A Cognitive Approach to solve Water Jugs Problem

Divya Saxena
Hindu College of Engineering
Sonipat, India

Naveen Kumar Malik
Hindu College of Engineering
Sonipat, India

V.R. Singh
PDM Educational Institutions
Bahadurgarh, India

ABSTRACT
Water-jug problem is a famous problem in the field of artificial intelligence, computer programming, recreational mathematics and psychology. Classical methods used to solve this problem are Depth first search, Breadth first search, Diophantine approach, etc. These methods are memory and time consuming. This paper implemented a cognitive approach with two new methods to solve water jug problem using the problem space computational model (PSCM) processing strategy of soar software. Result analyzed in term of time.

General Terms
Artificial intelligence, cognitive architecture, soar cognitive architecture

Keywords
Water jug problem, soar software, simple water jug agent

1. INTRODUCTION
Artificial Intelligence (AI) is the study to make computers intelligent as a human. The AI is interdisciplinary subject of fields like Electronics engineering, Computer engineering, Mathematics and philosophy etc [5]. Cognitive approaches are new area of research in AI. They provide infrastructure and framework for human like intelligent agents for various applications [11]. Some popular cognitive approaches available in literature are noted with name of architecture, founder, websites maintained with resources and basic features in table 1.The mentioned websites contain detailed information about the corresponding architecture.

1.1 Soar
Soar, cognitive approach, is described by John E. Laird, Allen Newell and Paul Rosenbloom at Carnegie Mellon University, Pitsburgh, Pennysylavania. Soar is a theory as about what cognition is, as well as, a computer programming software that implemented in Artificial Intelligent machines to achieve different aspects of human behavior [12].

Soar is different from other approaches in a way that it takes dynamic combination of wide variety of knowledge for solving problems. Knowledge can be programmed into the system or was learned through experience called chunking. Some applications of Soar are in reasoning tasks, medical diagnosis, natural language processing, robotic control, simulating pilots for military training, and controlling the non-player characters in mobile and computer games, etc [4].

Table 1. Popular Cognitive Approaches

Cognitive approaches	Founder and year	Defining links	Features
ACT-R	Anderson, 1976	http://act-r.psy.cmu.edu/	Human semantic memory
4CAPS	Thibadeau et al., 1982	http://ccbi.cmu.edu/4CAPS/	Combination of symbolic and activation based processing
SOAR	Laird, Rosenbloom & Newell, 1983	http://soar.eecs.umich.edu/	Multi-method problem solving, production systems, and problem spaces
Prodigy	Minton & Carbonell, 1986	http://cogarch.org/index.php/Prodigy/Properties	Means–end analysis, planning
MAX	Kuokka, 1991	http://cogarch.org/index.php/MAX/Architecture	Meta-level reasoning for planning and learning
ICARAUS	Langley & Shapiro, 2003	http://cll.stanford.edu/research/ongoing/icarus/	Concept learning and planning
EPIC	Kieras & Meyer, 1997	http://web.eecs.umich.edu/~kieras/epic.html	Models of human perception, action, and reasoning
4D/RCS	Albus	http://cogarch.org/index.php/4D-RCS	Hierarchical sensory processing, hierarchical real-time execution
Polyscheme	Cassimatis, 2004	http://dspace.mit.edu/handle/1721.1/8325	Integration of multiple methods of representation and reasoning

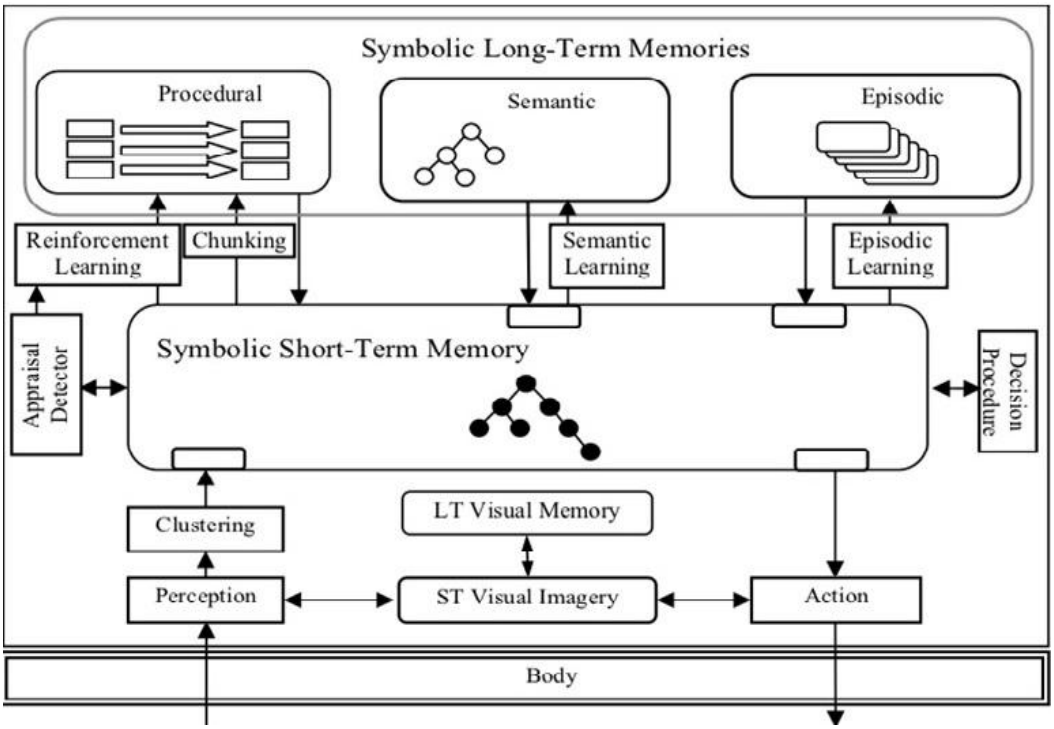

Fig 1: Soar 9 architecture [10]

1.1.1 Soar Architecture

The perception module receives the perceptions from the environment and transfers it to the global short term memory. Working memory controls the retrieval of knowledge from the long term memory and also responsible for initiating actions. The three long term memories namely procedural, semantic, episodic are independent from each other and have their separate learning mechanisms. Procedural long term memory contains the knowledge about how to do things (processing), the semantic memory stores the general facts and episodic memory stores the knowledge about snapshot of working memory. The procedural productions can change the status of the working memory. Short term working memory is connected with other memories and processes so change in it can initiate retrieval from semantic or episodic memory or can initiate action in the environment. Chunking means to learn new production rules whereas reinforcement learning means tuning the action of rules by changing numeric preferences in operator evaluation cycle. Figure 1 shows latest available soar cognitive architecture.

1.1.2 Soar Operation Cycle

The operation cycle of soar is shown in figure 2. Soar takes input from the environment in the form of visual perceptions or from sensors like pressure sensors, weight sensors (may be senses the weight of water in jug), and then elaborate (creates state) the current situation. Based on states created, proposes the operators or evaluate the operators (changes the operator preference) by means of rules matching for the current state. All these are done by the productions rules saved in long term memory of soar. Then the decisions module selects the operator among different proposed operators.

The most eligible operator can be chosen from the proposed operators by defining rules. Then the operator is applied (means action are taken) also by rules matching which causes changes in the environment state. The action can be like starting a motor of robot leg so that it can move to a desired place, or moving robots hand to fill a jug.

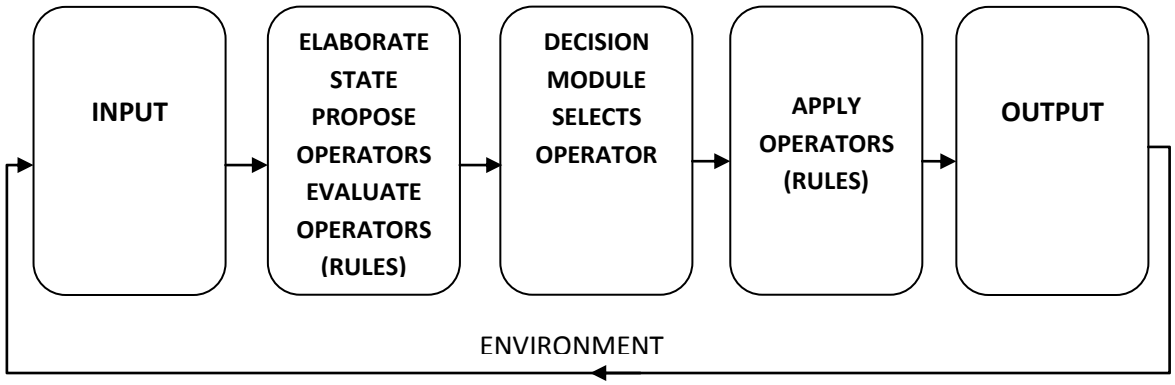

Fig 2: Operation Cycle of soar [4]

1.1.3 Water jug problem statement

Water-jug problem is most general planning problem of Artificial Intelligence [8]. The problem can be stated as below:

Sitting beside a river, two empty jugs of volume say x liters and y liters are provided. The aim is to calculate the number of moves to complete the task of getting z liters of water in jug x or jug y or as in combined quantity of both.

As a human, it is very easy to solve this problem by just thinking for few seconds. But with the machines there should be a designed program that calculates all the possible actions that moves an agent from initial state to goal state and then chooses the best optimal action among them that will achieve goal faster.

This paper implemented water jug problem with soar software which is used to design intelligent agents [1]. These intelligent agents perform the human able task based on their computational processing. Thus, processing is done by the soar program that is saved in machine or robot.

2. LITERATURE REVIEW

There are various methods applied to solve water jug problem. The techniques to proceed from start to goal can be classified into two groups namely informed search (heuristic) and uninformed search (blind) also shown in figure 3.

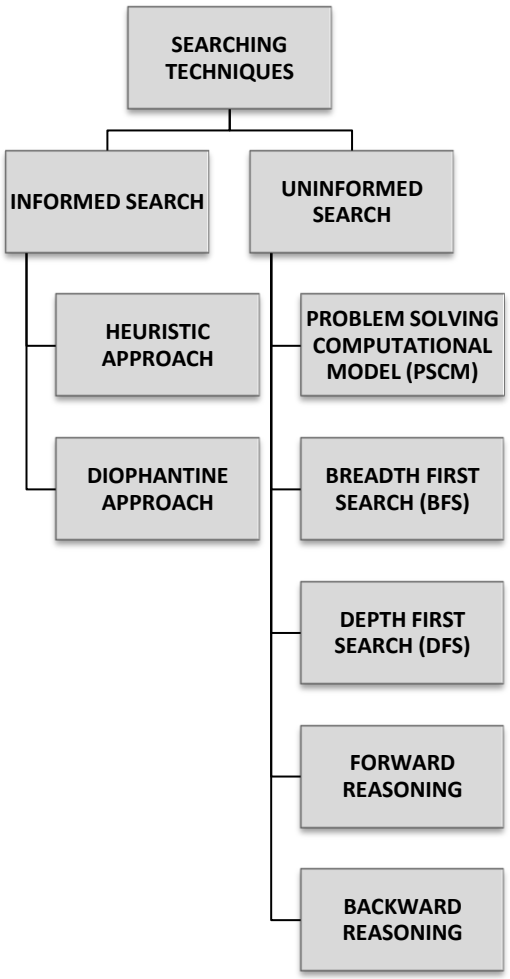

Fig 3: Classification of strategies used to solve the Water-jug Problem

2.1 Informed Search

2.1.1 Heuristic

The Heuristic technique improves the efficiency of a search process, possibly by sacrificing claims of completeness or optimality [7].

2.1.2 Diophantine Approach

The Diophantine approach is a manually or computationally solving arithmetic method by modeling problem as a Diophantine equation, namely mx+ny=d is solvable only if gcd(m,n) divides d. Then comparing two algorithms designed to fill jug m first or n first to get optimal result [6].

2.2 Uninformed Search

2.2.1 Forward Reasoning

The billiards/forward reasoning approach as understood by its name is a throw starting from initial state to goal [7]. Forward chaining approach is a popular implementation strategy for expert systems, business and production rule systems [13].

2.2.2 Backward Reasoning

The working backwards approach, starts from goal state and then assuming the previous states. This method uses less memory as compared to others [7]. Backward reasoning is implemented in logic programming systems usually employ a depth-first search strategy [14].

2.2.3 Breadth Fist Search

The Breadth First Search (BFS) expands all the nodes of one level first. The time and space is b^d for BFS where b and d are branching factor and solution depth respectively. Experimentally BFS is complete and optimal [9].

2.2.4 Depth First Search

The Depth First Search approach expands one of the nodes at the deepest level. The time and space are b^m and bm where b is the branching factor and m is the maximum depth. DFS is not complete and optimal [9].

2.2.5 Problem Solving Computational Model

In Problem Solving Computational Model (PSCM), a problem is viewed as a space of all the states and then, selecting the best connection between the starting and ending state [4]. The PSCM theory of soar is based on goals (aim of the problem), problem spaces (space of states), states (agent is in a state) and operator (alternative action). The agent is in a state and its alternative actions are decided by operators. Once the operator is applied, a new state is created. Soar chooses the optimal solution for a problem by applying the most preferable operator among the candidate operators available.

The drawbacks of previous methods are time consumption and memory requirement as compared to soar's faster approach PSCM. This paper introduces two methods to fasten the PSCM process of soar's water–jug problem solving. First method proposed creates the preference rules for fill and pour operates and the second, blocks some states of problem-space making it more goal–oriented.

3. THE PROBLEM

The problem work created by soar can be downloaded from http://soar.eecs.umich.edu/articles/downloads/agents/153-water-jug-simple link. The soar's specific-water jug problem can be stated as below:

You are given two empty jugs. One holds five liters of water and the other holds three liters. There is a well that has

unlimited water that you can use to completely fill the jugs. You can also empty a jug or pour water from one jug to another. There are no marks for intermediate levels on the jugs. The goal is to fill the three-liter jug with one liter of water [3].

The human brain know that the optimum solution is in five steps, by just first filling 3-liter jug, then pouring this amount into 5-litre jug, and then again filling 3-liter jug pouring it into 5-litre jug. The desired solution can be obtained.

Soar will solve it in number of steps. The problem space designed is shown in figure 4. Soar can take any path at random to reach from [0,0] to […,1] state.

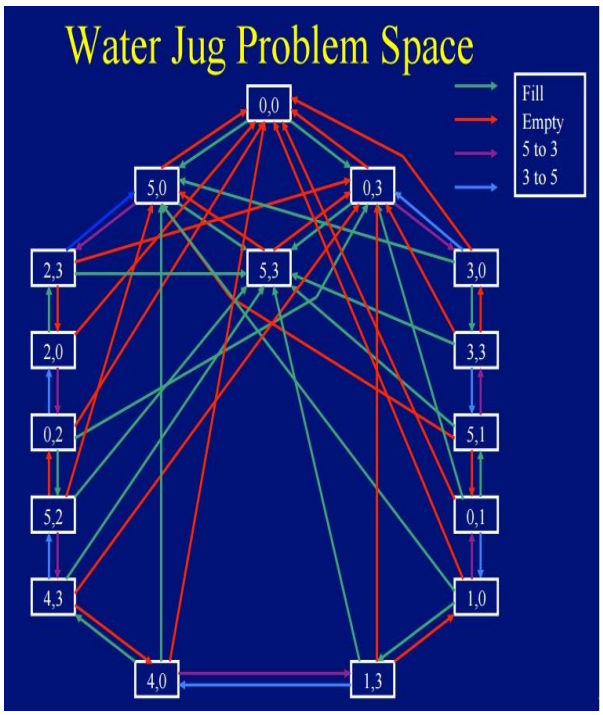

Fig 4: PSCM of Soar's simple water-jug-agent [3]

3.1 Operators

The operators of water – jug problem are filling a jug from the well, empty a jug into the well and pour from a jug to a jug.

3.2 Soar syntax

The table 2 shows some terms related to water jug problem in soar syntax. The soar's syntax is an example of AI language.

Table 2. Name and State representation in Soar syntax

Terms	In English words			In Soar syntax
Name	water jug			(<s>^name water-jug)
States	jug <j1>	Volume	5	(<s>^jug <j1>) (<s>^jug <j2>) (<j1>^volume 5) (<j1>^content 0) (<j1>^ empty 5) (<j2>^volume 3) (<j2>^content 0) (<j2>^empty 3)
		Content	0	
		Empty	5	
	jug <j2>	Volume	3	
		Content	0	
		Empty	3	

3.3 Default short term memory

The default short term memory structure is given in figure 5. This is the starting graph structure of working memory. Computational processing is done by adding and removing of branches over this initial graph.

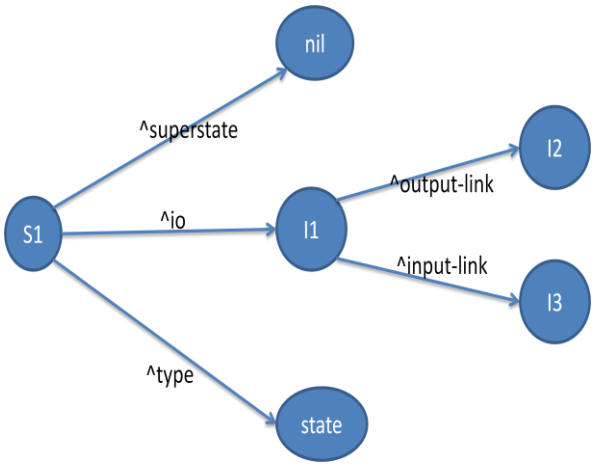

Fig 5: Default Soar's STM structure [3]

3.4 Initial state structure

The initial state structure of water – jug problem in short term memory is shown in figure 6. The new attributes are created after initialization of water-jug problem.

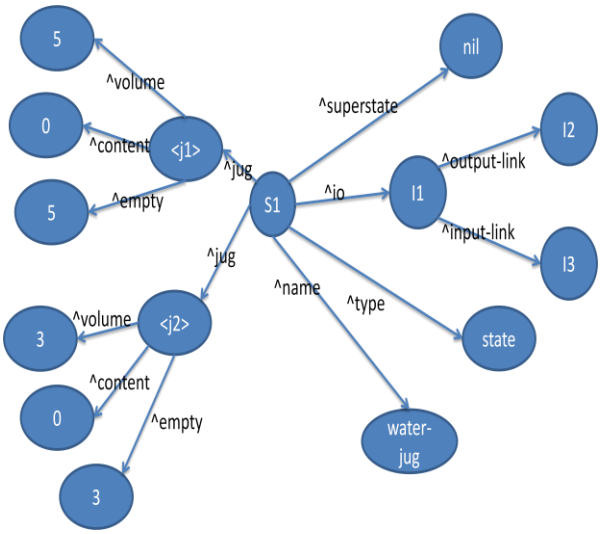

Fig 6: Initial STM structure of Soar's water-jug agent

3.5 Rules for Soar's water jug agent

The Table 3 shows rules saved in Soar's long term memory for simple water jug agent.

Table 3. Production rules in Soar's LTM for simple water-jug agent [2]

Rule no.	Simple English	Soar syntax
P1	If no task is selected, *then propose the initialize-water-jug operator.*	sp {propose*initialize-water-jug-new (state <s> ^type state) -(<s> ^name) --> (<s> ^operator <o> +) (<o> ^name initialize-water-jug-new) }
P2	If the initialize-water-jug operator is selected, *then*	sp {apply*initialize-water-jug-new (state <s>

	create an empty 5 gallon and empty 3 gallon jug	^operator.name initialize-water-jug-new) -->(<s> ^name water-jug-new ^jug <j1> ^jug <j2>) (<j1> ^volume 5 ^content 0) (<j2> ^volume 3 ^content 0) }
P3	If a jug has volume *v* and contents c, *then it has empty v – c.*	sp {water-jug-new*elaborate*state (state <s> ^name water-jug-new ^jug <j>) (<j> ^volume <v> ^content <c>) -->(<j> ^empty (- <v> <c>)) }
P4	If there is a jug that is not full, *then propose the fill operator.*	sp {water-jug-new*propose*fill (state <s> ^name water-jug-new ^jug <j>) (<j> ^empty > 0) -->(<s> ^operator <op> + =) (<op> ^name fill ^fill-jug <j>) }
P5	If the fill operator is selected for a jug, *then change the contents of that jug to its volume.*	sp {apply*fill (state <s> ^name water-jug-new ^operator <op> ^jug <j>) (<op> ^name fill ^fill-jug <j>) (<j> ^content <c> ^volume <v>) -->(<j> ^content <v>) (<j> ^content <c> -) }
P6	If there is a jug that is not empty, *then propose the empty operator.*	sp {water-jug-new*propose*empty-jug (state <s> ^name water-jug-new ^jug <j>) (<j> ^content > 0) -->(<s> ^operator <op> + =) (<op> ^name empty ^empty-jug <j>) }
P7	If the empty operator is selected for a jug, *then change the contents of that jug to 0.*	sp {apply*empty (state <s> ^name water-jug-new ^jug <j> ^operator <op>) (<op> ^name empty ^empty-jug <j>) (<j> ^volume <v> ^content <c>) --> (<j> ^content 0) (<j> ^content <c> -) }
P8	If there are two jugs and first jug is not full and second jug is not empty, *then propose pouring water from the second jug into the first jug.*	sp {water-jug-new*propose*pour-jug (state <s> ^name water-jug-new ^jug <i> ^jug {<j> <> <i>}) (<i> ^content > 0) (<j> ^empty > 0) -->(<s> ^operator <op> + =) (<op> ^name pour ^empty-jug <i> ^fill-jug <j>) }
P9	If pour operator is selected for two jugs and the contents of the jug being emptied <= the empty amount of the jug being filled, *then set the contents of the jug being emptied to 0 and set the contents of the jug being filled to the sum of the two jugs.*	sp {apply*pour*empty-empty (state <s> ^name water-jug-new ^operator <op>) (<op> ^name pour ^empty-jug <i> ^fill-jug <j>) (<j> ^volume <jv> ^empty <je> ^content <jc>) (<i> ^volume <iv> ^content { <ic> <= <je> }) -->(<i> ^content 0 <ic> -) (<j> ^content (+ <ic> <jc>) <jc> -) }
P10	If pour operator is selected for two jugs and the contents of the jug being emptied > the empty amount of the jug being filled, *then set the contents of the jug being emptied to its contents minus the empty of the jug being filled and set the contents of the jug filled to its volume.*	sp {apply*pour*not-empty-empty (state <s> ^name water-jug-new ^operator <op>) (<op> ^name pour ^empty-jug <i> ^fill-jug <j>) (<j> ^volume <jv> ^empty <je> ^content <jc>) (<i> ^volume <iv> ^content { <ic> > <je> }) -->(<i> ^content (- <ic> <je>) <ic> -) (<j> ^content <jv> <jc> -) }
P11	If five volume jug's content is c1 and three volume jug's content is c2, *then print 5 has c1 and 3 has c2.*	sp {monitor-state (state <s> ^name water-jug-new ^jug <j1> <j2>) (<j1> ^volume 5 ^content <c1>) (<j2> ^volume 3 ^content <c2>) -->(write (crlf) |5 has | <c1> | : 3 has | <c2>) }

P12	If fill operator is selected to fill jug of volume v, *then print fill v.*	sp {monitor-fill (state <s> ^name water-jug-new ^operator <o>) (<o> ^name fill ^fill-jug <j>) (<j> ^volume <v>) --> (write (crlf) |fill (|<v>|)|) }
P13	If empty operator is selected to empty jug of volume v, *then print empty v.*	sp {monitor-empty (state <s> ^name water-jug-new ^operator <o>) (<o> ^name empty ^empty-jug <j>) (<j> ^volume <v>) --> (write (crlf) |empty (|<v>|)|) }
P14	If pour operator is selected to fill j2 from j1, *then print pour j1's volume v1 : content c1 and j2's volume v2 : content c1.*	sp {monitor-pour (state <s> ^name water-jug-new ^operator <o>) (<o> ^name pour ^fill-jug <j2> ^empty-jug <j1>) (<j1> ^volume <v1> ^content <c1>) (<j2> ^volume <v2> ^content <c2>) --> (write (crlf) |pour(| <v1> |:| <c1> |,| <v2> |:| <c2> |)|) }
P15	If there is a jug with volume five and contents three, *then write that the desired goal achieved and halt.*	sp {detect*goal (state <s> ^name water-jug-new ^jug <j>) (<j> ^volume 5 ^content 3) --> (write (crlf) | goal achieved|) (halt) }

4. PROGRAM WORKING

The solution of water jug problem taken in the following description is the shortest path taken from initial state to desired state for simplicity of explanation.

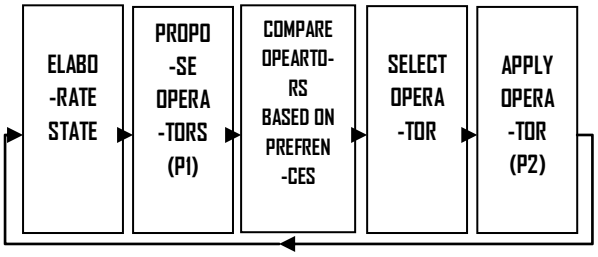

Fig 7: First phase of operation cycle

In the first cycle of operation displayed in figure 7, soar first elaborates the current state and then proposes operators based on situation, so rule P1 fires. The rule is if no tasks (means doing nothing) then start water jug problem. In this paper, water – jug problem is visualized as an internal problem solving in soar. Consider a robot provided with two empty jugs in his hand then the state initialization will be done through input from sensors (may be weight sensor or visual perceptions). As only one operator is candidate for selection, it is selected and applied.

The application rule matches rule P2 so it fires. The Rule P2 creates the initial state of two empty jugs by adding attributes to the default short term memory as shown in figure 5 and figure 6. The created status in soar syntax is written as:

<j1> ^volume 5 ^content 0
<j2> ^volume 3 ^content 0

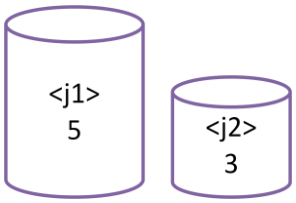

Fig 8: First phase creates two empty jugs <j1> and <j2>

The empty attribute is not originated till now, because a separate rule is needed for calculation of empty space as its value changes in each phase. Just to show diagrammatically, two jugs are created like in figure 8.

The first rule P1 is retracted as there is a task now and states created (two empty jugs) persist. Now, in the next cycle in figure 9, again it will elaborate the current situation and fires rule P3 as the conditions of P3 now matches the current short term memory structure.

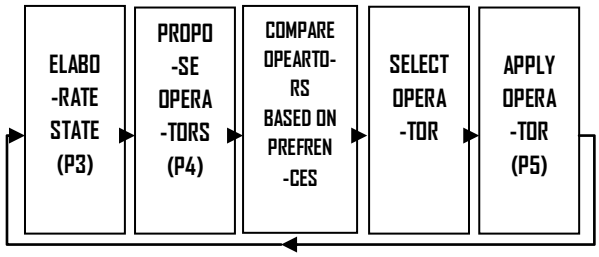

Fig 9: Second phase of operation cycle

It creates an additional attribute on both jugs named empty which calculates the empty amount left in the jug. Now according to current empty situation, the P4 rule fires, proposing operators to fill jugs may be <j1> or <j2>. The comparison is done because of the preferences associated with operators. As given random preference, so it can choose any one, or say <j2>, and selects operator to fill <j2>. Now the rule P5 matches and changes the status of short term memory by changing content value of <j2> to 3.

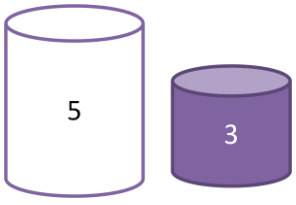

Fig 10: Second phase fill three-volume jug <j2>

It is essential to remove the attribute so there is an action of removing the last value in rule. The current status of STM in soar language can be shown as below and with diagram in figure 10.

<j1> ^volume 5 ^content 0 ^empty 5
<j2>^volume 3 ^content 3 ^empty 3

In the third cycle of figure 11, it will again elaborate the state and update the empty attribute value. Then based on current condition matching, it will fire the rules P4, P6, P8 and propose operators fill, empty and pour the jug. Again, by comparing randomly it chooses any one, say pour rule P8 and applies the rule P9. The application rules for pour operator has two conditions, one is when it will empty the source jug fully and the second is when it will not empty the source jug means some amount of water left. Soar choose P9 that will empty the jug <j2> fully and changes the content of <j1> and <j2> to 3 and 0 respectively. The current status of short term memory and jugs are shown below in equation and in figure 12.

<j1> ^volume 5 ^content 3 ^empty 5
<j2> ^volume 3 ^content 0 ^empty 0

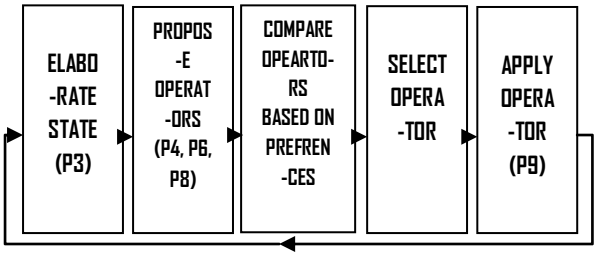

Fig 11: Third phase of operation cycle

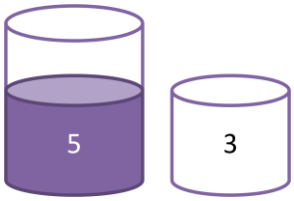

Fig 12: Third phase pour water from <j2> to <j1>

The fourth operating cycle as in figure 13 will continue from elaboration phase updating the empty attribute of <j1> and <j2> equal to 2 and 3. The short term memory state proposes three operators namely fill, empty and pour (P4, P6, and P8) and then randomly chooses fill and hence applies P5 to fill jug <j2>.

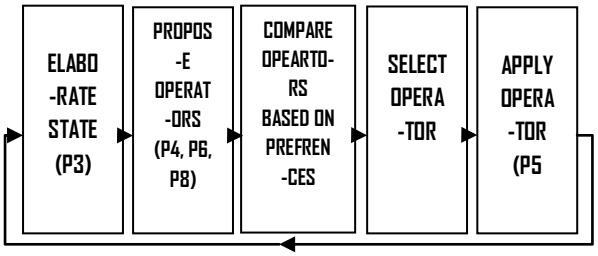

Fig 13: Fourth phase of operation cycle

The content value of <j2> goes to 3 again. The graph of STM and diagrams of jugs are shown below and in figure 14.

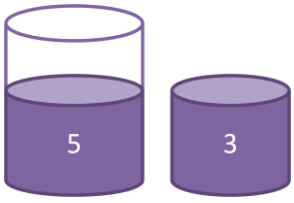

Fig 14: Fourth phase again fills jug <j2>

<j1> ^volume 5 ^content 3 ^empty 2
<j2> ^volume 3 ^content 3 ^empty 3

In the fifth phase of figure 15, after following the same steps of operation (first elaborating empty attribute's value to 2 and 0, then proposing operators fill, empty and pour). Let soar randomly chooses to pour jug <j2> to jug <j1> by application of rule P10. The change in values is written in soar syntax below and by virtual diagram jugs in figure 16.

<j1> ^volume 5 ^content 5 ^empty 2
<j2> ^volume 3 ^content 1 ^empty 3

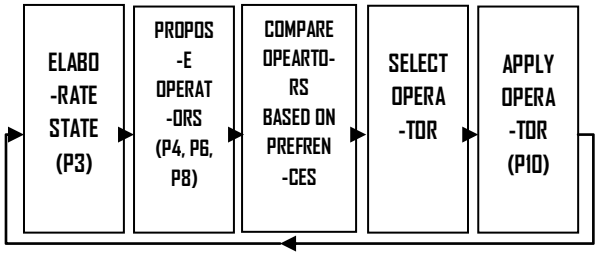

Fig 15: Fifth phase of operation cycle

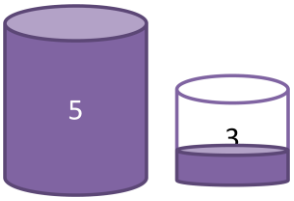

Fig 16: Fifth phase pour from jug <j2> to <j1>

In the next elaboration phase, the rule P15 fires as the condition of the rule now matches the current status of short term memory which is the desired result also and hence it will print that goal is achieved. If this program is used with an intelligent agent or a robot, a speaker can be used saying the task completed.

5. PROPOSED METHOD I

Soar`s idea for a search procedure is the discovery of a unknown path in the problem space that links the initial state with the goal state.

Table 4. New rules of Method I added in referenced method

Rule no.	Simple English	Soar syntax
P16	If operators proposed are empty and pour *Then prefer pour over empty.*	sp {prefer*pour (state <s> ^operator <o1> + ^operator <o2> +) (<o1> ^name empty) (<o2> ^name pour) --> (<s> ^operator <o2> > <o1>) }
P17	If operators proposed are empty and fill *Then prefer fill over empty.*	sp {prefer*fill (state <s> ^operator <o1> + ^operator <o2> +) (<o1> ^name empty) (<o2> ^name fill) --> (<s> ^operator <o2> > <o1>) }

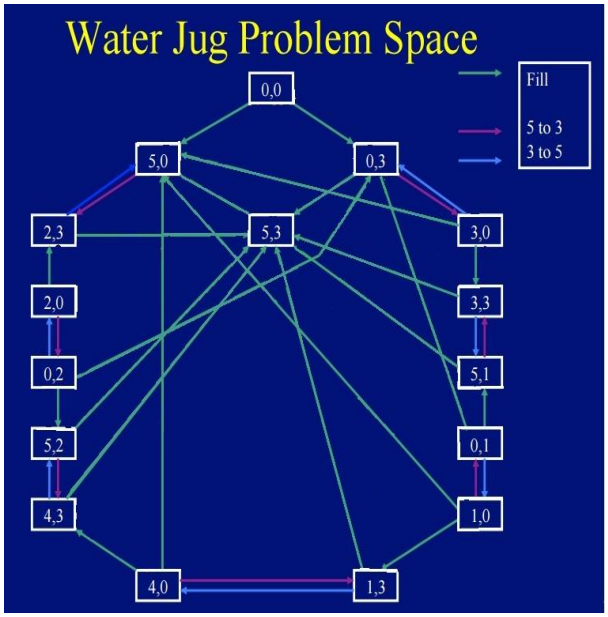

Fig 17: PSCM of Method I

Two new rules P16 and P17, shown in table 4, are added to the referenced method's fifteen rules to give the empty operation as worst preference compared to fill and pour. The new problem space design can be seen in figure 17.If thinking logically, the empty operator is extending the steps only without requirement because optimal solution can be possible in five steps without emptying the jug.

In the new soar's operation cycle, if proposed operators are filling and empty it will choose fill. If operators proposed are pouring and empty it will choose pour and if all three of them proposed than it will choose between fill and pour randomly.

6. PROPOSED METHOD II

The second method proposed has been developed by focusing on the idea of technique used. In this approach reduction of the problem space of water–jug simple agent is done by restricting it from pouring filled five–gallon jug to three-gallon jug and then emptying the filled three-gallon jug as their actions only lengthening the steps. This condition is applicable in any water-jug problem but only when the focused jug (task to get some amount of water in smaller jug) is the smaller one.

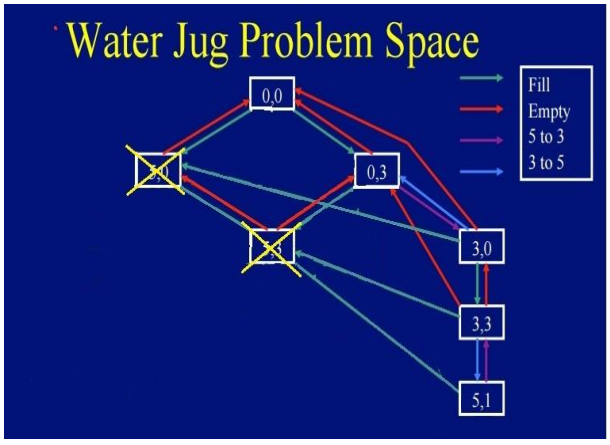

Fig 18: PSCM of Method II

The rule added with other 15 rules of referenced method, is P18 shown in table 5 with minor changes of pouring operators in bolds. Whenever the problem space's states reaches {5;(0,1,2,3)} state it will reject the pour operator. The problem space can be seen in figure 18.

Table 5. New rules of Method II added in referenced method

Rule no.	Simple English	Soar syntax
P8	If there are two jugs and first jug is not full and second jug is not empty, *then propose pouring water from the second jug into the first jug.*	sp {water-jug-new*propose*pour-jug (state <s> ^name water-jug-new ^jug <i> ^jug {<j> <> <i>}) (<i> ^content > 0) (<j> ^empty > 0) --> (<s> ^operator <op> + =) (<op> ^name pour ^empty-jug <i> ^fill-jug <j>) **(<op> ^done <d>)** }
P9	If pour operator is selected for two jugs and the contents of the jug being emptied <= the empty amount of the jug being filled, *then set the contents of the jug being emptied to 0 and set the contents of the jg being filled to the sum of the two jugs.*	sp {apply*pour*empty-empty (state <s> ^name water-jug-new ^operator <op>) (<op> ^name pour ^empty-jug <i> ^fill-jug <j>) **(<op> ^done <> true)** (<j> ^volume <jv> ^empty <je> ^content <jc>) (<i> ^volume <iv> ^content { <ic> <= <je> }) --> (<i> ^content 0 <ic> -) (<j> ^content (+ <ic> <jc>) <jc> -) }
P10	If pour operator is selected for two jugs and the contents of the jug being emptied > the empty amount of the jug being filled, *then set the contents of the jug being emptied to its contents minus the empty of the jug being filled and set the contents of the jug filled to its volume.*	sp {apply*pour*not-empty-empty (state <s> ^name water-jug-new ^operator <op>) (<op> ^name pour ^empty-jug <i> ^fill-jug <j>) **(<op> ^done <> true)** (<j> ^volume <jv> ^empty <je> ^content <jc>) (<i> ^volume <iv> ^content { <ic> > <je> }) --> (<i> ^content (- <ic> <je>) <ic> -) (<j> ^content <jv> <jc> -) }
P18	If the five volume jug with content more than one is	sp {reject*pour (state <s> ^name water-jug-new ^operator <o>) (<o> ^name pour

	being poured, *then reject the pour operator*.	^empty-jug <i> ^fill-jug <j>) (<i> ^volume 5 ^content {<c> > 1} ^empty <e>) -->(<s> ^operator <o> + -) (<i> ^content 0 <c> -) (<i> ^empty 5 <e> -) (<s> ^done true) }

7. RESULT AND DISCUSSIONS

From the result table 6, it is observe that the soar's water jug simple agent program's efficiency has been increased by 19.717% by Method I and 72.32% by Method II. It should also be noted that the results shown in table 5 are not fixed. They are generated at random but the reduction of efficiency is sure. The implemented work for 11th run of experiment in soar debugger window is shown in figures 19, 20 and 21.

Table 6. Comparison of number of steps obtained by Referenced method and Proposed Method I and Method II

Experiment	Steps in Referenced Method	Steps in Proposed Method I	Steps in Proposed Method II
1st RUN	345	39	50
2nd RUN	51	385	35
3rd RUN	156	254	66
4th RUN	200	72	5
5th RUN	57	46	11
6th RUN	569	39	134
7th RUN	44	636	32
8th RUN	97	142	21
9th RUN	24	67	9
10th RUN	69	175	86
11th RUN	227	184	102
12th RUN	351	80	15
13th RUN	22	5	44
14th RUN	379	34	72
15th RUN	223	111	102
16th RUN	321	104	12
17th RUN	122	158	42
18th RUN	487	56	30
19th RUN	47	202	75
20th RUN	104	138	88
Total	3895	3127	1078
Average	194.75	156.35	53.9
Efficiency		19.717% Increased	72.32% Increased

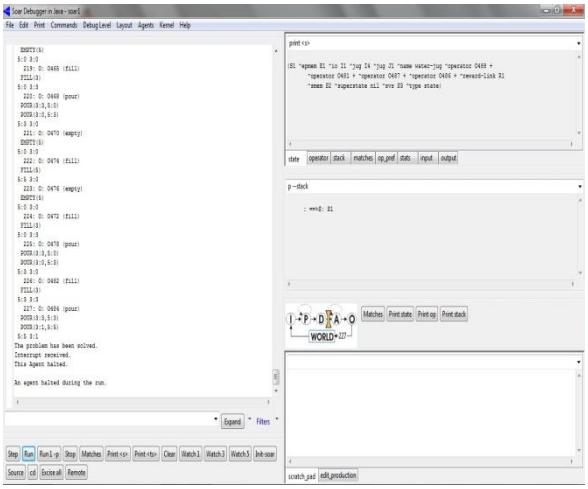

Fig 19: Soar debugger window for 11th run of Referenced Method

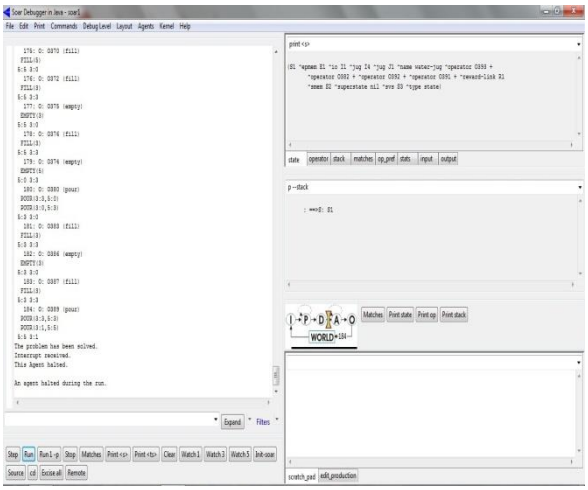

Fig 20: Soar debugger window for 11th run of Method I

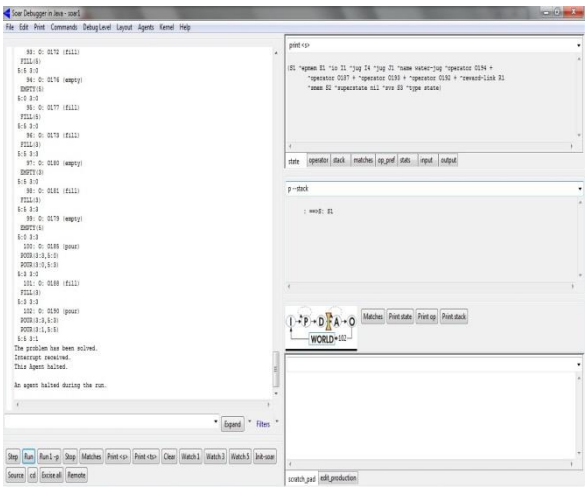

Fig 21: Soar debugger window for 11th run of Method II

8. CONCLUSION AND FUTURE SCOPE

Cognitive architectures provide new approach to develop intelligent agents for various real time applications. The simple water jug problem is implemented in soar version 9.5.0 software. It has advantage of reduction in average time to solve the water jug problem as compared to other available methods for same. Water jug soar agent with seventeen rules in method I and sixteen rules in method II have increased the efficiency in the time domain by 19.717% and 72.32% respectively.

Further soar software features, namely chunking, reinforcement learning can be used to solve water jug problem for better results.

9. REFERENCES

[1] SOAR: A Cognitive Architecture http://soar.eecs.umich.edu/articles/downloads/soar-suite/103-soar-suite-9-4-0

[2] Laird, J. E. 2014 Soar Cognitive Architecture. Water_Jug_Simple_Agent.zip http://soar.eecs.umich.edu/articles/downloads/agents/153-water-jug-simple

[3] Laird, J. E. 2014 Soar9 Tutorial Part 1. University of Michigan, 1-44

[4] Laird, J. E. 2012 The Soar Cognitive Architecture. MIT Press.

[5] Saxena, D. and Malik, N. K. 2015. Understanding Soar: An outside's perspective. In Proceedings of the Second National Conference on Machine Intelligence and Research Advancement, 43-51.

[6] Man, Y. K. 2013. An Arithmetic approach to the General Two Water Jugs Problem. International Association of Engineers, 145-147.

[7] Leon, F., Horia, Z. and Galea, D. 2005. A Heuristic for Solving the Generalized Water Jugs Problem. Bulletin. Polytechnic Institute of Iaşi, tome LI (LV), 1-4.

[8] Carder, H. P., Handley S. J., Perfect, T. J. 2008. Counter Intuitive and alternative moves choice in the Water Jug Task. Brain and Cognition. Science Direct, 11-20.

[9] Aggrawal and Anderson, R. J. 1988. A Random Algorithm for Depth First Search. Combination, 1-12.

[10] Laird, J. E., 2008. Extending the Soar Cognitive Architecture. Frontiers in Artificial Intelligence and applications. IOS Press, 224-235.

[11] Cognitive architecture http://en.wikipedia.org/wiki/Cognitive_architecture

[12] Soar (cognitive architecture) http://en.wikipedia.org/wiki/Soar_(cognitive_architecture)

[13] Forward chaining https://en.wikipedia.org/wiki/Forward_chaining

[14] Backward chaining https://en.wikipedia.org/wiki/Backward_chaining