Journal of Computer Science

Original Research Paper

Exploring Learning Capability of an Agent in SOAR: Using 8-Queens Problem

[1]Neha Rajan and [2]Sunderrajan Srinivasan

[1]Department of Computer Science, Mewar University, Rajasthan, India
[2]Department of Computer Science, PDM University, Bahadurgarh, India

Article history
Received: 04-11-2019
Revised: 25-03-2020
Accepted: 23-05-2020

Corresponding Authors:
Neha Rajan
Department of Computer Science, Mewar University, Rajasthan, India
Email: neharajan9@yahoo.com

Abstract: Cognitive architecture deals with describing the intelligent behavior of an agent. The description of intelligent behavior states how well an agent can solve and represent variety of problems of the domain-independent task. The intelligence of an agent is considered in terms of its learning capabilities. In this study, we are exploring solving 8-Queens combinatorial problem using SOAR symbolic cognitive architecture. An 8-Queens problem consists of various constraints which is expressed by Constraint Satisfaction Problem (CSP). The constraints are further generalized in the Fuzzy Constraint Satisfaction Problem (FCSP) (a sub domain of CSP), which simplifies the condition of constraints by providing the priority value to the location of queen. This paper provides a way to solve 8-Queens problem by using a heuristic search and backtracking. These concepts are implemented in SOAR to find an efficient solution of similar task. The implementation of 8-Queens in SOAR provides computation efficiency in solving and a way for an agent to learn their own production rules to solve similar domain problems. The 8-Queens problem is analyzed by two parameters. First parameter defines how an agent can learn and transfer rules to solve similar domain problem. The second parameter describes number of chunks required to solve a problem.

Keywords: Procedural Memory, Chunking, Constraint Satisfaction Problem, Backtracking

Introduction

The recent advancement of cognitive architecture has made general purpose intelligent agents smart. These agents can solve tasks just as humans can (Lindes, 2018). The various popular programming language such as Java, C++ and C are not required for these tasks. Moreover, these languages do not offer any built-in features to communicate the problem space as effectively and efficiently as an intelligent agent can perform in comparison. Cognitive Architectures such as GPS, SOAR (Laird, 2012), ACT or Clarion provide the higher levels of abstraction fixed processes, memories and their associated algorithms and data structures. However, the user must still program the tasks at the symbol level to symbolize and process knowledge about the environment for reasoning, problem solving activities and goal-oriented behavior.

Research on cognitive architectures is essential as these architectures provide capabilities like creation and understanding of problem tasks. This general-purpose behavior of agents in cognitive architecture resembles the problem-solving skills of humans (Wray and Chong, 2007). In a previous couple of decades, AI research has actively used specific algorithms to solve particular problems, whereas cognitive architecture aims to cover across diverse sets of tasks and domains. Cognitive architectures are more capable of handling domain related problems or independent domain learning based on real-life issues. The problems are well described by CSP. CSP is used to show how constraints are related logically among several variables. CSP is defined knowledge in a set of hard constraints. These constraints are restricted by particular values, which sometime results in no-solution of the problem. The rigidity is overcome by FCSP that makes hard constraints accessible in such a way that constraint encompasses both decision similarities among permissible instantiations of variables and set preferences amid constraints.

The FCSP is utilized in making symbols in SOAR to define the facts about the real world. These symbols can

be manipulated as some predefined instruction sets. Furthermore, these instruction sets are defined under the term as rule-based deductive reasoning, which manufactures the knowledge about the problem, represents in production rules and also determines how to solve problems using different techniques.

The integration of FCSP and SOAR helps general agent to become intelligent enough to perform a variety of tasks related to the problem. This paper demonstrates the solving method for the 8-Queens Problem and makes an agent intelligent enough to solve similar domain tasks or repeated task.

Here is a brief description of 8-queens problem. The 8 different queen pieces need to be placed on 8*8 chessboard such that each queen piece can follow the following constraints:

C1- No two or more queens can place in the same row
C2- No two or more queens can place in the same column
C3- No two or more queens can place in the same diagonal

An architectural framework is required to represent the above-mentioned problem statement. By providing framework, a computational infrastructure is maintained for developing intelligent agents. These agents are designed to break down problems into sub-problems, look for algorithms to solve these sub-problems and define how to integrate learning with performance and, how to express knowledge so that it can be retrieved effectively and efficiently.

The reason for using SOAR as architectural framework is due to SOAR's ability to simplify computation. The computation is basically based on problem spaces, states, operators and goals.

SOAR manages several memories in its architecture as depicted in Fig. 1 such as working memory, long-term memories as episodic memory, procedural memory and semantic memory. Each memory has its own specific role to perform, such as working memory is responsible for getting knowledge from long-term memories as well as being the basis for initiating action.

The three long-term symbolic memories are independent in performance (Christophe *et al*., 2009) and have separate learning mechanism. The procedural long-term memory is used for retrieving the knowledge that controls the processing. The knowledge in procedural long-term memory (Laird *et al*., 2017) is represented as production rules (other words if-else statement) that match conditions against the contents of working memory and perform actions in parallel. Production rules commonly modify the status of working memory. To control behavior of these production rules, there is need to generate preferences, which are used by the decision procedure to select an appropriate operator (Mininger and Laird, 2018). Operator selection is a major key of making decision and taking action in SOAR.

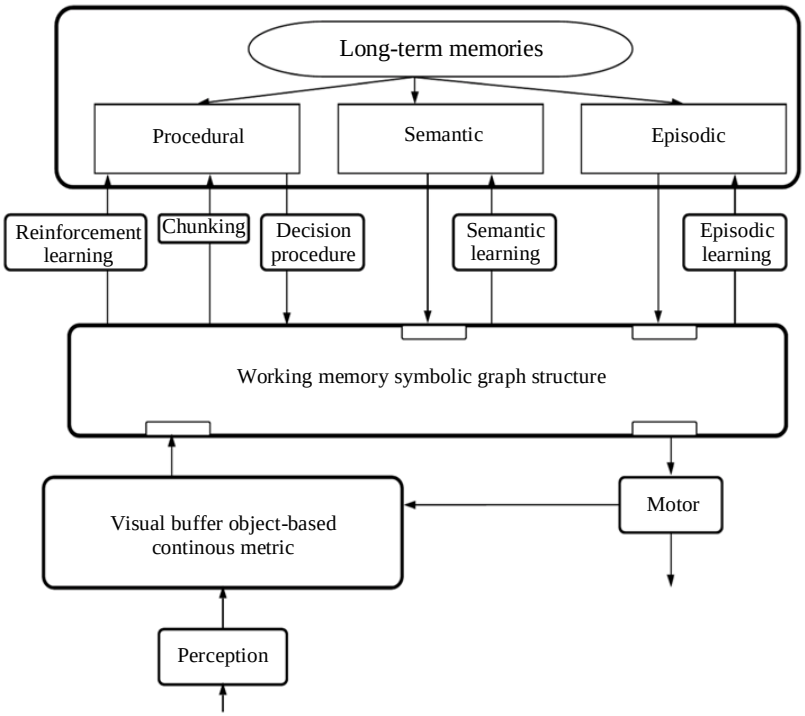

Fig. 1: The architecture of SOAR cognitive architecture

Once an operator is selected, it is applied and causes persistent changes in working memory. The other memories monitor some of the reserved areas of working memory. The changes that occur in working memory can either initiate retrievals from semantic/episodic memory or initiate actions in the environment. Soar has a couple of learning mechanisms (Zwaan and Radvansky, 1998) correlated by procedural memory that consists of chunking and reinforcement learning. Semantic memory store general facts, whereas episodic memory store snapshots of working memory.

Method

Integrated Architecture of FCSP and SOAR.

The design goals of integrated FCSP and SOAR are depicted below in Fig. 2. The Fig. 2 has two majors sections which are used for solving 8-Queens problem. First, the FCSP is applied to those variables that have domain independent priority values. The heuristic search approach is applied using backtracking in SOAR.

Second, is to focus on how SOAR internal function procedure is represented to select the priority value of the variable (Cassimatis, 2006). The knowledge of solving problem is expressed in integrated architecture of SOAR and FCSP which consists of 1. Soar Core 2. Soar Agent 3. External Agent.

Soar Core

Soar Core represents a fixed set of computations which efficiently bring large forms of symbolic knowledge to perform different tasks using various techniques. The core part consists of several computational mechanisms such as working memory (state representation); long-term memory (Laird, 2012) (defines functioning in the form of procedural, semantic and episodic memory); decision procedures (provides a link to interact between working and long-term memory) and learning procedure.

Soar Agent

An agent needs some steps of instruction to solve a problem. The instructions are encoded in the form of production rules which performed in long-term memory where other memories are contributed in the preparation of production rules (Mohan and Laird, 2014).

These rules have general criteria for usage: First, an operator is proposed then a defined operator is selected and lastly the selected operator is applied. Operators perform specific actions to aid in decision making. The decision-making procedure is accomplished by using heuristic approach which is summarized in Fig. 3.

These heuristics steps help in maintaining decision cycle through which all Problem Space Computational Model (PSCM) (Laird, 2012) functions as depicted in Fig. 4 will perform their tasks.

These functions include State Elaborations, Operator Proposal, Operator Evaluation, Operator Selection, Operator Elaboration and Operator Applications. The functionality performed by each function is considered as SOAR's process cycle.

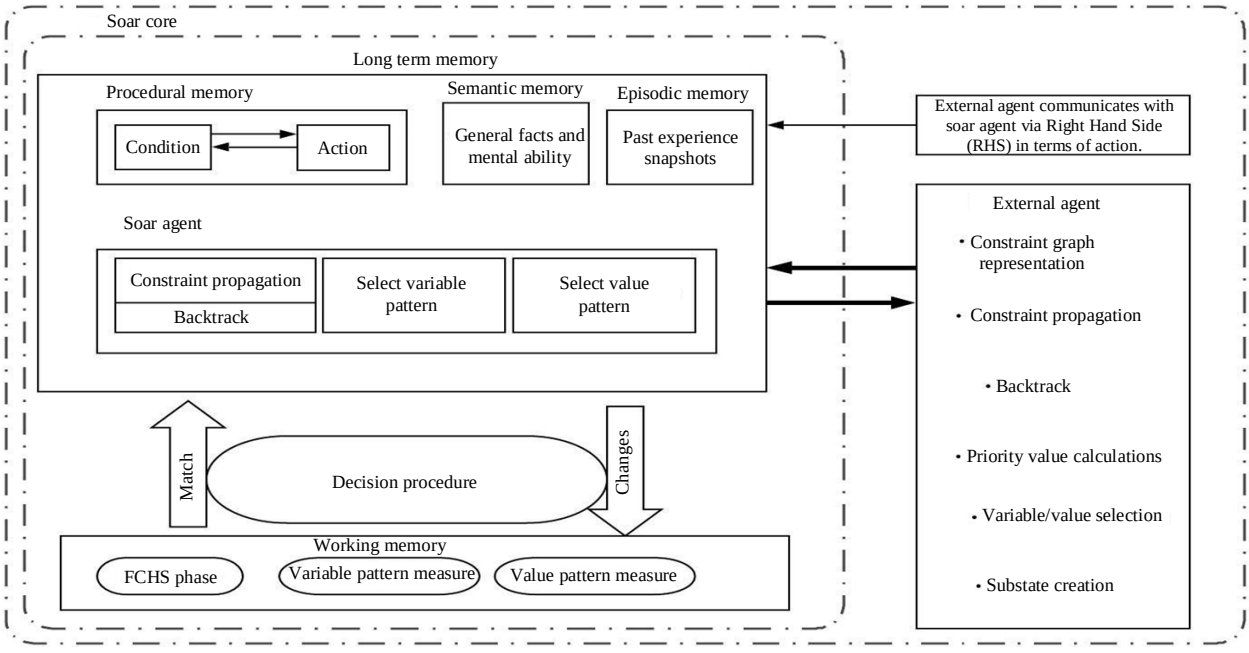

Fig. 2: Integrated architecture FCSP-SOAR

Repeat

1. Mention Constraints (based on FCSP)
2. Represent Propagation within state
   - calculate maximum priority value or assign value to variable (**Algorithm 1 mentioned in 2.2**)
   - If Variable is not found, then perform backtrack (**Algorithm 2 mentioned in 2.2**)
3. Select Variable (based on fuzzy heuristics which suggested by comparing the variable)
4. Select Value (based on fuzzy heuristics which gives higher priority concept)

Fig. 3: Heuristic approach to solve problem

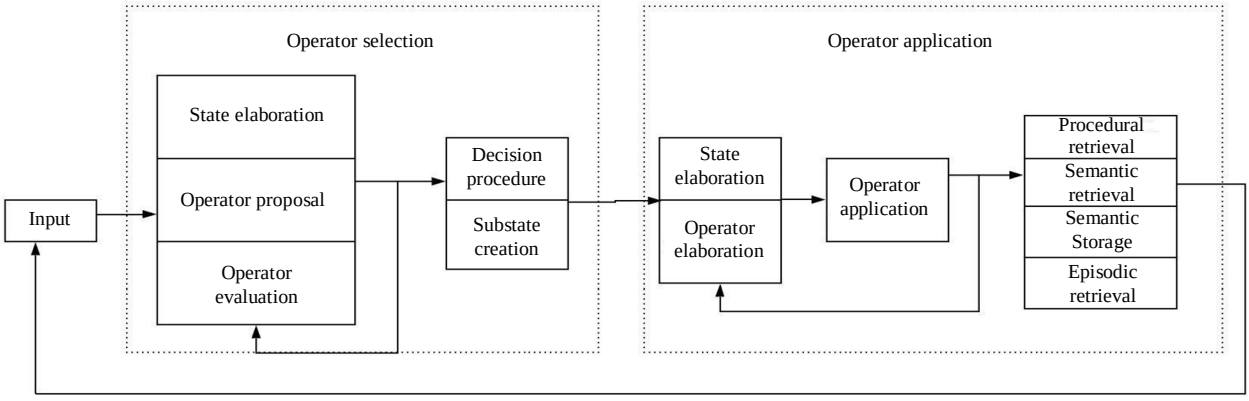

Fig. 4: Soar process cycle

Table 1: Pattern rules and related heuristics

Name	Pattern rules	Heuristics
MPV	P_i, several priority values remaining in the domain.	The highest P_i variable is selected.
DEG	D_i, several chances to move according to constraint are linked to variable.	The largest D_i value variable is selected.

Soar's process cycle provides two possibilities: Operator Selection and Operator Application in Fig. 4. One possibility is procedural knowledge encoded as rules; the other is a problem search in a substate (Langley *et al*., 2009) after an impasse. Operator selection is implemented by a fixed decision procedure that analyses the preferences created by operator proposal and operator evaluation functions and either selects a new operator or detects an impasse and generates a substate. In operator application, procedural knowledge and processing in a substate are available as in other functions. However, operator application is distinguished due to its ability to initiate agent actions in the external environment and where the agent can retrieve structures from semantic memory or episodic memory as well as store structures to semantic memory.

External Agent

The process which is responsible for operating production rules is termed as External Agent. The production rules are most important to select which heuristic will come and perform an action.

Table 1 describes two heuristics that are considered to evaluate a pattern that places queen on the board with 8-Queen problem constraints:

- Degree (DEG)-Degree is a value that defines how many choices are available for a queen to be placed at a location
- Maximum Priority Value (MPV) -Priority is a value that defines the priority of a cell where queen can be placed

Queen will select the cell that has the highest priority value and is within the degree of queen movement.

From the pattern an agent is able to make decision for selecting a particular variable and its value for solving the problem.

The Relationship between Soar and External Agent

The SOAR agent act as the brains and external agent act as body. External agent performs the task requested by SOAR agent and after task execution external agent transfers knowledge and state description back to SOAR agent. The SOAR agent (Gluck and Laird, 2019) maintains a binary graph of constraints known as constraint graph in working memory. This graph helps in keeping status of changing state. This relationship also maintains the number of levels where both agents can share more intrinsic information and can improve their functionality to solve the problem.

Level 0: Architecture retrieves relevant knowledge (Perceiving and comprehending the knowledge).

A Soar agent at the level-0 (knowledge level) consists of an object that can connect with an environment, knowledge and a set of goals. The Principle of rationality (Lee and Anderson, 2001) determines interaction behavior. With the existence of knowledge level; the quality and content of experience are provided to soar agents and they, in turn, select an action for the agent. To setup, a better relationship between the external agent/world and soar agent; the knowledge has to be more general. Generality is defined by the interaction between the knowledge level system and the environment. The interaction decides the range of goals an agent can achieve and the scope of knowledge that an agent can acquire and use.

Level 1: Decision Making

Decision making is a critical component in solving the problem. It is dependent on problem space. Decisions are required to change the state of a problem using the operator. The procedure of decision making (Walsh *et al*., 2013) involves the parts of working memory, which are elaborated by parallel access of long-term memory to exhaustion (Laird *et al*., 2017). The elaboration process determines the changing behavior in existing state to new state. This procedure repeats till it finds the solution of problem space or finds the sub-problem/subgoal of a problem. Otherwise backtrack is applied.

Backtracking is a procedure where the variables of a problem are initiated linearly and the validity of instantiated constraints associated with variables is checked. If any variable breaks one or any condition related to specified restrictions, then backtracking is done to the various recent mentioned variable. The backtracking performs a depth-first search of the potential FCSP solutions typically. The run-time complexity of backtracking is better. Sometimes the complexity of nontrivial problems is exponential because the backtracking standard bears from thrashing; i.e., exploration in several elements of the space continues breaking for the same reasons. To improve this kind of situation (Genesereth *et al*., 2005), a variable can be ordered with a value known as priority value. This will eliminate inconsistent node is eliminated from the domains.

In 8-Queens problem, an agent may traverse constant search for the solution that can lead to thrashing (Hinrichs and Forbus, 2013). Thrashing means searching similar sub optimal solutions again and again. To minimize thrashing, locations must be ascertained with some priority value, so that highest priority value get selected. Figure 5 describes the sequential instantiation of queens on board location with task parameters that generate desired sub-state after satisfying the constraints. These constraints (Mohan *et al*., 2012) are represented as production rules in Soar procedural memory which helps SOAR agent to make solution-oriented decisions.

If the constraints are not satisfied and queen has chosen minimum priority location, then algorithm recommends the procedure of Backtracking in Fig. 6.

The backtracking algorithm in Fig. 6 is modeled on a recursive depth-first search. The preference function is the critical function which holds the selection of location based on priority value. The above-mentioned algorithms help in setting constraints and well get integrated with Soar.

Level 2: Subgoaling Processing

Subgoaling level is related to the decision-making procedure. When decision-making procedure is incompetent to make a selection of solution then an impasse state occurs in problem-solving. An impasse state suggests that the system is not capable to know how to proceed further to solve a problem. To resolve an impasse condition system automatically generate sub-goal state. The impasses and thus their sub-goals are varied from problem to problem.

For subgoaling (Walsh and Gluck, 2014), SOAR provides the capability and knowledge to an architecture for resolving impasse conditions. This is done using a look ahead algorithm. The same approach is applied in FCSP-Soar.

Level 3: Pattern Measure Evaluation

FCSP-SOAR uses three parameters to measure the pattern between location on board and their priority value. The first is a simple calculation that provides priority value preference rules for selecting further tasks.

This preference rule selects MPV. The procedure requires circulation series (looping) to acquire a solution. Second, if a pattern evaluation is directed towards sub-state domain failure, then it gets rejected. Third, if no match is found in the sub-state, then pattern evaluation considered as a “failure” state.

DOI: 10.3844/jcssp.2020.642.650

Algorithm 1: MINIMUM-THRASHING FOR 8-QUEENS

function MINIMUM-THRASHING (*x*, d, n-max-steps) **returns** a solution or failure
inputs: x as location on board, d as domain, Pr represents priority value, n-max-steps as the maximum steps, a queen can perform

1. current_val_queen ← an initial position of queen on board
2. **for** i = 1 to n-max-steps **do**
3. **if** current_val_queen is the solution then return current_val_queen
4. Pr(x) ← a next row is selected from d, domain
5. *MPV* ← Pr(x) of current queen location is checked with other location variable of row, the maximum priority value MPV for Pr(x) is selected that minimizes THRASHING (Pr(x),v, current, csp)
6. set Pr(x) = MPV maximum value in current_val_queen

else return failure or **BACKTRACK**

Fig. 5: Algorithm for minimum thrashing

Algorithm 2: BACKTRACK_8-QUEENS

function BACKTRACK-SEARCH (*d*, *x*) **returns** a solution or failure
return BACKTRACK ({}, *d*, *x*)
function BACKTRACK (state, *d*, *x*) returns a solution, or failure

1. **If** state is completing, **then return** state
2. Preference(x) select move-sequence (*d*, *x*)
3. **for each** value in domain (Preference(x), state, *d*, *x*) **do**
   3.1 **if** value is maximum priority with state **then**
   3.2 add {Preference(x) = value} to stat
   update prev(x) = {preference(x), value}
   backtrack is performed
   else if value is minimum priority then
   update preference(x) = value
   add preference to state
   result ← BACKTRACK(state, *d*, preference (x)).
   if result ≠ failure **then**
   return result
4. remove {*Pr*(*x*) = value} and preference from state
   add {*Pr*(*x*) = old value} from visited.
5. **return** failure

Fig. 6: Algorithm for backtrack

With these pattern measure evaluations, the FCSP-SOAR determines the learning production rules that helps in making an agent learn and practice the specific task. This approach creates small chunks of learning. Resulting chunks have their own set of conditions, either unary or binary conditions, composed in operators.

Results and Discussion

The above-mentioned integrated architecture and number of levels are used for 8-Queens problem as shown in Fig. 7 and describes as follows:

At level 0; an integrated architecture delivers knowledge related to the 8-queens problem which demonstrates that an external agent can interact with the SOAR agent and access different types of available memories (Gluck and Laird, 2019).

To define production rules, procedural memory is utilized and provides enough knowledge to apply some decision procedure to calculate priority value for the movement of queen; and update the working memory with the variable pattern measure and value pattern measure. Throughout this procedure, backtracking algorithms come into play to determine the solution of the problem.

The implementation of the 8-Queens problem is based on PSCM functions and the creation rules of states that are stored in SOAR's long-term memory.

At level- 1, for decision making, several rules, such as, preference rules, elaboration rules, operator selection rules that aid SOAR decision making. These learned rules are passed by using concept of transfer of learning. This concept helps in avoiding unnecessary impasse state. The results of heuristic and learned rules of an agent is based on two computational factors, described below.

Factor 1: Analysis by Learning and Transfer of Learning

The transfer of learning is based on the number of decisions that are required to solve 8-Queens problem where the number of decisions depends on 64(8*8 board-cells) variables that represents each cell on board and 4 constraints C1, C2, C3, C4 mentioned in (Introduction). According to constraints, the SOAR agent required 38 production rules to place queens at their desired locations. Some of the test cases mentioned in Table 2, we applied to evaluate the performance of an agent.

Figure 8, if we look at the 8-queens bars at case-2, the test case represents hardcoded value and variable, where each board location value is hard-coded with a specific constant that defines the priority. The result is decreased number of decisions by 41% in comparison with the random choice of pattern measures used in test case 1. The subgoaling and learning feature were selected for architecture to generate chunks at an average of 60 chunks/subgoaling.

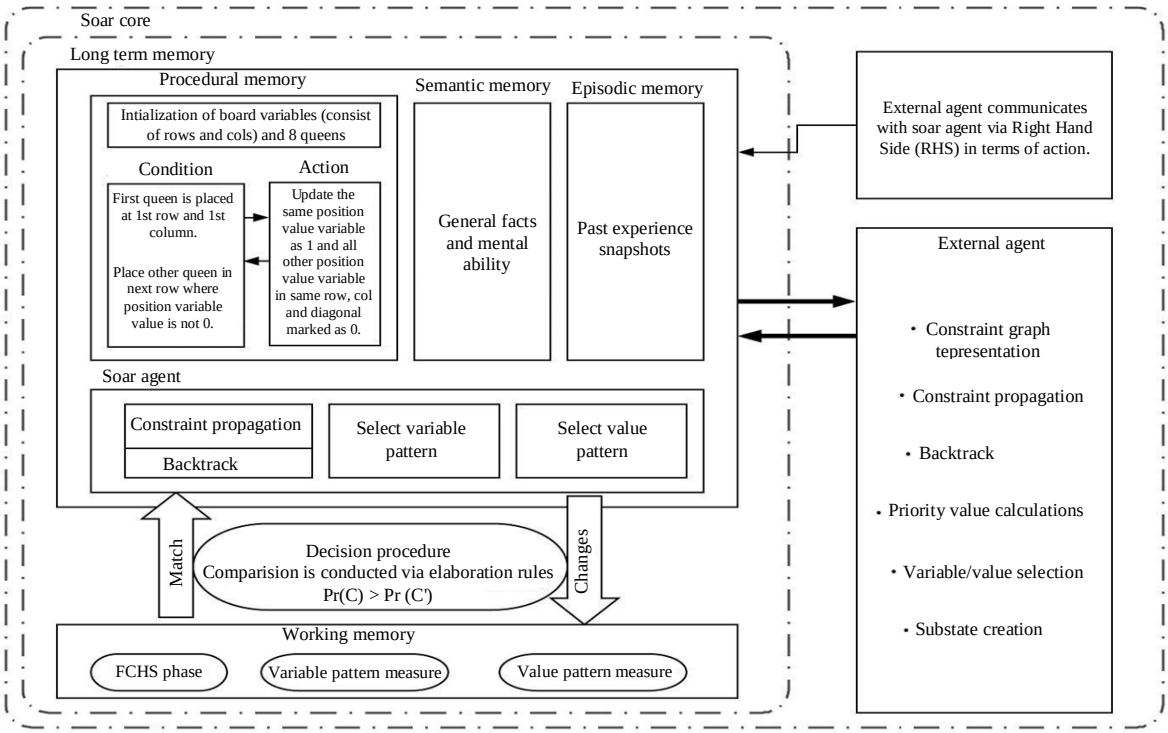

Fig. 7: Representation of 8-queens in FCSP-SOAR

Table 2: Test cases

S.no	Test cases	Description
1	Random Choice	Random variables and values are selected (indifferent).
2	Hard- Coded Choice	Explicitly coded Priority in term of MPV, number of degrees in terms of DEG variable are selected.
3	Internal received chunks	Problem runs working memory and acquire chunks.

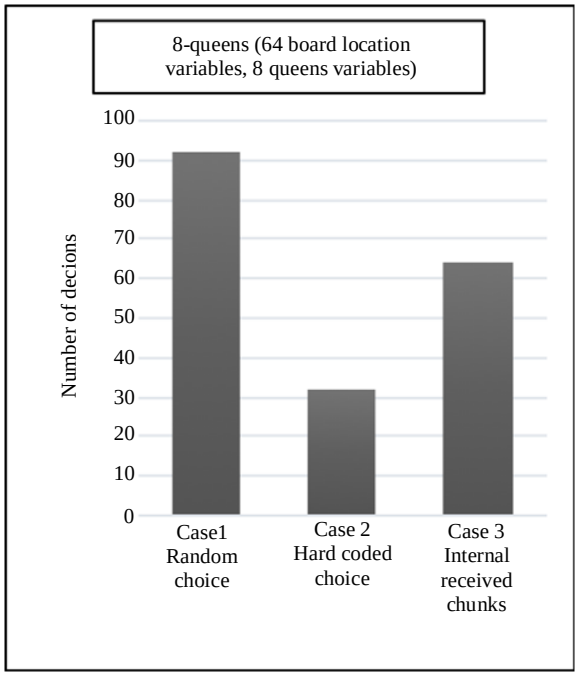

Fig 8: Decisions represented as a Sub-goaling/Learning Test Cases

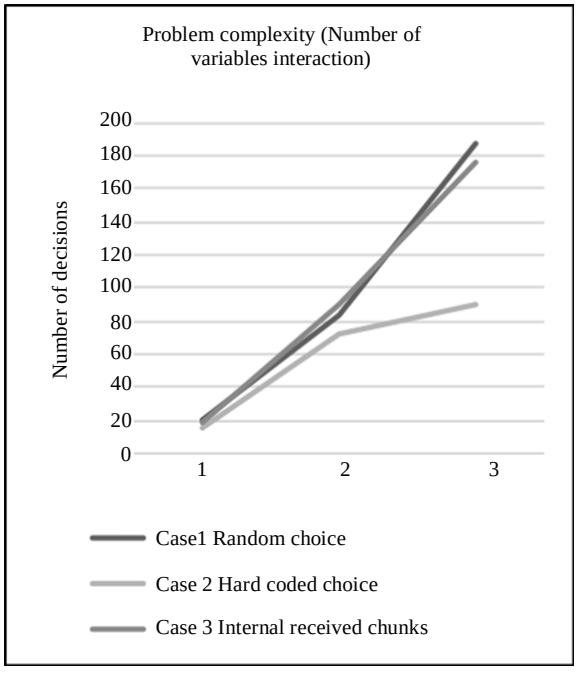

Fig. 9: Comparing of decisions defined as problem complexity

Case 3 explains the way to improve the problem-solving technique that uses the 64 (8*8 chessboard) internal decision cycles. These cycles include backtracking, failure and goal-oriented rules to acquire memorized regulations. These rules are then correlated with hard-coded heuristics mentioned in case 2. The end result is 21% decrease in conclusion sequences. From case 3 results, it can be noted that Soar applied both aggregate range of variables and value patterns to ensure a solution.

Factor 2: Analysis by Competence

Another factor explored as the analysis of competence which describes the performance of stored knowledge depicted in Fig. 9. The performance is determined in terms of complexity of production rules. The firing of production rules satisfies the constraints of the problem and checking learned knowledge- all these tasks are measured by problem complexity under the decision cycles in Fig. 9.

The stored knowledge is utilized many times for random selection. The related chunks, created by random selections, increase in complexity in comparison to hardcoded test cases. This means that the agent consumes maximum time to understand the learned knowledge in comparison with hardcoded values.

Conclusion and Future Work

The unification behavior of FCSP-SOAR provides the representation and handling constraints; involves the concept of providing value as preference associated with prioritized constraints. The concept of FCSP, a subpart of CSPs, offers a range of combinatorial problems to solve problems like 8-Queens.

This specific form of FCSP constraint-based reasoning introduced a SOAR to solve the 8-Queens problem using a generalized set of production rules. The rules include the reasoning of putting together constraint propagation, rule chaining and backtracking.

The production rules are based on an if-then rule which describes the capability of Soar architecture to solve any problem. These production rules further learned by an agent while solving the problem. The integration of FCSP is not that hard to implement as the same production rules are associated with some constraint specific value which further expands the problem search space. When associated priority values are satisfied and a specific solution is got, then the problem is solved otherwise backtracking algorithm is used.

Future work is to improvise the networks of constraints so that subgoaling can be satisfied and related chunks can be learned. These learned chunks provide better efficiency in satisfying a constraint network, which are also known as macro-constraint. The production rules among preference rules and priority values will define for the macro constraint that can be utilized for any combinatorial problem. This behavior can improve the decision procedure's efficiency and can make an agent intelligent to solve similar problems or repeated problems.

Author's Contributions

Neha Rajan: Concept development, implementation testing, validation and writing the manuscript.

Sunderrajan Srinivasan: Implementation validation and proofreading.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Cassimatis, N., 2006. A cognitive substrate for human-level intelligence. AI Magazine, 27: 45-56.

Christophe, L., S. Lakhdar, T. Sébastien and V. Vincent, 2009. Reasoning from last conflict(s) in constraint programming. Artificial Intell., 173: 1592-1614. DOI: 10.1016/j.artint.2009.09.002

Genesereth, M., N. Love and B. Pell, 2005. General game playing: Overview of the AAAI competition. AI Magazine, 26: 62-72.

Gluck, A. and J.E. Laird, 2019. Interactive task learning: Humans, robots and agents acquiring new tasks through natural interactions. Strüngmann Forum Reports.

Hinrichs, T. and K. Forbus, 2013. X goes first: Teaching simple games through multimodal interaction. Adv. Cognitive Syst., 2: 205-218.

Laird, J., K. Gluck, J. Anderson, K. Forbus and O. Jenkins *et al*., 2017. Interactive task learning. IEEE Intell. Syst., 32: 6-21. DOI: 10.1109/MIS.2017.3121552

Laird, J.E., 2012. The soar cognitive architecture, MIT Press.

Langley, P., J.E. Laird and S. Rogers, 2009. Cognitive architectures: Research issues and challenges. Cognitive Syst. Res., 10: 141-160. DOI: 10.1016/j.cogsys.2006.07.004

Lee, F.J. and J.R. Anderson, 2001. Does learning a complex task have to be complex? A study in learning decomposition. Cognitive Psychol., 42: 267-316. DOI: 10.1006/cogp.2000.0747

Lindes, P., 2018. The common model of cognition and humanlike language comprehension. Proc. Comput. Sci., 145: 765-772. DOI: 10.1016/j.procs.2018.11.032

Mininger, A. and J.E. Laird, 2018. Interactively learning a blend of goal-based and procedural tasks. AAAI, 32: 1487-1494.

Mohan, S. and J. Laird, 2014. Learning goal-oriented hierarchical tasks from situated interactive instruction. Proceedings of the 28th AAAI Conference on Artificial Intelligence, (CAI' 14), AAAI, pp: 387-394.

Mohan, S., A. Mininger, J. Kirk and J.E. Laird, 2012. Acquiring grounded representations of words with situated interactive instruction. Adv. Cognitive Syst., 2: 113-130.

Walsh, M.M., E.H. Einstein and K.A. Gluck, 2013. A quantification of robustness. J. Applied Res. Memory Cognition, 2: 137-148. DOI: 10.1016/j.jarmac.2013.07.002

Walsh, M.W. and K.A. Gluck, 2014. Mechanisms for robust cognition. Cognitive Sci., 39: 1131-1171. DOI: 10.1111/cogs.12192

Wray, R. and R. Chong, 2007. Comparing cognitive models and human behavior models: Two computational tools for expressing human behavior. J. Aerospace Comput. Inform. Commun., 4: 836-852. DOI: 10.2514/1.27099

Zwaan, R. and G. Radvansky, 1998. Situation models in language comprehension and memory. Psychol. Bull., 123: 162-185. DOI: 10.1037/0033-2909.123.2.162