Introducing Constrained Heuristic Search to the Soar Cognitive Architecture

(29th Soar Workshop, University of Michigan)

Sean A. Bittle
Mark S. Fox

June 25th, 2009

university of toronto
mechanical & industrial engineering

Agenda

- Introduction: The problem, objectives of research
- Review:
  - Soar Cognitive Architecture
  - Constraint Satisfaction Problems (CSP)
  - Hyper-Heuristics
  - Constrained Heuristic Search (CHS)
- Design of CHS-Soar
  - Learning via subgoaling and chunks (Soar 8.6.3)
    - Experiments and Results
  - Reinforcement learning (Soar 9.0)
- Future Work and Issues

Introduction

General problem solving and domain independent learning are central goals of AI research on cognitive architectures.

Problem:

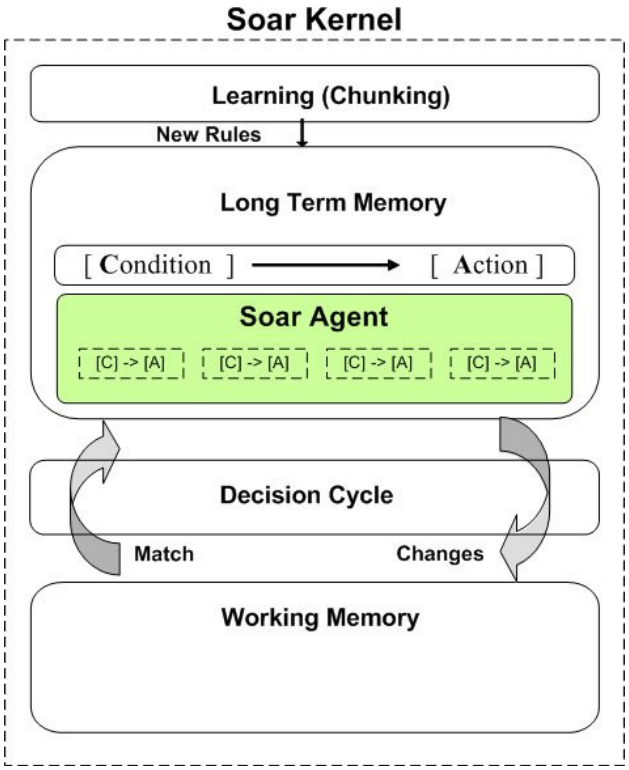
- However, there are few examples of domain independent learning in cognitive architectures

Objective:

- Demonstrate Soar can learn and apply domain independent knowledge

Soar Cognitive Architecture

- Current work based on Soar version 8.6.3, and Soar 9.0

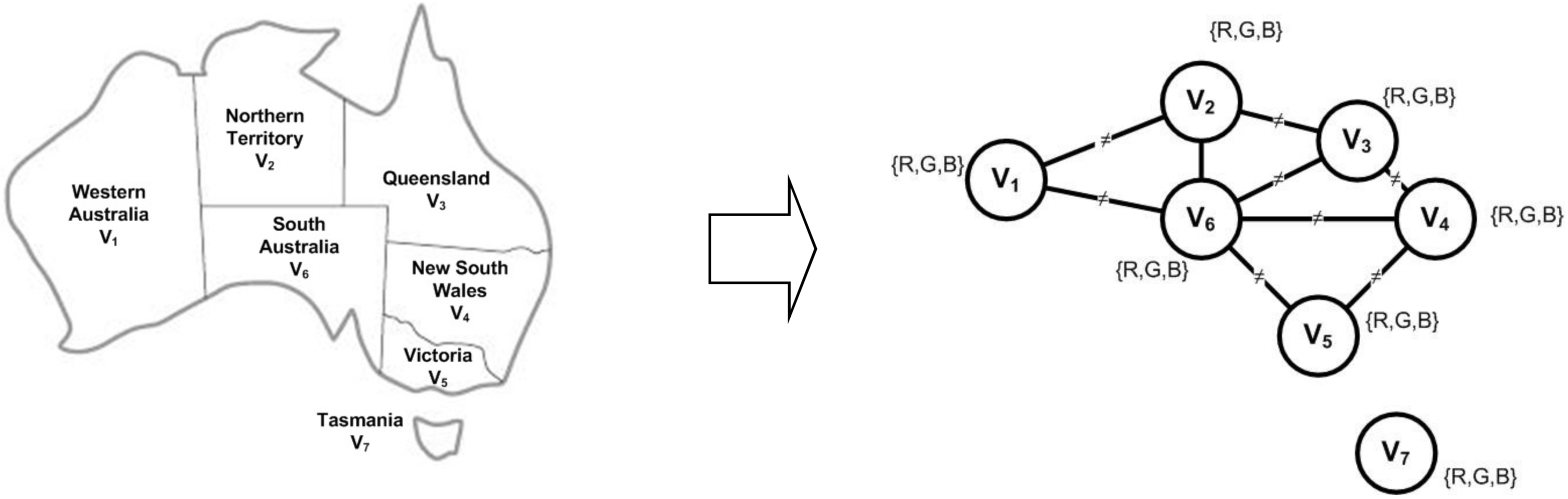


Constraint Satisfaction Problems (CSP)

- Constraint satisfaction is a sub-domain of constraint programming dealing with problems defined over a finite domain

- More formally, CSP consists of a finite set of:
  - Variables (X_1, X_2,…X_n)
  - Constraints (C_1, C_2,…C_n)
  - Each variable has a finite domain D_i of possible values

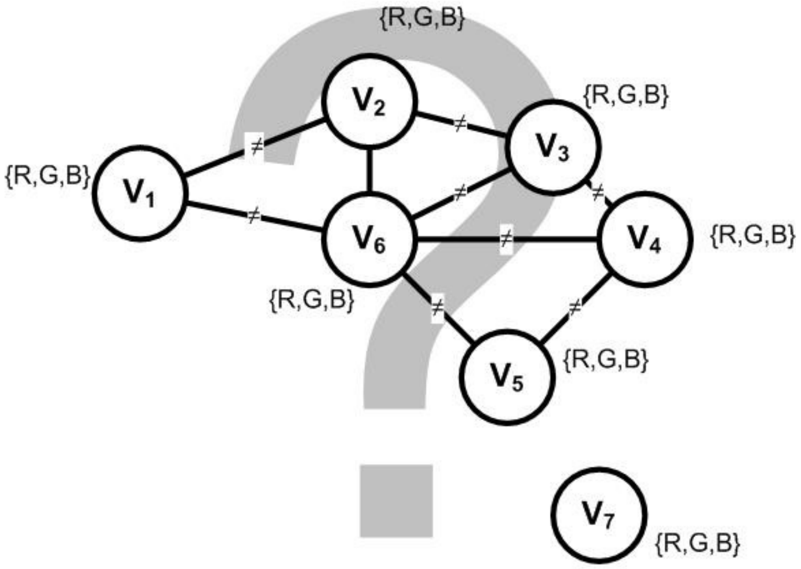
- Useful to represent CSP as a binary constraint graph



Constraint Satisfaction Problems (CSP)

- Backtrack search is the general approach used to solve a CSP
- General-purpose methods can provide ways to improve backtrack search efficiency:
  - Can we detect inevitable failure early? → Propagation
  - Which variable should be assigned next? → Variable Ordering
  - In what order should its values be tried? → Value Ordering

Use heuristics to guide variable and value ordering



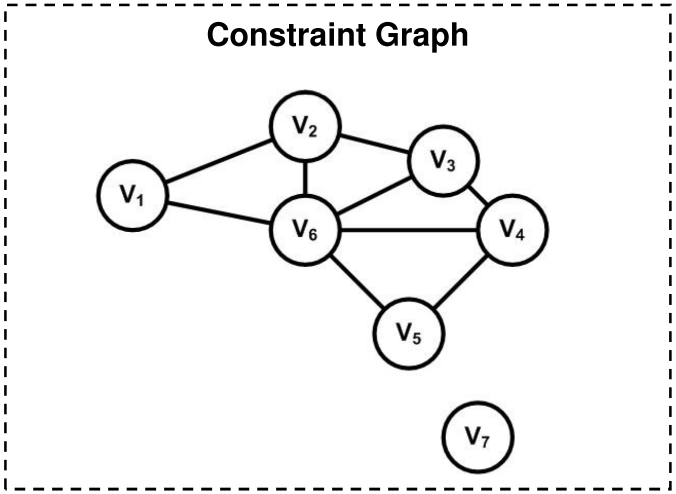
Hyper-Heuristics

Problem with variable and value ordering heuristics is "effective generality"

- Hyper-Heuristics are *"Heuristics to Choose Heuristics"*
- A hyper-heuristic is a high-level heuristic which uses some type of learning mechanism in order to choose (switch) between various low-level heuristics
- Most popular learning approach based on using a Genetic Algorithm (GA)

Constrained Heuristic Search (CHS)

- Developed by Fox, Sadeh, Bayken, 1989
- CHS is a problem solving approach that combination of constraint satisfaction and heuristic search where the definition of the problem space is refined to include:
  - ***Problem Topology***
  - ***Problem Textures***
  - ***Problem Objective***

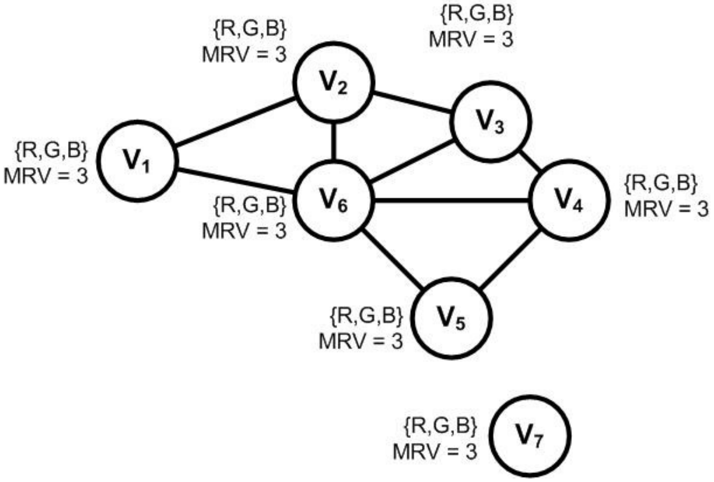


Constrained Heuristic Search (CHS)

What are Texture Measures?

- A texture measurement is a technique for distilling information embedded in the constraint graph into a form that heuristics can use
- A texture measurement is not a heuristic itself, but can be considered the constituent parts of a heuristic

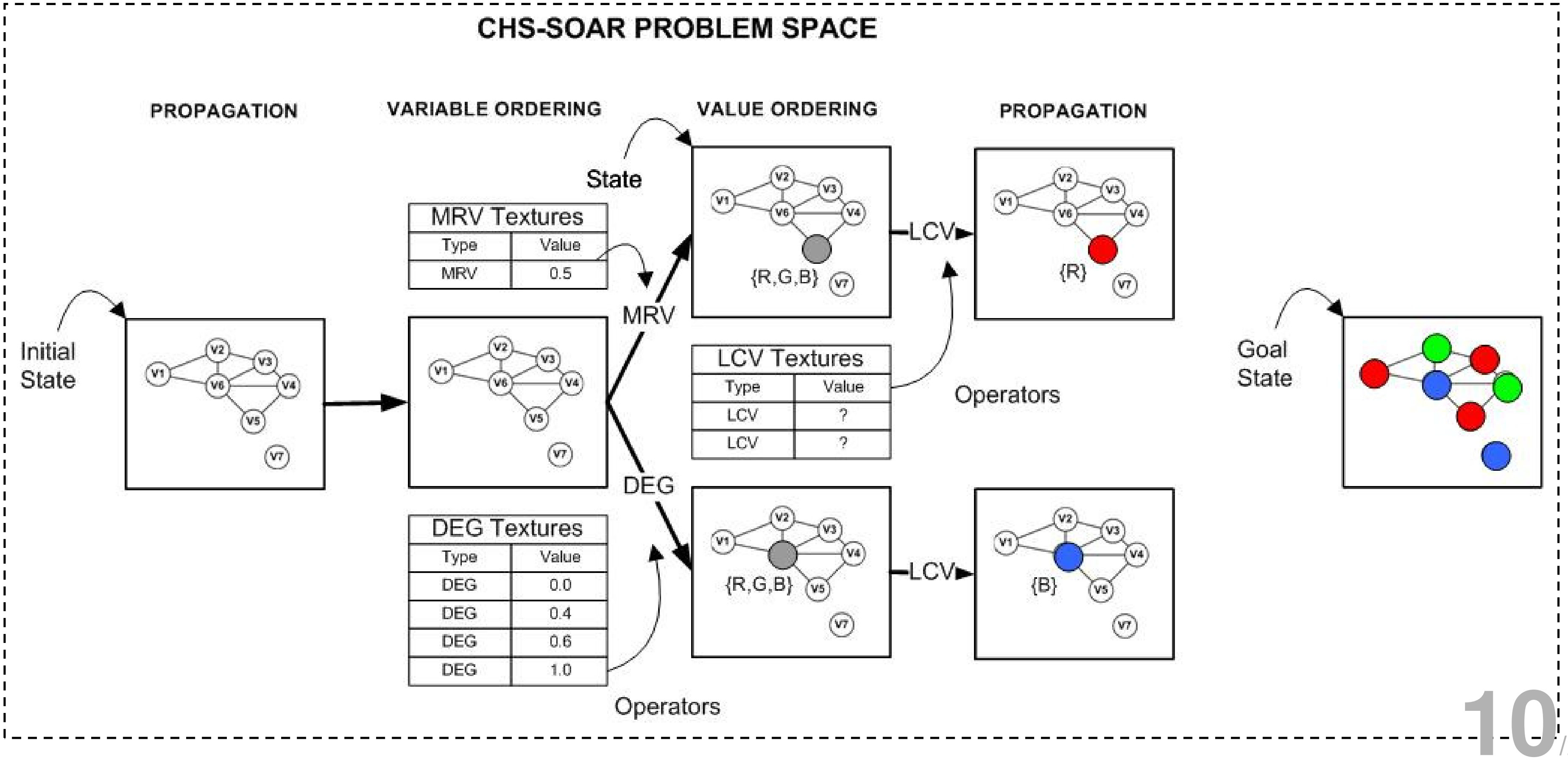
Ordering	Name	Texture	Heuristic
Variable	Minimum Remaining Values (MRV)	D_i, number of remaining values in domain of variable.	Select the variable with the smallest D_i, value e.g. pick the variable with the fewest legal values.



Design of CHS-Soar

"How Does CHS-Soar Solve Problems?"

CHS-Soar problem solving is formulated by applying operators to states within a problem space in order to achieve a goal



MRV Textures

Type	Value
MRV	0.5

LCV Textures

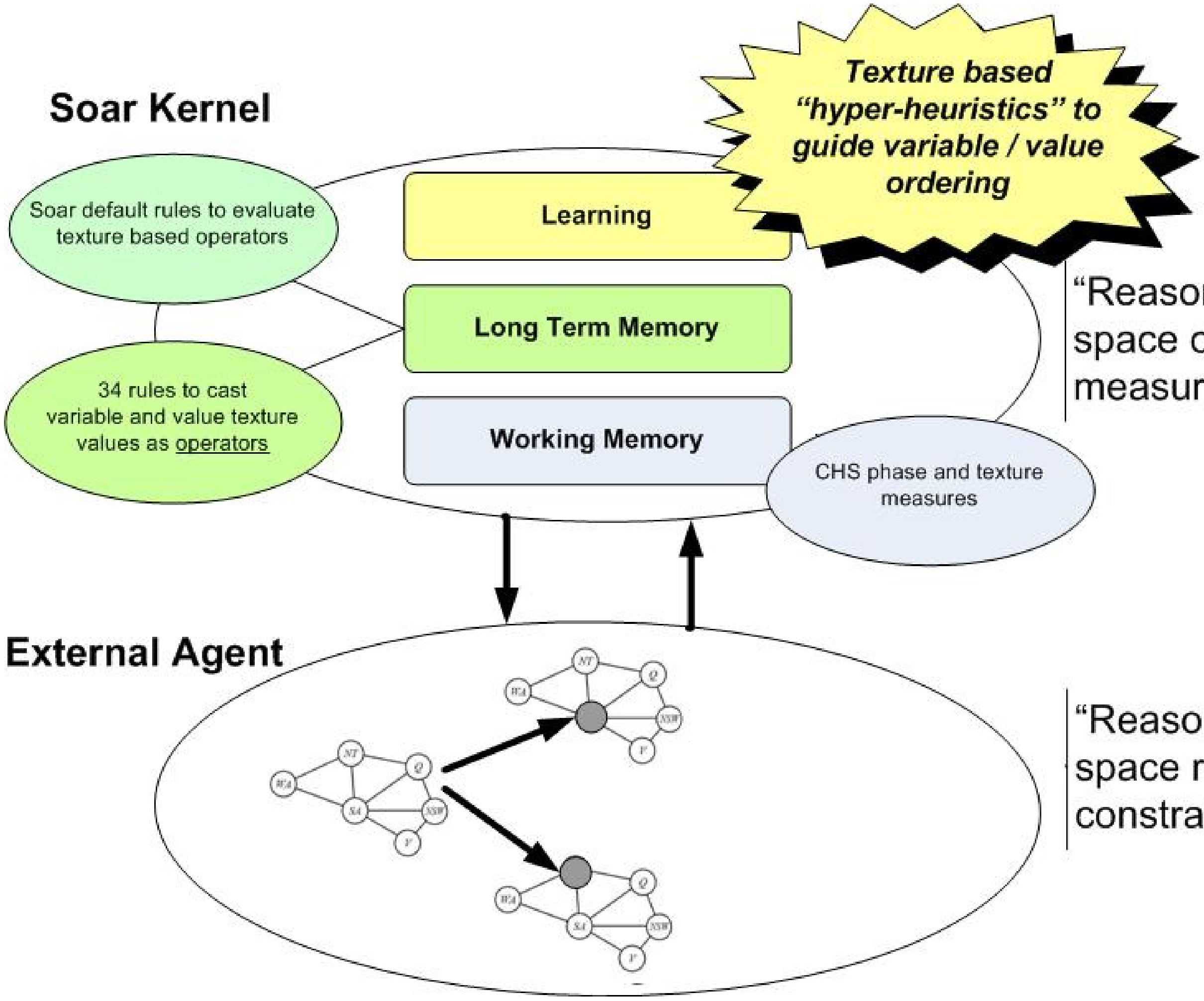
Type	Value
LCV	?
LCV	?

DEG Textures

Type	Value
DEG	0.0
DEG	0.4
DEG	0.6
DEG	1.0

Design of CHS-Soar

"What are we trying to Learn?"



Soar Kernel

Soar default rules to evaluate texture based operators

34 rules to cast variable and value texture values as operators

Learning

Long Term Memory

Working Memory

CHS phase and texture measures

Texture based "hyper-heuristics" to guide variable / value ordering

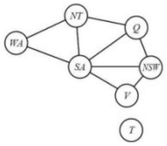
External Agent

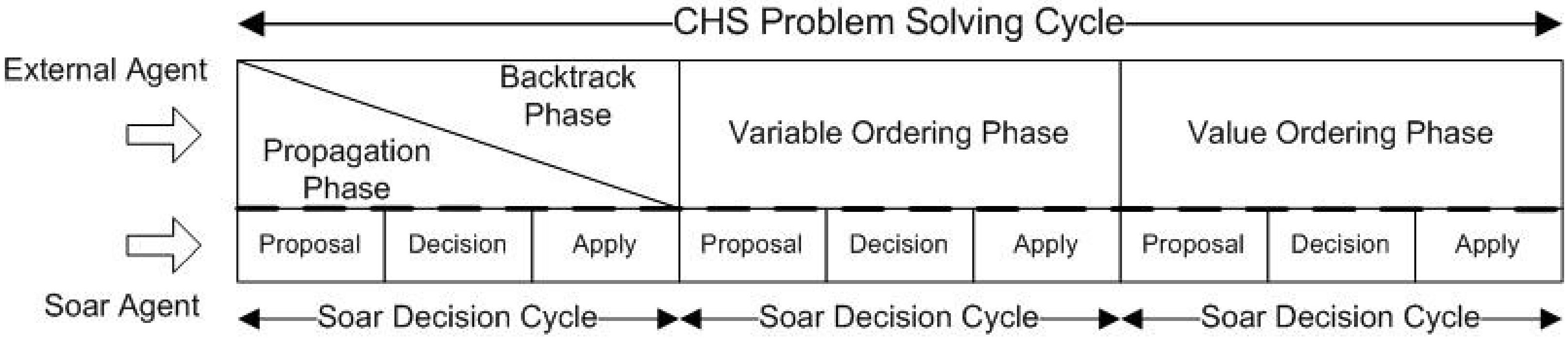
"Reasons" over a state space of texture measures

"Reasons" over a state space represented by a constraint graph

Design of CHS-Soar

"How does CHS-Soar Solve Problems (and Learn)?"





CHS Problem Solving Cycle

External Agent

Backtrack Phase

Propagation Phase

Variable Ordering Phase

Value Ordering Phase

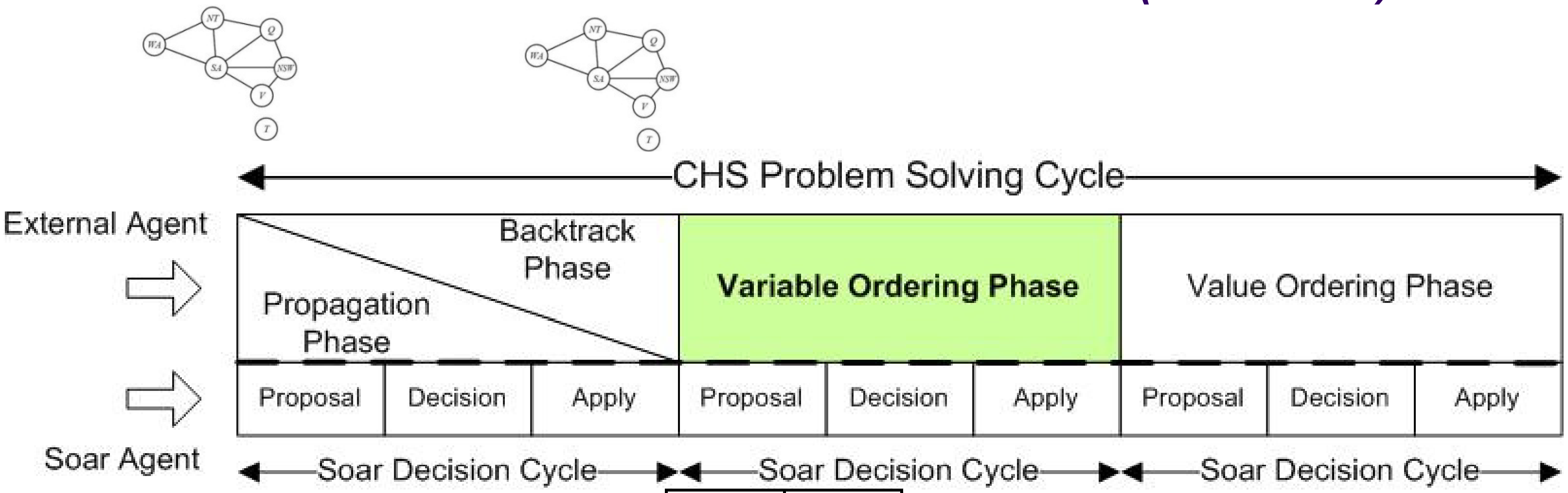
Proposal | Decision | Apply | Proposal | Decision | Apply | Proposal | Decision | Apply

Soar Agent

Soar Decision Cycle | Soar Decision Cycle | Soar Decision Cycle

Design of CHS-Soar

"How does CHS-Soar Solve Problems (and Learn)?"

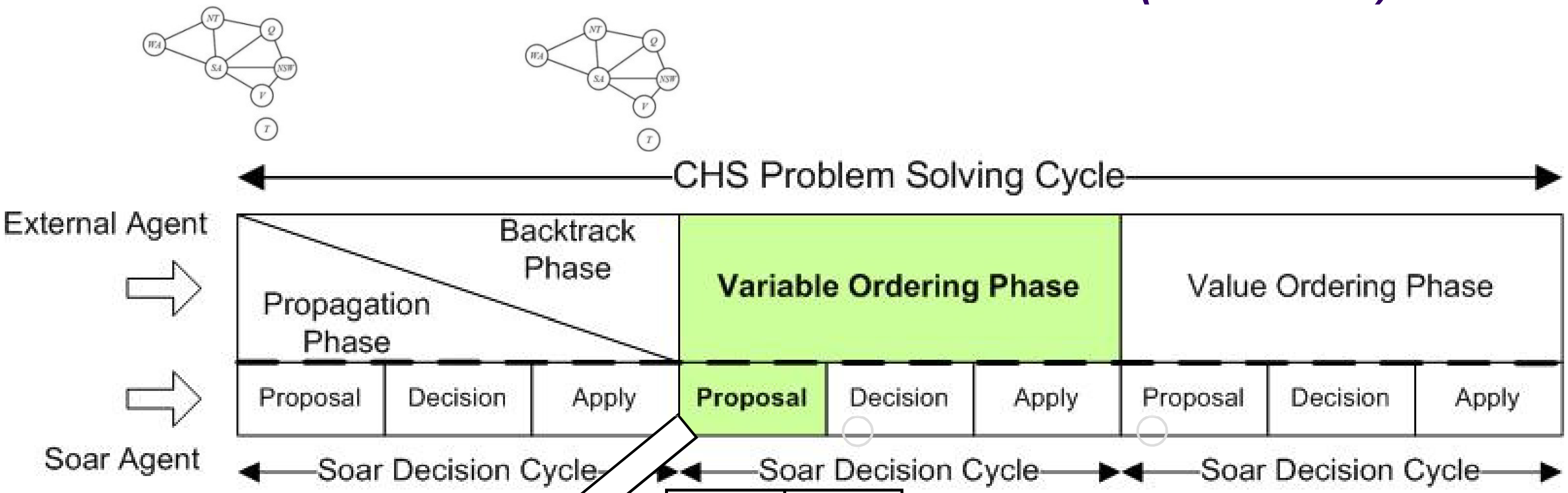


Type	Value
MRV	0.50
DEG	0.00
DEG	0.40
DEG	0.60
DEG	1.00

"pruned" and normalized

Design of CHS-Soar

"How does CHS-Soar Solve Problems (and Learn)?"



Type	Value
MRV	0.50
DEG	0.00
DEG	0.40
DEG	0.60
DEG	1.00

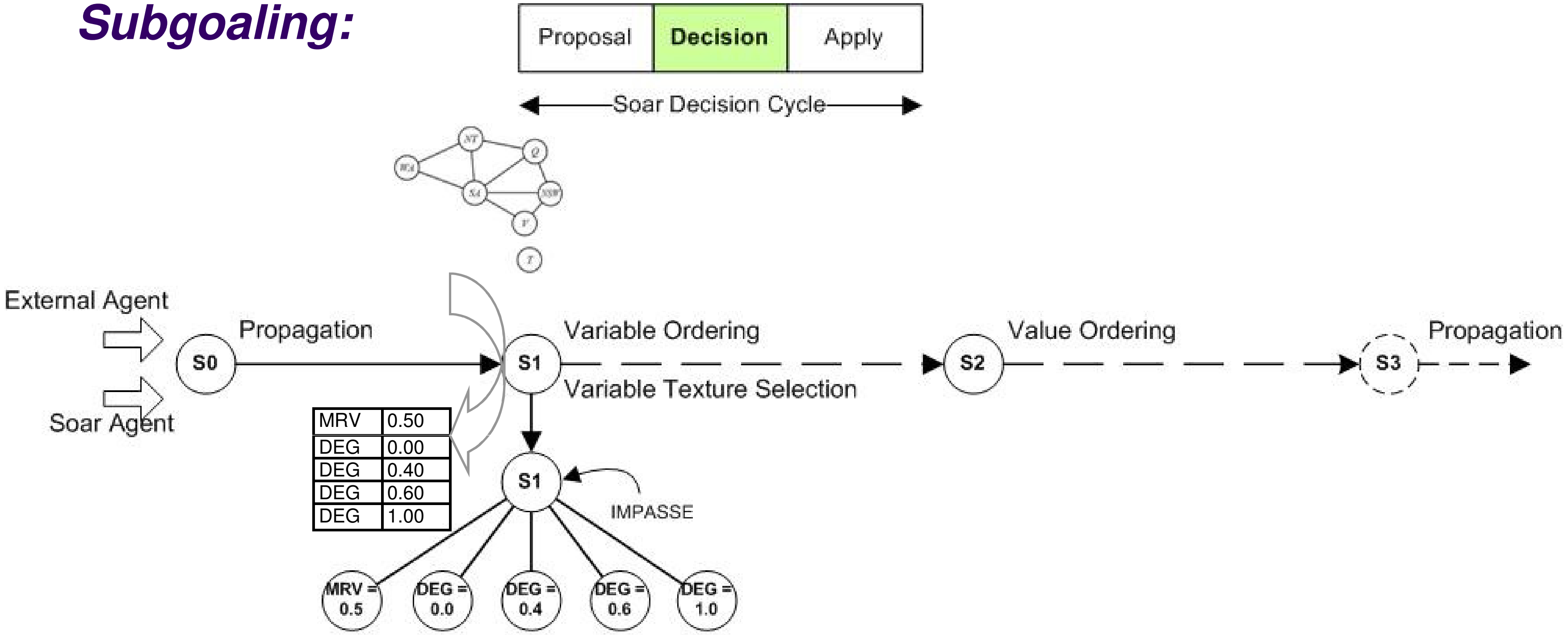
```
sp {apply*initialize
:
-->
:
 ^vartextype DEG MRV
 ^valtextype LCV
:
}
```

Variable Texture Propose Rule

```
sp {propose*SelectVariableTexture
   (state <s> ^name CHS-Soar
              ^problem-space <p>
              ^phase SelectVariableTexture
              ^top-state.vartextype <type>
              ^<type> <value> { <value> <= 1.0 }  )
  -->
        (<s> ^operator <op> + )
       (<op> ^name SelectVariableTexture
             ^type <type>
             ^value <value> )   }
```

Design of CHS-Soar

Subgoaling:

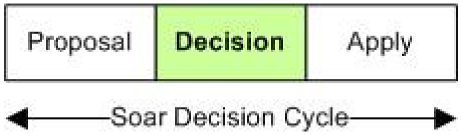


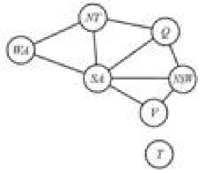
MRV	0.50
DEG	0.00
DEG	0.40
DEG	0.60
DEG	1.00

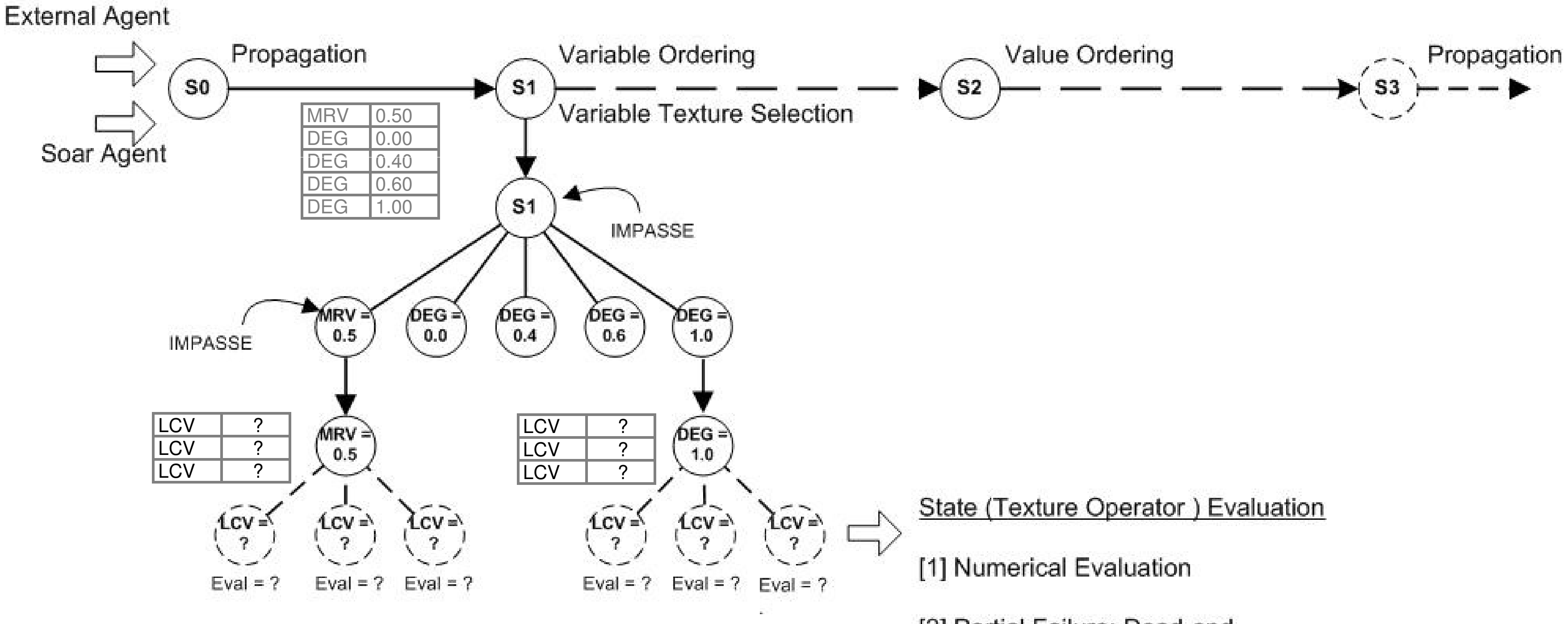
Which (VAR texture measure) Operator to Select?

Design of CHS-Soar

Subgoaling:







MRV	0.50
DEG	0.00
DEG	0.40
DEG	0.60
DEG	1.00

LCV	?
LCV	?
LCV	?

LCV	?
LCV	?
LCV	?

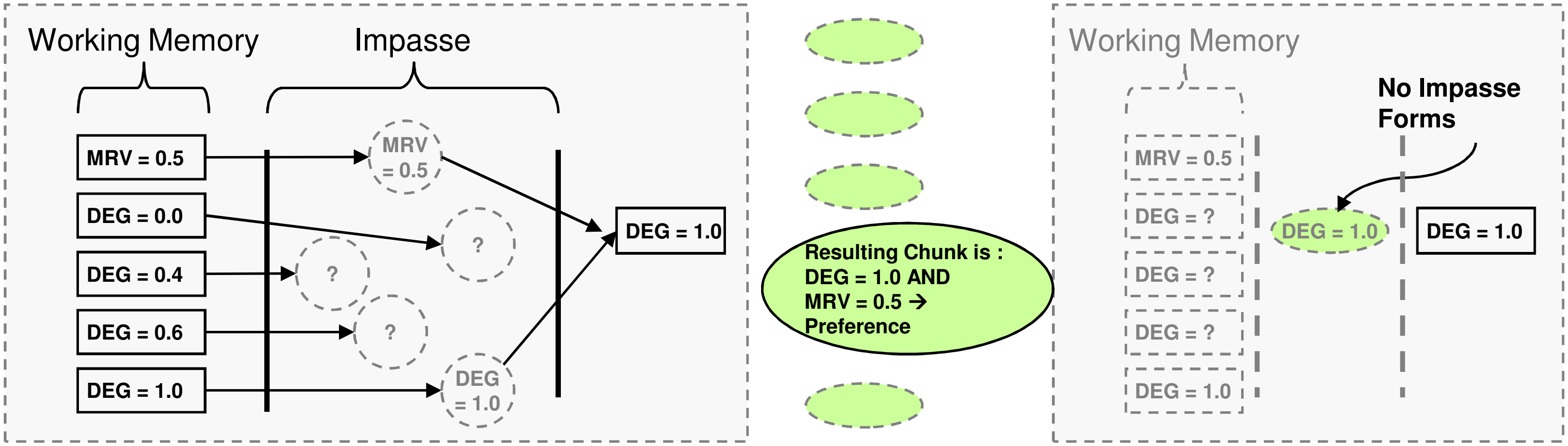
State (Texture Operator) Evaluation

[1] Numerical Evaluation

[2] Partial Failure: Dead-end

Design of CHS-Soar

Subgoaling-Chunking:



Standard Soar Chunk (Water Jugs)

```
sp {chunk-54*d150*tie*2
    :chunk
    (state <s1> ^name water-jug ^operator <o1> +
         ^problem-space <p1>
         ^desired <d1> ^jug <j1> ^jug <j2>)
    (<o1> ^name fill ^jug <j1>)
    (<p1> ^name water-jug)
    (<j1> ^contents 0 ^volume 3)
    (<j2> ^contents 0 ^volume 5)
    (<d1> ^jug <j3>)
    (<j3> ^contents 1 ^volume 3)
    -->
    (<s1> ^operator <o1> >) }
```

CHS-Soar Binary Chunk (decoupled from problem type)

```
sp {chunk-514*d513*tie*4
    :chunk
    (state <s1> ^phase |SelectVariableTexture|
          ^top-state <s1> ^name |CHS-Soar| ^desired <d1>
          ^operator <o1> + ^operator <o2> + ^problem-space <p1>)
    (<d1> ^better higher)
    (<o1> ^value 1.   ^name |SelectVariableTexture| ^type |DEG|)
    (<o2> ^value 0.14 ^name |SelectVariableTexture| ^type |MRV|)
    (<p1> ^name |CHS-Soar|)
    -->
    (<s1> ^operator <o2> < <o1>) }
```

Experiments and Results

Experiments conducted to investigate:

1. Intra (e.g. within) problem type learning and problem solving
2. Inter (e.g. across) problem type learning and problem solving

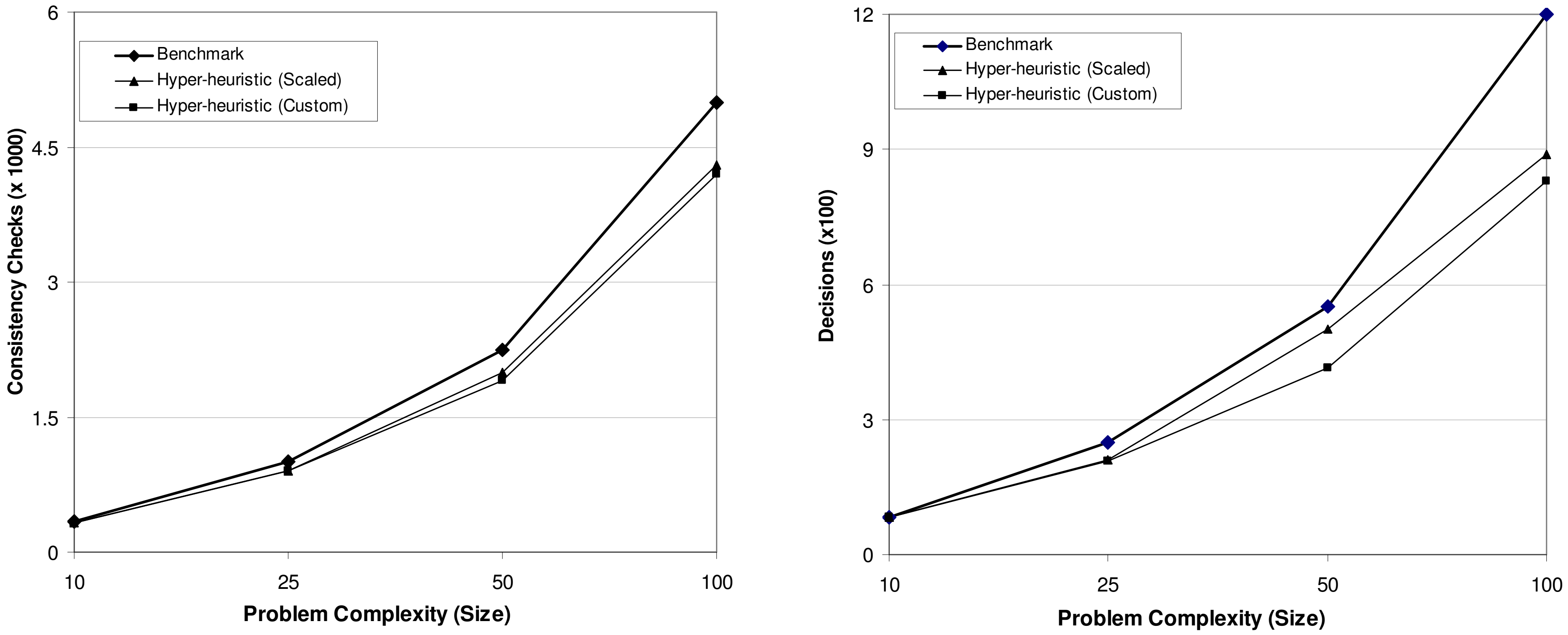
Problem types considered to date:

- Towers of Hanoi, Water Jugs
- Job Shop Scheduling (JSS)
- Map Coloring
- Radio Frequency Assignment Problem (RFAP)
- N-Queens
- Random CSP's
- Vehicle Routing Problem (VRP)

Experiment 1:

Intra-Problem Solving and Learning

Map Coloring Problem

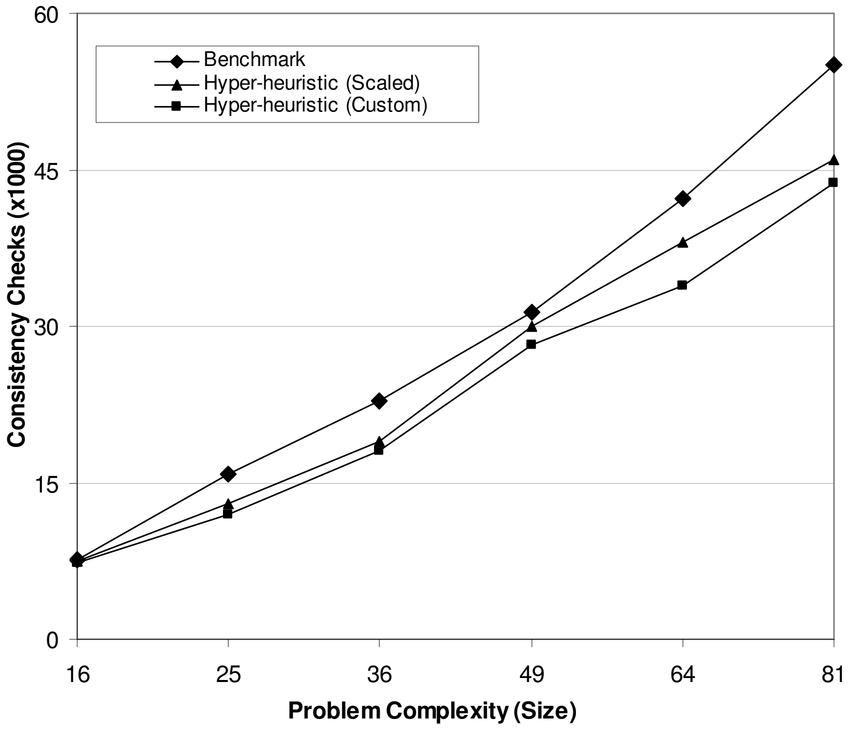


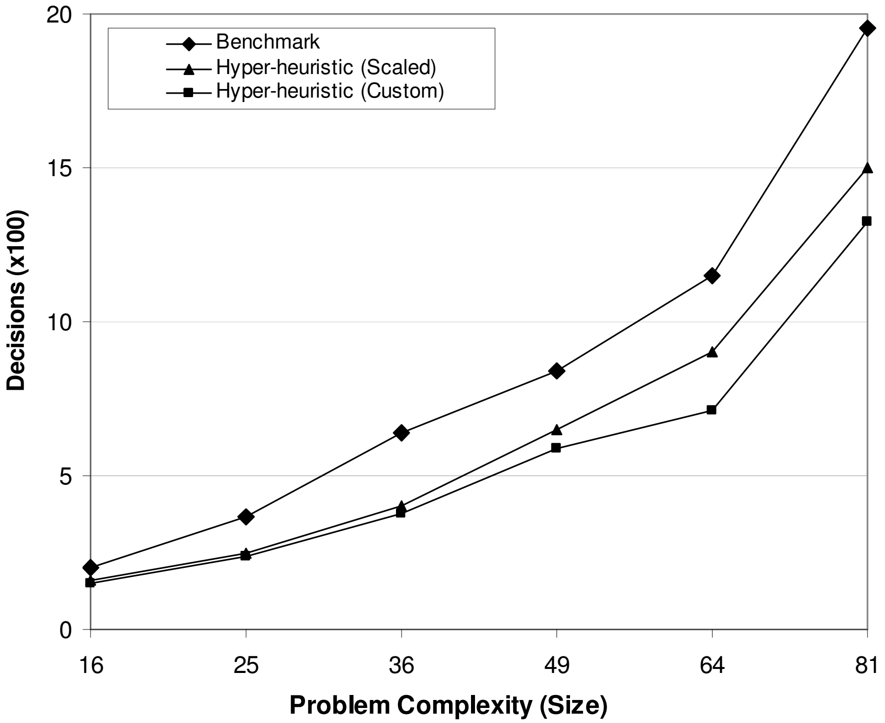
Learned non-max/min texture based rules delivery superior problem-solving performance over traditional heuristics

Experiment 1:

Intra-Problem Solving and Learning

Job Shop Scheduling Problem



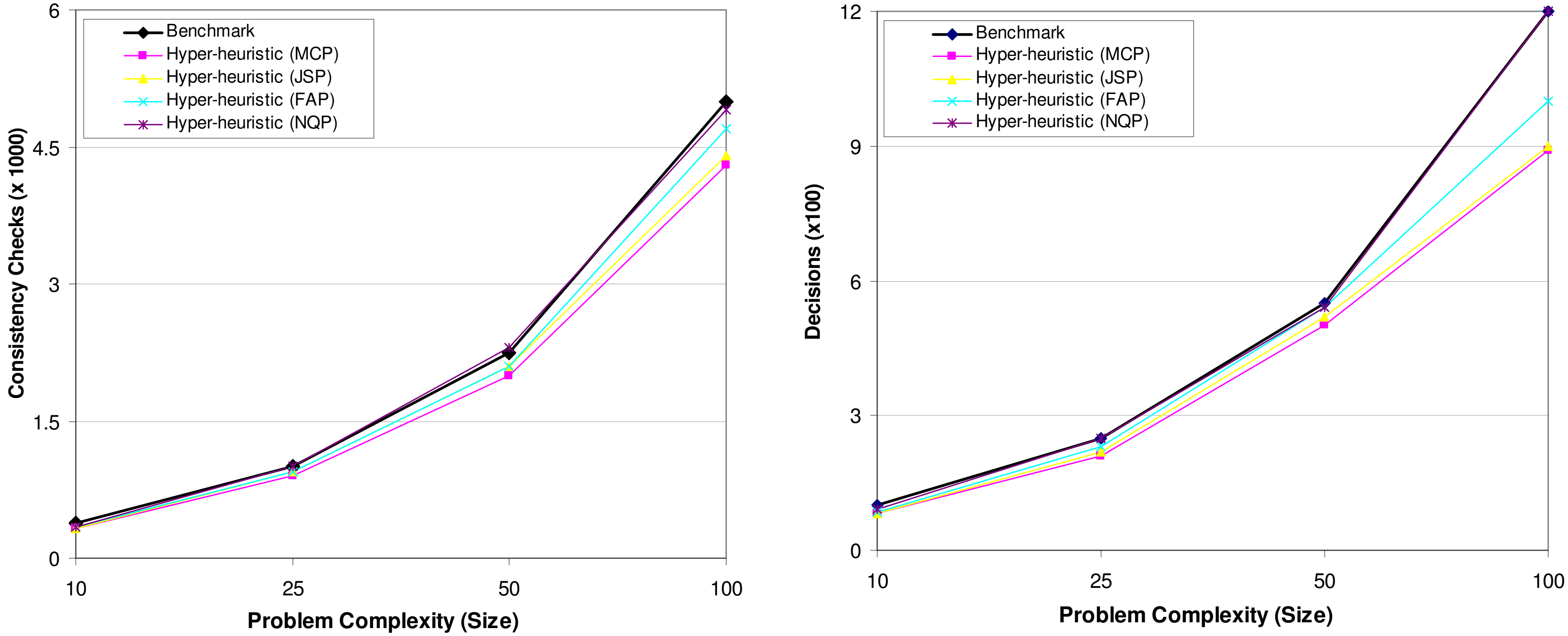


Learned non max/min texture based rules can scale to deliver superior problem-solving performance over traditional heuristics

Experiment 2:

Inter-Problem Solving and Learning

Map Coloring Problem

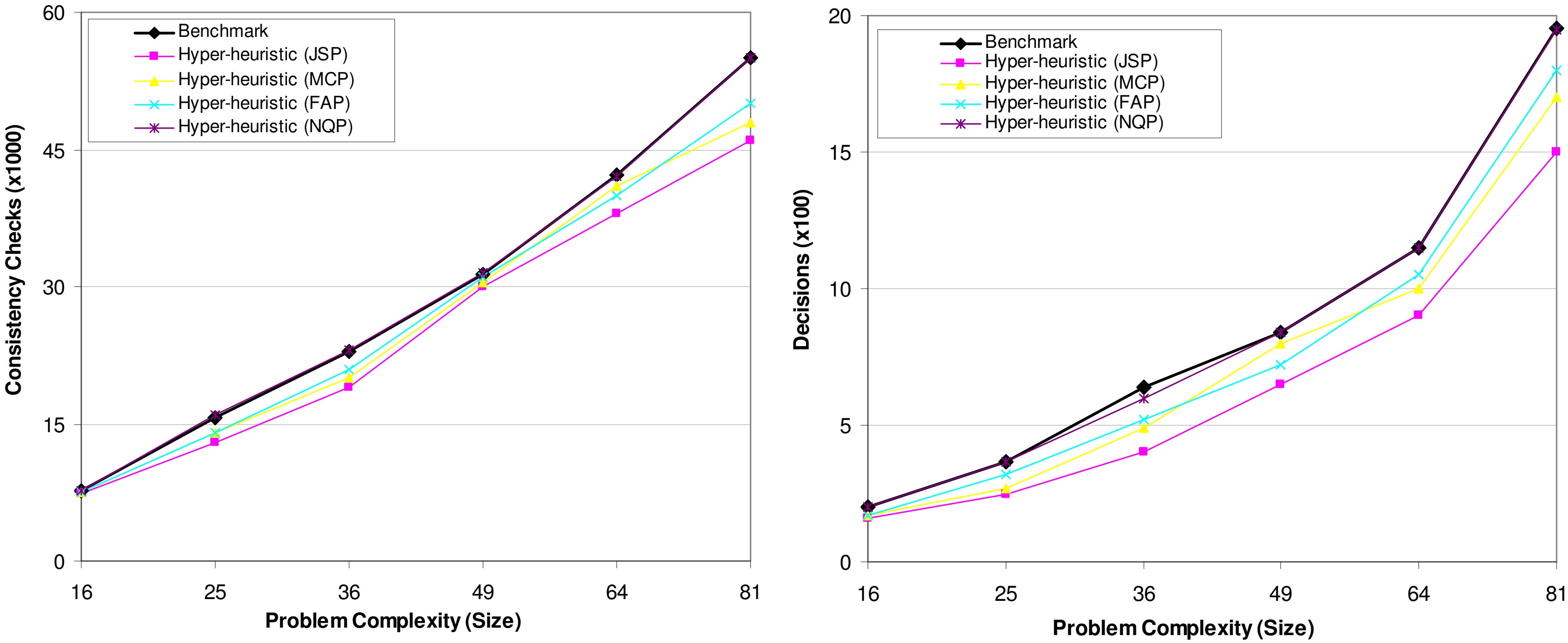


Learned rules while solving one problem type can be successfully be applied in solving different problem types and deliver superior problem-solving performance

Experiment 2:

Inter-Problem Solving and Learning

Job Shop Scheduling Problem



Learned rules while solving one problem type can be successfully be applied in solving different problem types that scale and deliver superior problem-solving performance

Design of CHS-Soar-RL

Issues with Subgoaling-Chunking (Soar 8.6.3):

- Chunk preferences are fixed (drawback)
- Chunking - subgoaling, allows us to “look-ahead” (benefit)

Issues with Reinforcement Learning (RL, Soar 9.0):

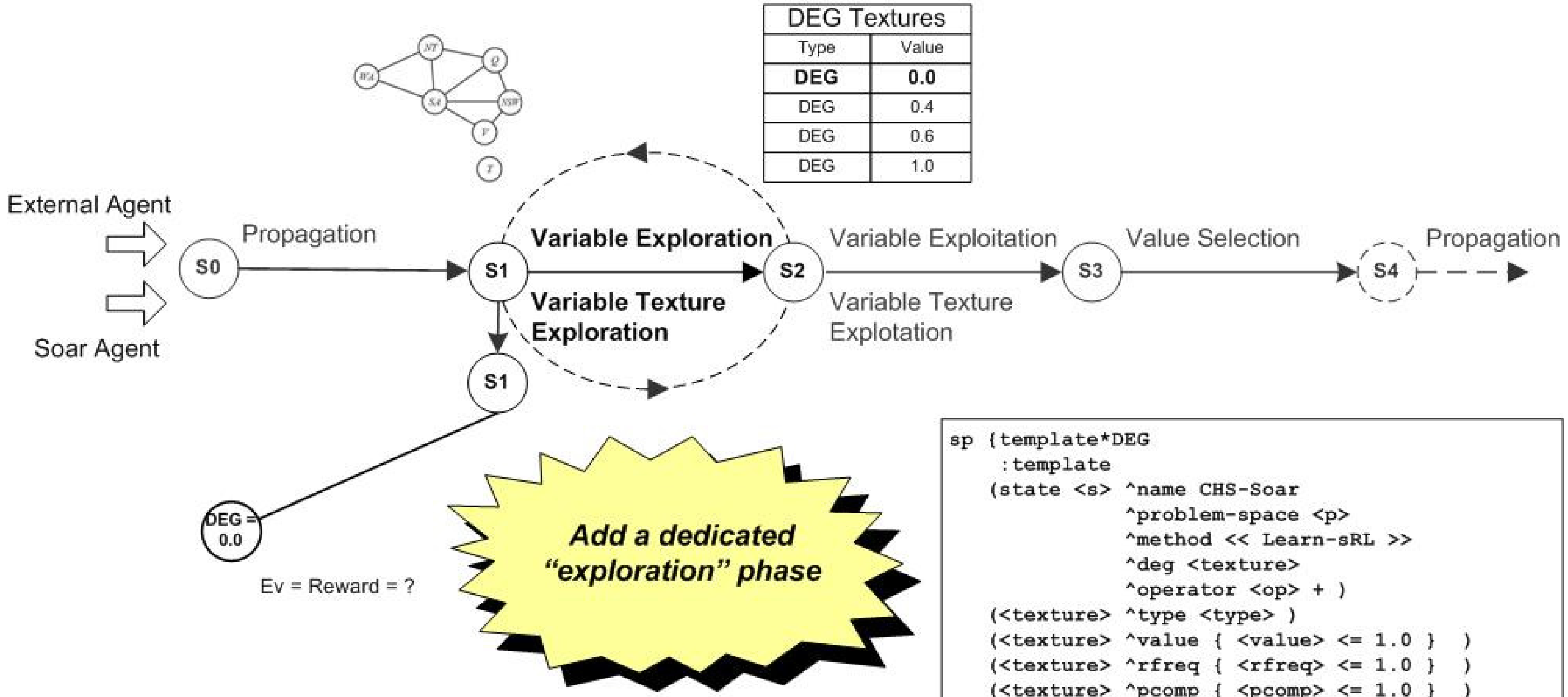
- RL rules can change numerical preferences (benefit)
- Does not allow us to subgoal in order to “look-ahead” (drawback)

Design goal of CHS-Soar-RL is to combine the benefits of both

- Allow us to “look-ahead”
- Use RL which allow num. preferences to change

Design of CHS-Soar-RL

How can we "look-ahead" with RL?



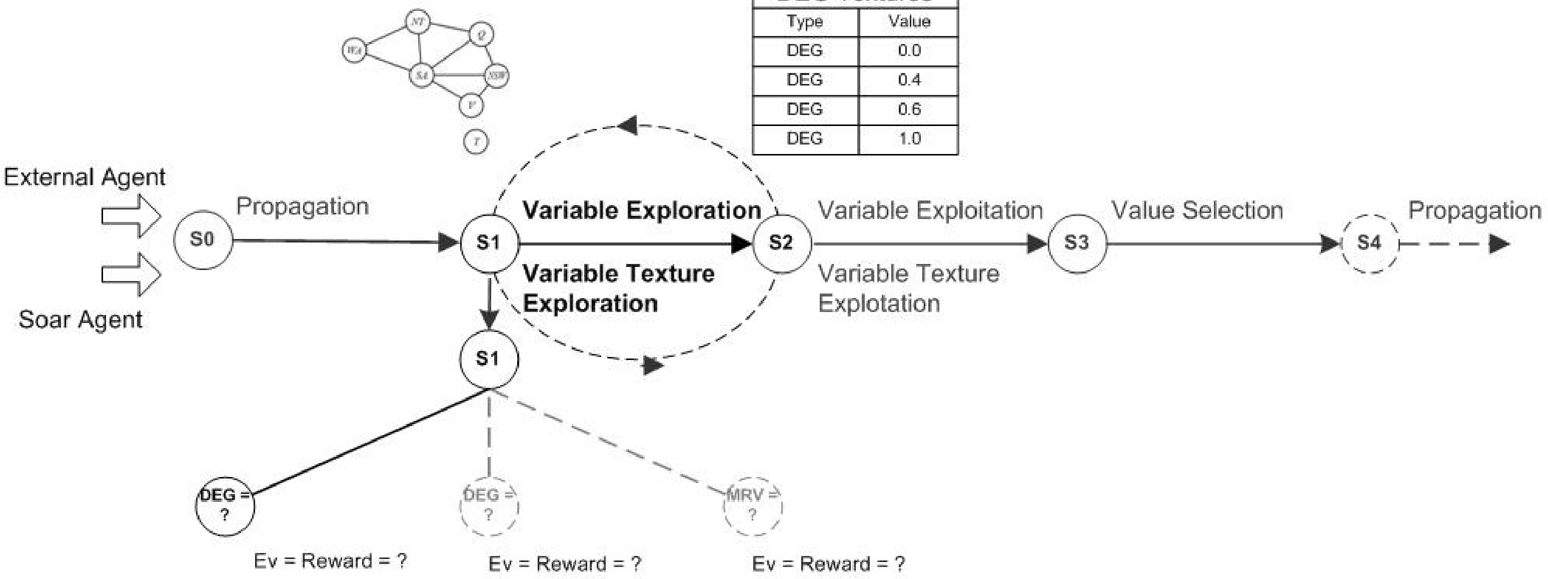
DEG Textures	
Type	Value
DEG	**0.0**
DEG	0.4
DEG	0.6
DEG	1.0

Add a dedicated "exploration" phase

```
sp {template*DEG
   :template
   (state <s> ^name CHS-Soar
              ^problem-space <p>
              ^method << Learn-sRL >>
              ^deg <texture>
              ^operator <op> + )
   (<texture> ^type <type> )
   (<texture> ^value { <value> <= 1.0 }  )
   (<texture> ^rfreq { <rfreq> <= 1.0 }  )
   (<texture> ^pcomp { <pcomp> <= 1.0 }  )
        (<op> ^name << SelectVariableTexture >>
              ^type <type>
              ^value <value>
              ^rfreq <rfreq>
              ^pcomp <pcomp> )
-->
(<s> ^operator <op> = 0 )
}
```

Design of CHS-Soar-RL

How can we “look-ahead” with RL?



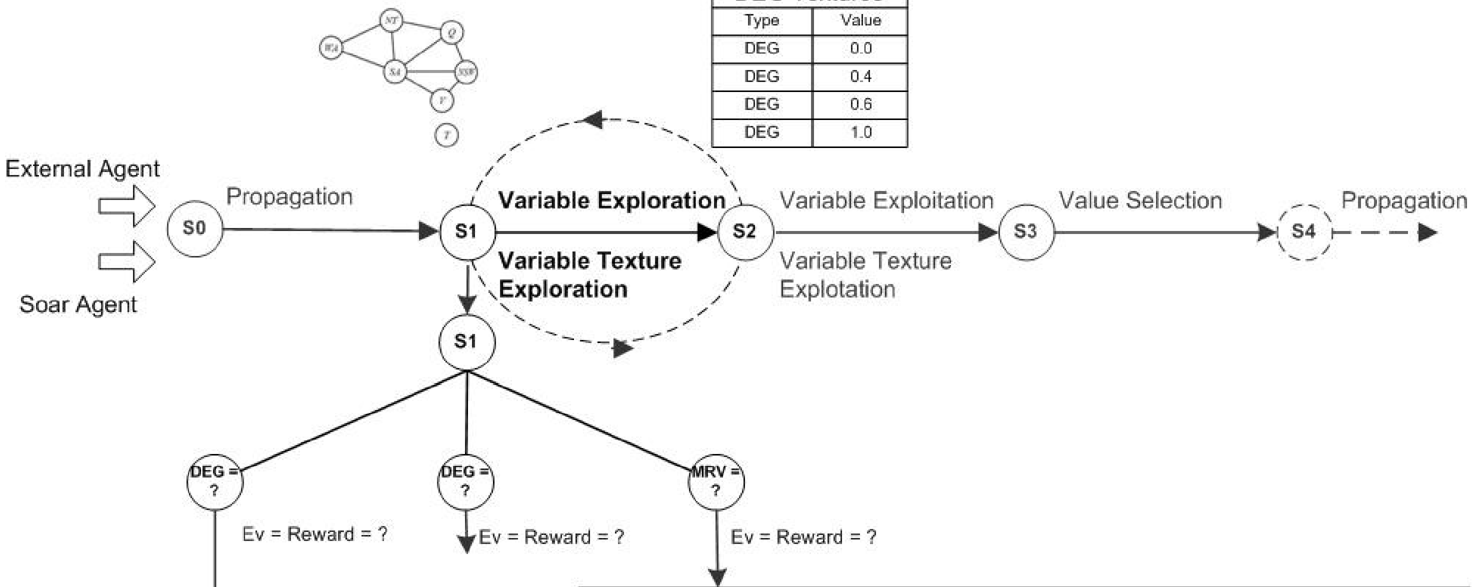
DEG Textures	
Type	Value
DEG	0.0
DEG	0.4
DEG	0.6
DEG	1.0

```
sp {propose*ExploreVariableTexture_DEG
   (state <s> ^name CHS-Soar
              ^problem-space <p>
              ^phase ExploreVariableTexture
              ^method << Learn-sRL >>
              ^deg <texture>)
   (<texture> ^type <type> )
   (<texture> ^value { <value> <= 1.0 }  )
   (<texture> ^rfreq { <rfreq> <= 1.0 }  )
   (<texture> ^pcomp { <pcomp> <= 1.0 }  )
   -->
         (cmd rl --set learning on)
         (<s> ^operator <op> +  )
        (<op> ^name SelectVariableTexture
              ^type <type>
              ^value <value>
              ^rfreq <rfreq>
              ^pcomp <pcomp> )
}
```

```
sp {apply*ExploreVariableTexture_DEG
    (state <s> ^name CHS-Soar
               ^problem-space <p>
               ^operator <op>
               ^phase { <phase> = ExploreVariableTexture }
               ^method << Learn-sRL >>
               ^reward-link <r>)
         (<op> ^name SelectVariableTexture
               ^type { <type> = deg }
               ^value <value>
               ^rfreq <rfreq>
               ^pcomp <pcomp> )
-->
          (<s> ^name CHS-Soar)
(<r> ^reward.value (float (exec explorevariabletexture <s> |:| <type> |:| <value> )))
          (<s> ^phase GetVariableTexture_DEG)
          (<s> ^phase <phase> -)
}
```

Design of CHS-Soar-RL

How can we "look-ahead" with RL?



DEG Textures	
Type	Value
DEG	0.0
DEG	0.4
DEG	0.6
DEG	1.0

```
sp {|rl*template*MRV*9|
    (state <s1> ^method |Learn-sRL| ^name |CHS-Soar| ^mrv <d1>
          ^operator <o1> + ^problem-space <p1>)
    (<d1> ^pcomp 0.14 ^rfreq 0.83 ^value 0.1 ^type mrv)
    (<o1> ^pcomp 0.14 ^rfreq 0.83 ^value 0.1 ^name |SelectVariableTexture|
          ^type mrv)
    -->
    (<s1> ^operator <o1> = 0.49980834)
}
```

```
sp {|rl*template*DEG*6|
    (state <s1> ^method |Learn-sRL| ^name |CHS-Soar| ^deg <d1>
          ^operator <o1> + ^problem-space <p1>)
    (<d1> ^pcomp 0.14 ^rfreq 0.33 ^value 0.6 ^type deg)
    (<o1> ^pcomp 0.14 ^rfreq 0.33 ^value 0.6 ^name |SelectVariableTexture|
          ^type deg)
    -->
    (<s1> ^operator <o1> = 0.61100514)
}
```

Nuggets

- Demonstrated integration of rule and constraint based reasoning
- Demonstrated CHS-Soar ability to reason about a small group of well known variable and value texture measures leading to improved solutions over traditional unary heuristics
- Demonstrated the ability to learn hyper-heuristics while solving one problem type can be successfully be applied in solving different problem types and deliver superior problem-solving performance over traditional combinations of unary heuristics
- Soar's rule based encoding dramatically expands the expressiveness of the hyper-heuristic by encoding the constituent textures of each heuristic-not simply the low level heuristics

Coal

- Limited only to CSP problems (and challenge of CSP representation)
- Effort to calculate textures can outweigh benefits
- Many “intermediate” texture measures evaluations provide no insight
- Textures are “proxies” for actual variables and value leads to random selections
- Scalability or results to more realistic CSP problems?
- Ability to export results for other CP solvers (i.e. ILOG) to use

Questions

***Introducing Constrained Heuristic Search
to the
Soar Cognitive Architecture***