problem*)
returns True IFF *node* is a complete decision state.
if forall K in Constituent-Kernels(Conflicts[*problem]),*
State[*node*] contains a kernel in K
then return True
else return False

Next Best Kernel of Known Conflicts

Function Next-Best-Kernel (*OCSP*)
returns the next best cost kernel of Conflicts[OCSP].
f(x) ← G[*OCSP*] (*g*[*OCSP*](x), *h[OCSP]*(x))
loop do
if Nodes[*OCSP*] is empty **then return** failure
node ← Remove-Best(Nodes[*OCSP*], f)
add State*[node]* to Visited[OCSP]
***new-nodes* ← Expand-Conflict(*node, OCSP*)**
for each *new-node* ∈ *new-nodes*
unless ∃ n ∈ Nodes[*OCSP*] such that State[*new-node*] = State[*n*]
OR State[*new-node*] ∈ Visited[*problem*]
then Nodes[*OCSP*] ← Enqueue(Nodes[*OCSP*], *new-node, f*)
if Goal-Test-Kernel[*OCSP*] applied to State[*node*] succeeds
Best-Kernels[*OCSP*]
← **Add-To-Minimal-Sets(Best-Kernels[OCSP], *best-kernel*)**
if *best-kernel* ∈ Best-Kernels[OCSP]
then return State[*node]*
end

An instance of A*

Outline

- Optimal CSPs
- Application to Model-based Execution
- Review of A*
- Conflict-directed A*
- Generating the Best Kernel
- Intelligent Tree Expansion
- Extending to Multiple Solutions
- Performance Comparison

Expand Only Best Child & Sibling

Constituent kernels

M2=U ∨ M1=U ∨ A1=U

Order constituents by decreasing utility

- Traditionally all children expanded.
- But only need to expand the child with the best candidate, if it can be identified apriori (how?).

⇨ This child is the one with the best estimated cost f = g+h.

Expand Only Best Child & Sibling

Constituent kernels

M2=U ∨ M1=U ∨ A1=U

Order constituents by decreasing utility

- Traditionally all children expanded.
- But only need to expand the child with the best candidate, if it can be identified apriori (how?).

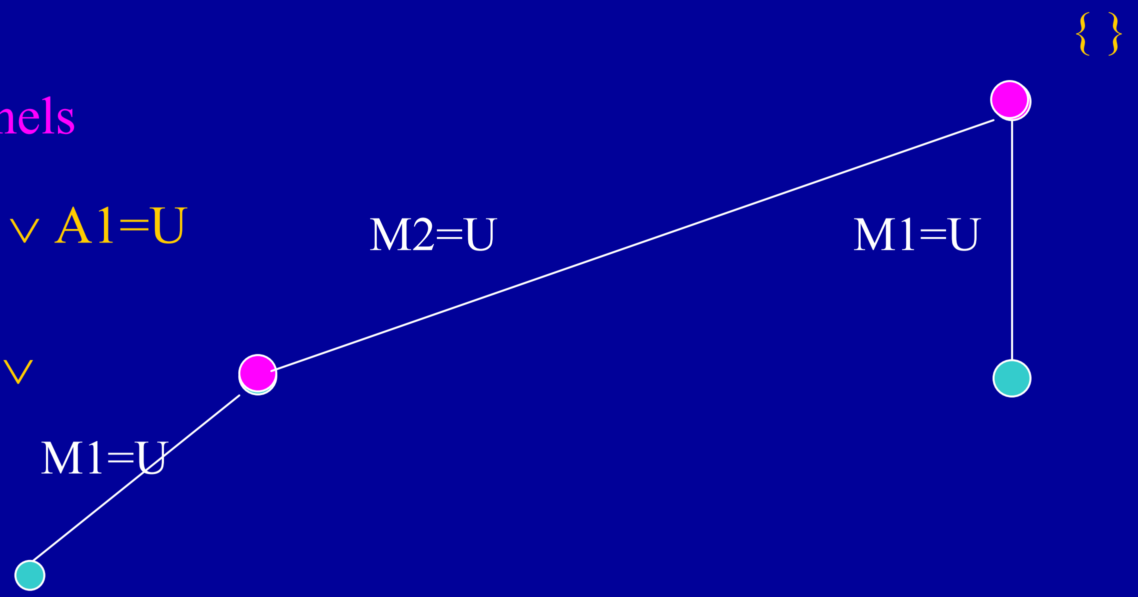
⇨ This child is the one with the best estimated cost f = g+h.

When Do We Expand The Childs Next Best Sibling?

Constituent kernels

M2=U ∨ M1=U ∨ A1=U

M1=U ∨ M3=U ∨ A1=U ∨ A2=U



- When a best child has a subtree or leaf pruned, it may have lost its best candidate.
- One of the child's siblings might now have the best candidate.

⇨ Expand child's next best sibling:

  - when child expanded in order to resolve another conflict.

Expand Node to Resolve Conflict

function Expand-Conflict(*node, OCSP*)
  return Expand-Conflict-Best-Child*(node, OCSP)* $\cup$
    Expand-Next-Best-Sibling (node, OCSP)

function Expand-Conflict-Best-Child(*node, OCSP)*
  if for all K_γ in Constituent-Kernels(Γ[OCSP])
    State[*node*] contains a kernel $\in K_\gamma$
  then return {}
  else return Expand-Constituent-Kernel*(node,OCSP)*

function Expand-Constituent-Kernel(*node, OCSP)*
  $K_\gamma \leftarrow$ = smallest uncovered set $\in$ Constituent-Kernels(Γ[*OCSP*])
  $C \leftarrow \{y_i = v_{ij} \mid \{y_i = v_{ij}\}$ in K_γ, $y_i = v_{ij}$ is consistent with State[*node*]}
  Sort *C* such that for all *i* from 1 to |*C*| - 1,
    Better-Kernel?(C[*i*],C[*i+1*], *OCSP*) is True
  Child-Assignments[*node]* $\leftarrow$ *C*
  $y_i = v_{ij} \leftarrow$ *C*[1], *which is the best kernel in K_γ consistent with State[*node*]*
  return {Make-Node({$y_i = v_{ij}$}, node)}

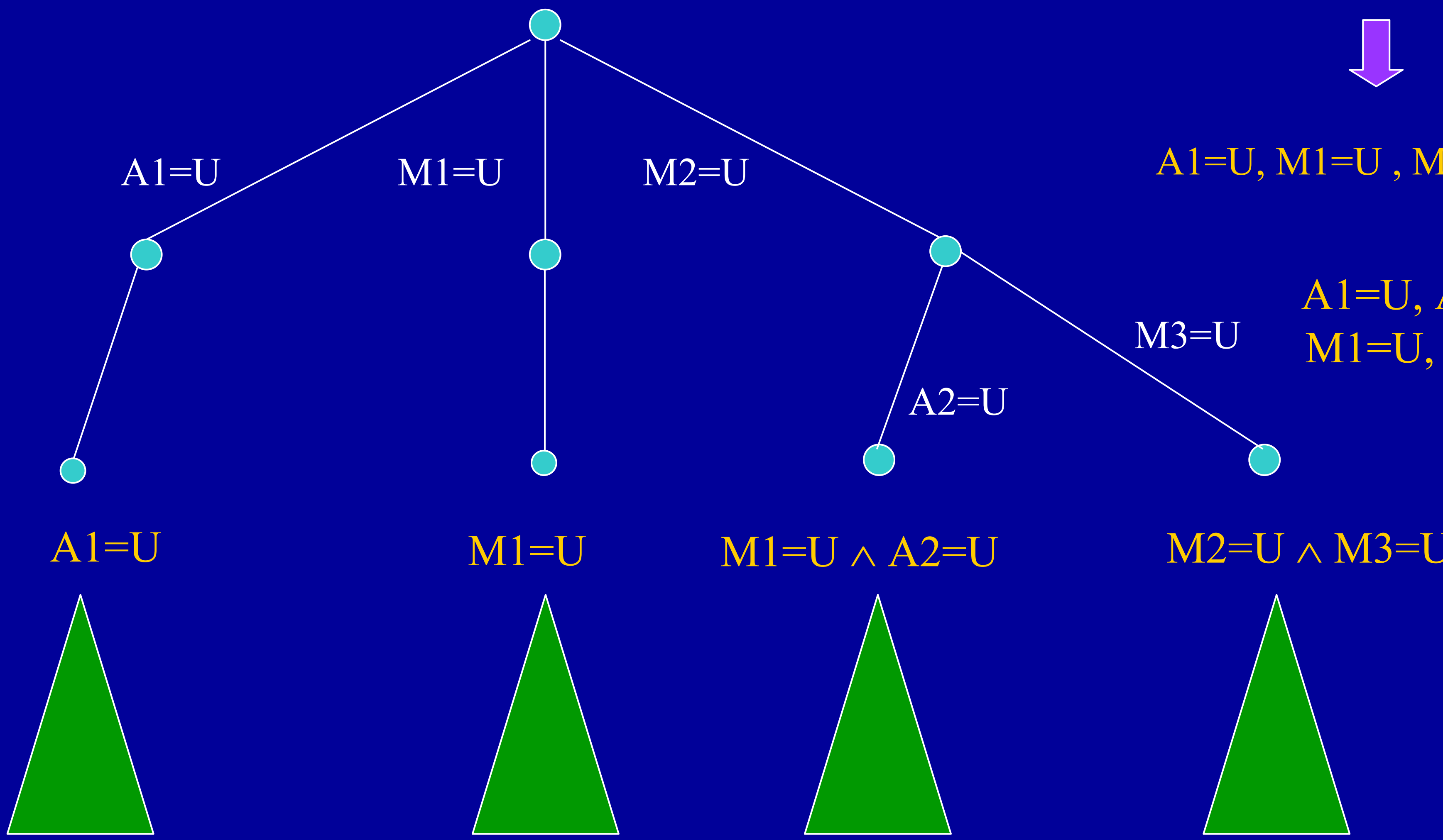
Expand Node to Resolve Conflict

function Expand-Next-Best-Sibling(*node, OCSP)*
if Root?[*node*]
then return {}
else $\{y_i = v_{ij}\} \leftarrow$ Assignment[*node*]
$\{y_k = v_{kl}\} \leftarrow$ next best assignment in consistent child-assignments[Parent[*node*]] after $\{y_i = v_{ij}\}$
if no next assignment $\{y_k = v_{kl}\}$
or **Parent**[*node*] already has a child with $\{y_k = v_{kl}\}$
then return {}
else return {Make-Node($\{y_k = v_{kl}\}$, Parent[*node*])}

Outline

- Optimal CSPs
- Application to Model-based Execution
- Review of A*
- Conflict-directed A*
- Generating the Best Kernel
- Intelligent Tree Expansion
- Extending to Multiple Solutions
- Performance Comparison

Multiple Solutions: Systematically Exploring Kernels



Constituent Kernels

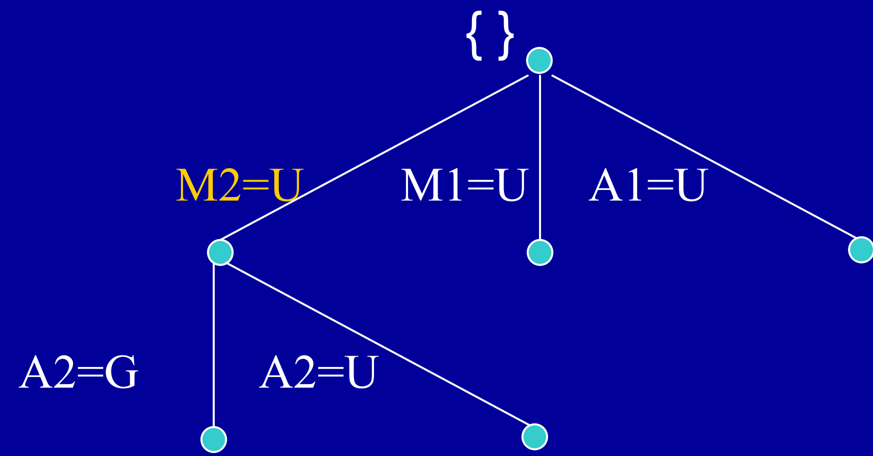
A1=U, M1=U , M2=U

A1=U, A2=U, M1=U, M3=U

Child Expansion For Finding Multiple Solutions

Conflict

$\neg (M2=G \wedge M1=G \wedge A1=G)$



If Unresolved Conflicts:

- Select unresolved conflict.
- Each child adds a constituent kernel.

If All Conflicts Resolved:

- Select unassigned variable y_i.
- Each child adds an assignment from D_i.

Intelligent Expansion Below a Kernel

Select Unassigned Variable

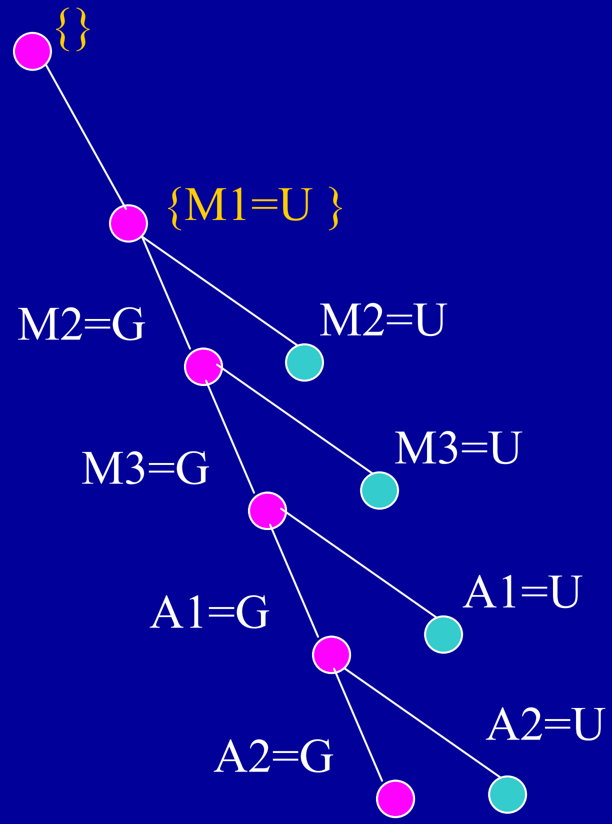
M2=G ∨ M2=U

Order assignments by decreasing utility

Expand best child

Continue expanding best descendents

When leaf visited, expand all next best ancestors. (why?)

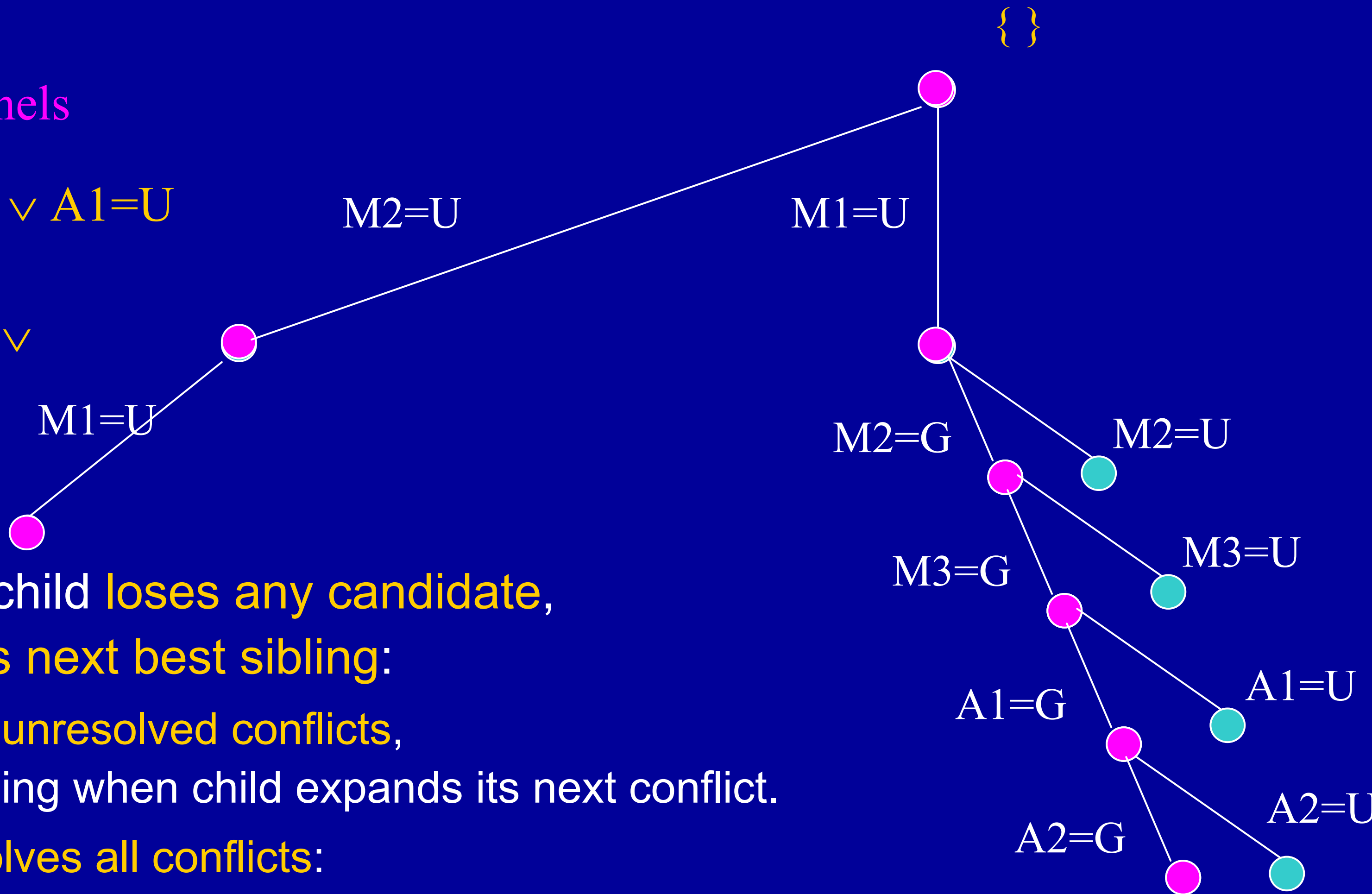


Putting It Together: Expansion Of Any Search Node

Constituent kernels

M2=U ∨ M1=U ∨ A1=U

M1=U ∨ M3=U ∨ A1=U ∨ A2=U



- When a best child loses any candidate, expand child's next best sibling:
  - If child has unresolved conflicts, expand sibling when child expands its next conflict.
  - If child resolves all conflicts: expand sibling when child expands a leaf.

Outline

- Optimal CSPs
- Application to Model-based Execution
- Review of A*
- Conflict-directed A*
- Generating the Best Kernel
- Intelligent Tree Expansion
- Extending to Multiple Solutions
- Performance Comparison

Performance: With and Without Conflicts

Problem Parameters				Constraint-based A* (no conflicts)		Conflict-directed A*			Mean CD-CB Ratio	
Dom Size	Dec Vars	Clau -ses	Clau -se lngth	Nodes Expanded	Queue Size	Nodes Expand	Queue Size	Conflicts used	Nodes Expanded	Queue Size
5	10	10	5	683	1,230	3.3	6.3	1.2	4.5%	5.6%
5	10	30	5	2,360	3,490	8.1	17.9	3.2	2.4%	3.5%
5	10	50	5	4,270	6,260	12.0	41.3	2.6	0.83%	1.1%
10	10	10	6	3,790	13,400	5.7	16.0	1.6	2.0%	1.0%
10	10	30	6	1,430	5,130	9.7	94.4	4.2	4.6%	5.8%
10	10	50	6	929	4,060	6.0	27.3	2.3	3.5%	3.9%
5	20	10	5	109	149	4.2	7.2	1.6	13.0%	13.0%
5	20	30	5	333	434	6.4	9.2	2.2	6.0%	5.4%
5	20	50	5	149	197	5.4	7.2	2.0	12.0%	11.0%

Conflict-directed A*

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

- Sherlock Holmes. The Sign of the Four.

1. Test Hypothesis
2. If inconsistent, learn reason for inconsistency (a Conflict).
3. Use conflicts to leap over similarly infeasible options to next best hypothesis.

Presentation Notes

- Change Example to Boolean Polycell
- Introduce CDA* before Sherlock-style Mode Estimation.
- Describe Kernels and Conflicts in terms of set/subset lattice.
- More Intuitive and focused introduction to A*
- Add systematicity in each development
- Add pseudo code for multiple solns and CBA*
- Show full search trees for each
- Highlight Important features of performance