# A Computer Program Automatically Acquiring Some Skills for a Simple Design Problem

Gourabmoy Nath

One of the principal dimensions of expertise of a skilled designer is what has been called procedural know-how; experiential knowledge that enables goal-oriented problem solving. In this paper, we present an experiment in machine skill acquisition in a very small and narrow, but yet a demonstrative well-defined design domain through a program that acquires procedural know-how automatically by learning from its experience of practice (computational run) which it applies to produce good design solutions for a slightly different problem. The primitive learning and problem solving methods are based on an architecture for general intelligence called SOAR. The learned knowledge is analysed in terms of its contents, bounds of potential success and failure and why and how can it can be designated as a skill. What is surprising is that, a subset of the acquired know-how contains a very common human strategy used to generate the best solutions to this design problem.

The goal of this paper is to present results of some experiments on machine skill acquisition in designing in a spirit similar to that of human skill acquisition, using a small but demonstrative well-defined design problem instance. This "spirit", the basis of the experimentation, is as follows. A "novice program" starts with some theoretical knowledge of generating and evaluating designs. After one instance of problem solving, the program learns from its own experience of a computational run and converts the theoretical knowledge of best/good designs into practical knowledge. This practical knowledge is a re-representation of the theoretical knowledge in terms of know-how or what to do in order to get to good solutions. Whether the learned knowledge is of any use or not is evaluated by running the program armed with the learned knowledge on a similar design problem. The learned knowledge can be categorized as a skill because it enables the program to direct its future searches to fruitful directions, by following a strategy.

Design expertise is tacit knowledge that is probably more(Akin, 1996) about procedural know-how (Akin, 1974; 1990a) than it is about declarative representations or know-that (Anderson, 1983). It is widely believed that knowledge of the former type can only be acquired only through experience that transforms declarative know-that into procedural know-how. Many researchers have referred to such procedural know-how using varied terminologies in computational, cognitive or mixed contexts, e.g. Smithers.et.al.(1990) calls it "intelligent control"; Coyne (1988) refers to it as meta-knowledge or "knowledge about design actions" that enable "experienced designers to engage in restricted search behaviours"; Oki and Lloyd Smith refers it as "conditions under which rules are useful or useless", Muller and Pasman (1996) address it as assembling of situation-specific schema from experience, Vancza (1991) as positive and negative control heuristics and Mitchell et.al.(1976) states it as practical design knowledge that "consists of knowing what design move to make in response to a given context". The explanation of expertise as transformation of declarative to procedural knowledge is in abundance in the information processing approach to psychology e.g. Langley (1987), state that experts possess a set of "heuristically useful conditions". Anzai (1987) demonstrate through cognitive experiments that experts acquire strategies that can choose moves by solving similar problems repeatedly. Kolodner (1983) describe the transformation of declarative to procedural knowledge as the key factor that distinguishes experts from novices.

The plan of this paper is as follows. First the design problem is described, and then the design process used to solve it is elaborated. Then the mechanism of learning is stated, although the focus of this paper is not on the learning algorithm. The experiment for skill acquisition is described, followed by its results and how a slightly different problem is solved using the skill acquired. The bounds of potential success of the learned knowledge are assessed, as are the limitations. Finally, the nature of the skill is interpreted using some theory.

## 1. The Design Problem

The problem is a relatively simple one; one of designing alternative shapes of rooms that have minimal perimeter for a given area. This problem is formulated as follows: There is a rectangular grid composed of a finite number of squares, enough to accommodate the area of the room. A room is constructed incrementally by allocating squares from the grid. When a grid square is allocated to a room the allocated grid square becomes a cell. A room design is an agglomeration of cells, a cell complex, with every cell having at least one of its adjacent grid squares allocated as another cell. Adjacencies of a cell occupying a given grid square are equivalent to adjacency of that grid square. Adjacency of a grid-square is defined as a condition where there is some grid-square to the north, east, south or west of the grid-square under consideration. Thus if grid-square (x,y)=(2,2) (x=horizontal grid row starting from 0(bottom-left corner), y= vertical grid column starting from 0) is under consideration, then the grid squares (2,3), (2,1), (3,2), (1,2) are adjacent to (2.2) in the north, south, east and west directions respectively. Grid-squares along a diagonal ((3,3), (1,1), (1,3) and (3,1)) that share one common point are not considered adjacent. The area of the room is the number of cells in the cell complex. So the grid is assumed to be large enough to accommodate the area of the room. The space within a cell complex is the space within the room. The external boundary of the cell complex is the shape of the room. The perimeter of the room is the number of unit external edges. For any two adjacent cells, obviously the shared edge is not external, while all non-shared edges are. The shape and the space of the cell complex together constitute a room design. The problem is to find a room shape that will minimize the perimeter of the room.

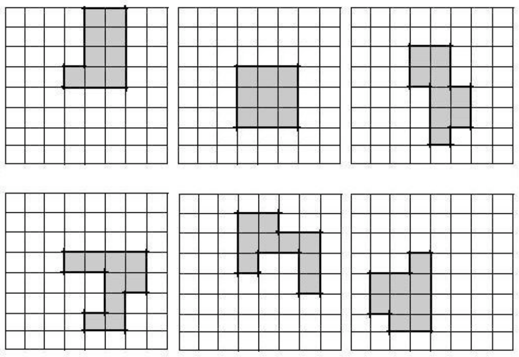
Figure 1. Cells occupying squares of a grid to form different valid cell complex configurations, each with a different external shape and perimeter and of area = 9 units.

## 2. The Design Process

The process of designing and learning was implemented using an integrated architecture for general intelligence called SOAR [1] (Laird, et.al., 1987; Newell, 1990) that is supposed to be an embodiment of a psychological theory of cognition (Newell, 1990). SOAR uses the classical symbolic state-operator abstraction of problem solving with an underlying production system [2] representation of knowledge as two of its fundamental architectural elements. However there are some differences[3] between a traditional production system and SOAR's production system. In the state-operator paradigm, operators transform information in a state to generate other states until a goal state is reached and identified.

The first design generation operator creates an initial design by choosing a location near the center of the grid and allocating a grid square to construct a cell of the room. Subsequent instances of another design operator propose to construct the next cell in an unallocated grid square adjacent (either to the north, south, east or west) to the most recently constructed cell (the current cell). An operator is defined in SOAR as a collection of rules, with at least one rule proposing the operator (like the above) and at least another applying a selected operator.

The definition of the non-initial operator proposal is roughly as follows: "IF there is a vacant grid square <v> in direction <d> adjacent to the current cell <c> propose an operator <O> that would construct a new cell <c1> with the grid square <v> in direction <d>" where <X>, denotes a variable X which can pattern match [4] working memory. The application rule for the same operator is as follows: If there is a selected operator <O> with attributes grid square <v> in direction <d> THEN construct a cell <c1> that occupies grid square <v> in direction <d> and make it the current cell <c>. Before learning, there is no strategy knowledge to choose between proposed operators; thus all proposed operators are applied in order to explore design alternatives.

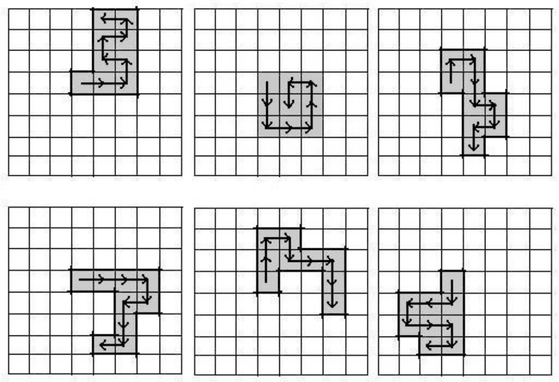

Figure 2. Some cell configurations of area = 9 units generated by different move-sequences from a given initial cell.

At each incremental building stage of the cell complex, one, two or three operators are proposed depending on the number of unallocated adjacent grid squares available. The second cell is proposed in 4 different ways, i.e. in all the cardinal directions. In this way, a cell complex is incrementally constructed. The process of designing is thus one of incremental constructive generation. The design generation stops when the total number of cells in the cell complex equals the area of the room. Then design evaluation is done to assess the quality of the design.

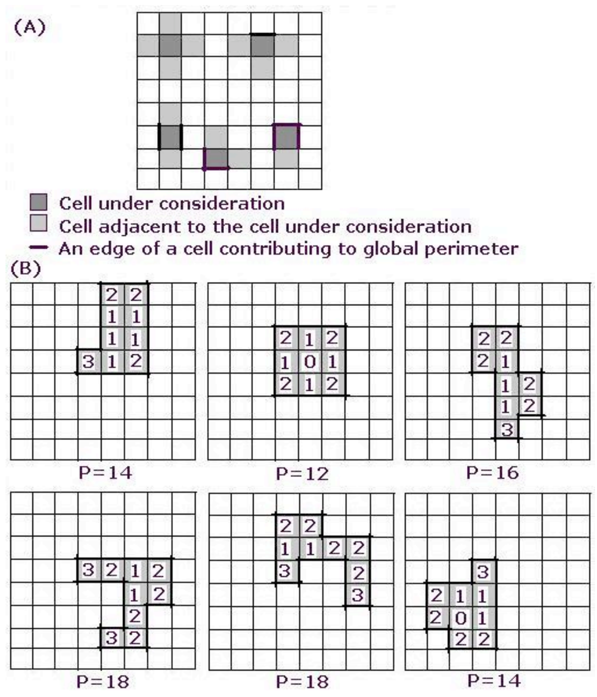
(A)
Cell under consideration
Cell adjacent to the cell under consideration
An edge of a cell contributing to global perimeter
(B)
P=14
P=12
P=16
P=18
P=18
P=14

Figure 3. (A) The cases of perimeter contribution rules of a cell under consideration with 4,3,2,1 adjacent cells with bold edges highlighting the edges contributing to perimeter (B) Application of the above rules, in cell complexes of different shapes, the number in each cell denotes its contribution to the global perimeter indicated by P= a number

The design evaluation criterion here is perimeter of the global shape generated. Global perimeter is calculated by using 4 simple rules: (a) If exactly 3 adjacent grid squares of a cell are occupied then the contribution of the cell to the global perimeter is 1 (b) If exactly 2 adjacent grid squares of a cell are occupied then the contribution of the cell to the global perimeter is 2 (c) If exactly 1 adjacent grid square of a cell is occupied then the contribution of the cell to the global perimeter is 3 (d) If all 4 adjacent grid squares of a cell are occupied then the contribution of the cell to the global perimeter is 0. The global perimeter is the sum of perimeter contributions from all the cells.

## 3. Learning from the experience of designing

The design process described above is tightly coupled with a simple domain-independent analytical learning algorithm called chunking [5] (Laird et.al., 1986; 1987) that is an architectural element of SOAR. The coupling is tight in the sense that learning can take place while problem-solving and what is learned becomes immediately available to the problem-solver that could be applied as seamlessly as it would apply human-encoded knowledge. For chunking to operate, the problem solver maintains a trace of what data got transformed by the design operators to result in what other data in the next state. This essentially captures the exact experience of problem solving.

In this problem, chunking is applied recursively on solution paths that led to optimal designs, when a problem of area= 4 was attempted first. The way chunking is applied and operates in the context of designing and particularly in this problem is described below. When generation stops and a design is found that has the least perimeter (satisfies the design evaluation metric), all intermediate design generations that led to this design are assigned positive credit, which implies that in similar situations in the future, following such solution paths or in other words, each decision in the solution path is desirable. To identify a *similar situation*, or the left hand side of the strategy rule, the chunking algorithm in a design context translates into the following: first find the features, F that were used to evaluate the evaluation metric. Then using the trace of design generation maintained by the problem solver, backtrace the features F in the parent state, that were used to generate the features that were used to evaluate the evaluation metric. F is called the preimage of the design evaluation metric. Then backtrace preimage F to its parent features. Continue backtracing recursively in this manner, until the state just after the initial state is reached. The set of backtraced features at each state along the solution path represents the exact condition under which the next design operator has proved to be profitable, because this solution path finally led to a solution with least perimeter. Each such exact condition is generalized using SOAR's implicit generalization strategy (Laird, et.al., 1986) [6] used in chunking viz.: (a) In the set of backtraced elements at a given level, in the search tree, replace the same working memory element by the same variable (b) Replace a different working memory element by a different variable. For each such "generalized condition-following operator" pair a rule is constructed, that associates the derived general condition to a preference on that operator over any other operator. We thus have a selection heuristic for each decision along the successful solution path.

This selection heuristic (chunk) learned through the analysis of experience acts as strategy knowledge in a future scenario, where there is a similar situation. "Similar" is defined as a pattern match of working memory with the left-hand-side of the learned strategy rule. It is a recognition pattern for the strategy knowledge to be applied. Note that, before learning there was no strategy knowledge, which the problem solver could utilize to select an operator amongst "north, south, east or west". The process of learning creates such strategy knowledge and if applied, could aid the problem solver to choose the right operator (between the available ones: north, south, east, west) at each design construction step. Before learning, the design process is one of generation and evaluation. If a similar situation is recognized, chunks apply to prefer an operator that has previously yielded success. The situation is used to predict the resulting action.

## 4. Skill acquisition experiment and results

The problem-solver armed with the above design process and chunking algorithm (applied recursively) tries to make a first attempt on the above mentioned design problem, where the requirements are to find minimal perimeter shapes of area 4 units. The optimal shape here is obviously the square. Some useful strategy rules are learned from this attempt. This short and simple problem solving session seems to be enough for the problem solver to gather knowledge to produce a solutions that is guaranteed to include the optimal solution to a problem where area is N. An experiment is conducted with N=9, it produces a solution set that includes the optimal solution.

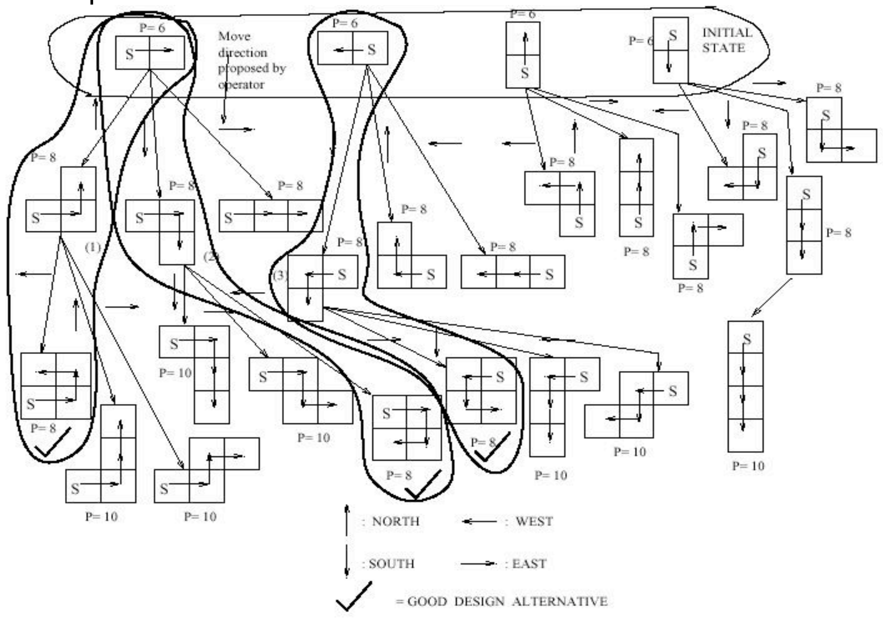

Figure 4. Design solution paths for area=4. Solutions emerging from some intermediate solutions, before the last level are not shown.

Figure 4 shows the different solution paths for the problem where area = 4 units. The best solutions are indicated with a symbol in the figure and their solution paths are annotated using closed curves in bold. The learned rules in the syntax of the programming language were examined by the program designer and are depicted graphically in Figure 5 for easy comprehension of their contents. In essence, there are two kinds of learned rules. The first rule prefers any operator that proposes a grid square such that on creation of the cell in the grid square, a global L-shape is formed; the second prefers an operator that proposes an operator that would close an existing L-shape cell configuration to form a global square shaped cell configuration. The learned rules are saved for future use.

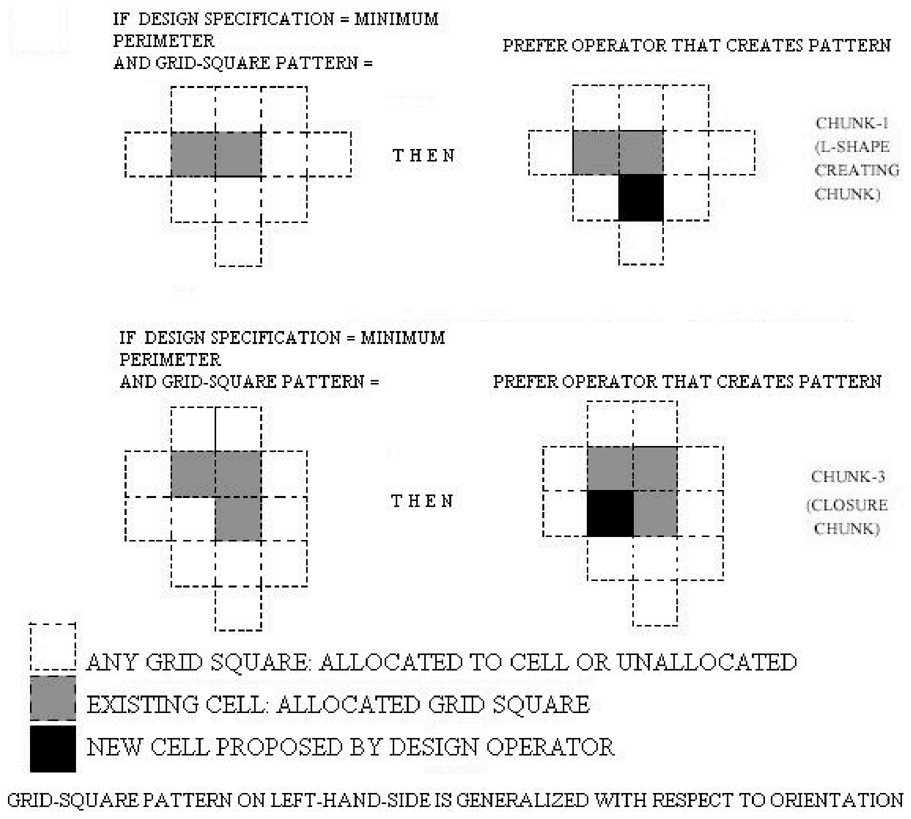
IF DESIGN SPECIFICATION = MINIMUM PERIMETER AND GRID-SQUARE PATTERN =
THEN
PREFER OPERATOR THAT CREATES PATTERN
CHUNK-1 (L-SHAPE CREATING CHUNK)
IF DESIGN SPECIFICATION = MINIMUM PERIMETER AND GRID-SQUARE PATTERN =
THEN
PREFER OPERATOR THAT CREATES PATTERN
CHUNK-3 (CLOSURE CHUNK)
ANY GRID SQUARE: ALLOCATED TO CELL OR UNALLOCATED
EXISTING CELL: ALLOCATED GRID SQUARE
NEW CELL PROPOSED BY DESIGN OPERATOR
GRID-SQUARE PATTERN ON LEFT-HAND-SIDE IS GENERALIZED WITH RESPECT TO ORIENTATION

Figure 5. Graphic depiction of the two types of learned rules

The next stage of the experiment was to examine whether the knowledge gained from experience was useful for a 9-celled problem. It is obvious that the optimal solution is a square of edge dimension 3 and perimeter 12. The program only produced configurations that completely (perimeter=12) or nearly satisfied requirements (perimeter = 14) as shown in Figure 6. Figure 7, shows a generative trace tree of the solutions produced using the learned action selection heuristic knowledge automatically derived using the previous problem. The figure also shows pattern matches of "similar situation" of the chunks that resulted in the selection heuristics being applied.

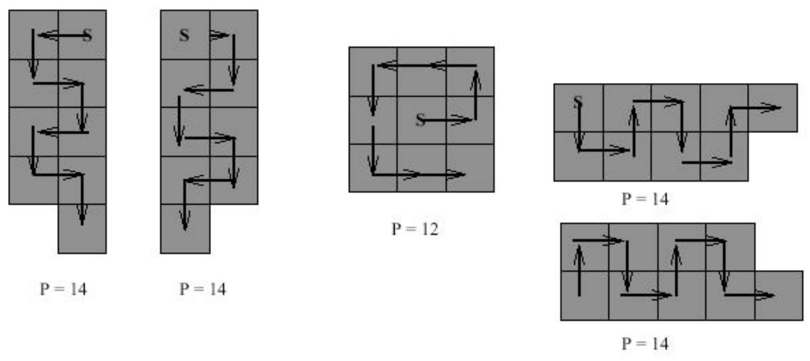

Figure 6. Some solutions produced by using learned strategy rules for area= 9 units

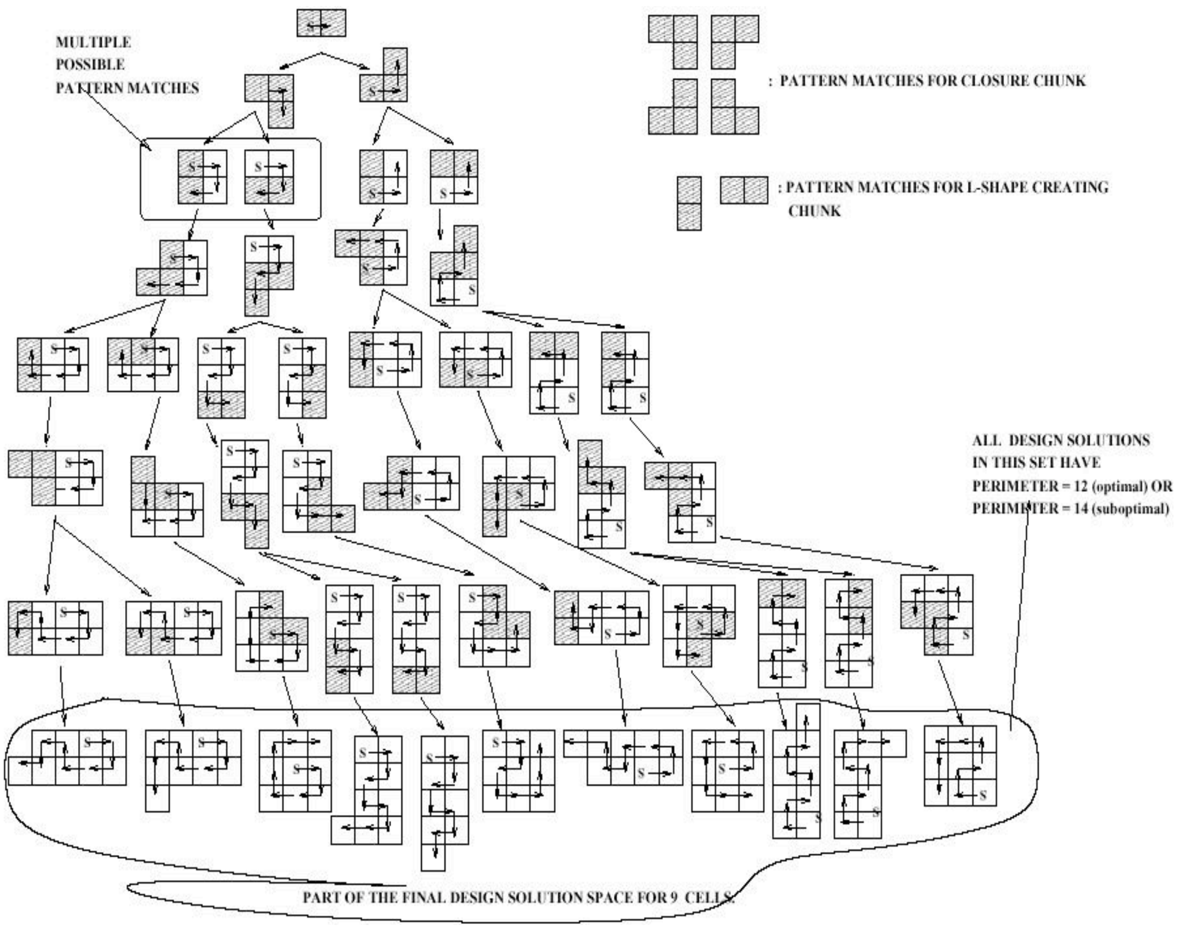
MULTIPLE POSSIBLE PATTERN MATCHES
: PATTERN MATCHES FOR CLOSURE CHUNK
: PATTERN MATCHES FOR L-SHAPE CREATING CHUNK
ALL DESIGN SOLUTIONS IN THIS SET HAVE PERIMETER = 12 (optimal) OR PERIMETER = 14 (suboptimal)
PART OF THE FINAL DESIGN SOLUTION SPACE FOR 9 CELLS

Figure 7. Strategy knowledge influencing design generation

## 5. Evaluating the bounds of potential success of the learned knowledge

Because the entire approach to designing this program has been symbolic, the learned rules are human examinable and the bounds within which these rules will be successful can be examined. This is quite unlike many sub symbolic methods of learning where the knowledge learned remains enmeshed within tuned weights and where such bounds could only possibly be derived through experimentation.

If the human is given this problem, a common approach [7] to solve it is often using fundamental geometric knowledge acquired in school. The square is a rectilinear shape that has the least perimeter for a given area. Thus anything close to a square will be the optimal solution. If the human starts working with the moves as the computer program does, he will soon discover that a spiral move sequence will achieve a square. The two strategy rules learned by the program subsume this common human strategy and are thus guaranteed to produce an optimal solution (see derivation of solution number 3 and 8 from the left hand side in the last level of the generative trace tree in Figure 7). This is indeed quite surprising. What is also surprising is that these two rules, also subsume, other strategies for producing the square (see solution number 6 and 11 from the left hand side in the last level of the generative trace tree in Figure 7).

But the L-shape creating strategy can produce the L in a direction contrary to that of the spiral move sequence, as can be seen in the two solutions of 6 and 7 cells on the extreme left-hand side of the 5th and 6th level of the tree in Figure 7. This is advantageous in producing sub optimal solutions if one is interested in them or even non-square shaped optimal solutions where the required area is low (e.g. 9 or 10). But as the area tends to be high, these two strategies, if not applied in the correct sequence tend to produce long elongated shapes and is not very advantageous as long as optimal/sub optimal perimeter of the shapes is a requirement. In such a case, the optimum perimeter is far lower than that of an elongated shape (consider area=100, optimum = 10 cell x10 cell square for which perimeter=40, a 2 cell x 50 cell rectangular configuration will make the perimeter 104). Finally, it may be noted that had the L-shape creating chunk been such that the L-shape could have been formed only in one way, viz. in the direction appropriate to maintaining the spiral move sequence, it would have matched exactly the common human strategy. But the way the pattern has been learned makes the L-shape creating strategy applicable in 2 ways resulting in 2 solutions. This surprises us, because its application (a) creates the square in another way (see solution number 6 and 11 from the left hand side in the last level of the generative trace tree in Figure 7) at a different location in the grid (b) creates sub optimal solutions for low values of area. On the other hand, for higher values of area, these sub optimal solutions of elongated form are not so good solutions.
The human could also immediately draw a square if the area is a perfect square. If the area is not a perfect square, it is quite common for the human to apply the spiral move heuristic. Thus for N= 10, the human produces a 9 celled square and adds the 10th cell adjacent to any boundary cell. It is common, although is not always the case, for the human to miss another optimal solution (a 2 cell X 5 cell rectangle) which the learned strategy heuristics can produce, but applying the L shape creating rule in a direction contrary to the spiral move sequence. This effect of the learned strategy rules is also quite surprising. To achieve

this skill level, more practice is needed by the program that would result in the creation of more chunks, just as the human will need to experience this phenomenon by working out a few more problem instances.

This work demonstrated the automatic acquisition of some human-like strategies for generating solutions to a simple design problem. In general, this approach may not work for demonstrating machine expertise acquisition for any general design optimization problem. When we analyse why this approach works in this problem, we find that the problem has some interesting properties, which was the reason behind the success of this method. This property was that good or optimal solutions could be produced by a repetitive sequence of the same design operators. In this case, sequential application of the L-shaped and square forming heuristic will lead to a spiraling move sequence. The experimentation with the 4-celled problem, helped in learning a generalization of that sequence; it is a generalization because once the square is formed, the learned strategy rule can form the L-shape in 2 ways. The application of learned knowledge for the 9-celled problem simply applied that sequence. The Tower of Hanoi problem is one such problem, though not in a design domain, which can be solved through cyclic sequences of operator application. There is evidence that people can also acquire such kind of knowledge (Anzai and Simon, 1979). Shell and Carbonell(1989) used machine learning algorithms to acquire similar knowledge (macro-operators) in a computational system.

## 6. Dissecting the essence of the skill through some theory

Despite the problem domain being small and simple, the experiment was instrumental in demonstrating automatic acquisition of human like skill in the domain. What has changed as a result of experience in the computer program? It is simply the addition and use of strategy knowledge that augments the existing knowledge by dictating which rule to apply in what situation. This is exactly what amount to "practical experiential knowledge" or "know-how"(procedural knowledge), as opposed to the earlier representation of the same knowledge as "know-that"(declarative knowledge) (Anderson, 1983; Akin, 1990). The essence of practicality in this knowledge is the goal-orientedness, which is captured in the left hand side of the rule viz. the situation that causes the pattern match of the strategy rule. The process of generation and evaluation is replaced by a process of knowledge-based prediction or foresight of a good decision at each step along a solution path, that enables problem solving to progress through implicit evaluation.

We also notice that the theoretical concept of a "good design" has changed representation. Before learning, it was simply a design that would minimize perimeter. After learning, the representation of the concept of a best design is changed to "what to do in order to get to the best" that also include conditions of the actual environment in which the theoretical concept of the best design was used. When Eastman (2001) posits that learning and mastering new design representations are central to developing design expertise, he probably refers to the genesis of such new representations. Bringing a medicinal analogy into picture, the theoretical concept of an antibiotic as probably that of a "drug that is used to cure bacterial infections". A more pragmatic, goal-oriented, know-how oriented concept representation of the antibiotic could be as follows: " a drug that is prescribed thrice a day for 10 days when a patient's blood report shows a significantly increased count of white blood cells, and the patient has a history of increased body temperatures > 39 degree centigrade in the last three days and the patient does not have a prior history of gastro-intestinal problems". Akin(1996) elucidates procedural know-how through an example in design; To use his example in this context, designers do not use the fundamental theoretical principles of geometry and geography to design an overhang for a glazed opening at a given altitude, azimuth and latitude, instead the designer has probably in his semantic memory chunks that somehow encode the subtasks of finding out extreme angles of solar incidence on a surface in order to determine the extent of horizontal projection of the overhang.

It is also seen, that in this approach, the concept of an optimal design becomes distributed (in terms of procedural knowledge) across the condition of each of the learned strategy rules, at each design construction step. Because there is no relation between the rules, or no concept of a learned strategy rule applying over a sequence of design decisions, it is natural to hypothesize that the rules could be over general. Over generality is in fact observed, when the strategy rules could in fact lead the design generation, say for a problem of area 100 to produce a elongated shape that is not even sub optimal.

The chunking algorithm, the foundation of learning in SOAR, is founded on the principles Chase and Simon's chunking theory(1973), which proposed that chess players, as well as other experts, acquire a large number of chunks (familiar units denoting perceptual patterns) through practice and study. Chunks are defined as groups of parts of some complex stimulus that are related to each other using some intricate web of relationships. In the case of chess, chunks provide valuable information such as plausible moves, potential plans, or evaluations of chess board positions. This theory has been used to explain the performance of chess masters, who seem to recall a briefly presented position almost perfectly. As per this theory, this phenomenon is possible because these experts can recognise more and larger chunks than weaker players, and they can find better moves because chunks give rapid access to key information that may be elaborated by further look-ahead search. Chase and Simon's theory has been, considerably validated by a wealth of protocol studies in other domains of expertise, including studies in the domain of architectural design by Akin(1986,1990a,b).

We have shown in the context of this small problem, how learning from computational experience changes/augments reasoning methods, concept representations so that problem-solving becomes more goal-oriented.

Before concluding, we reiterate two important facts (a) that the claims of similarities between the processes of human and machine skill acquisition is only to the degree(Newell, 1990) that a computational model of human cognition like SOAR permits, in spite of the fact that if we view skill acquisition in terms of input and output there is marked similarity (b) the acquisition of strategic knowledge of the type demonstrated in our experiment is probably one very important component of design skill or know-how, but there are other important incrementally acquired know-hows. Akin(1996) used in designing for problem re/formulation(Archea, 1986), problem decomposition, solution reassemblage which are not addressed in this work.

## 7. Utility of the experiment

The author believes that this work is a simple worked out example that is relevant for *understanding* the principal component of design expertise: learning reusable strategy heuristics from problem situations, so that next time under a similar situation a good design solution could be proposed quickly without having to go through an elaborate search process, sometimes even by mere inspection utilizing the acquired heuristic predictors. This is the most critical behaviour that distinguishes a novice from an expert. In other words understanding how designers transform a piece of bookish theoretical knowledge into practical knowledge for fast solution generation. The *understanding* in this case is of course with respect to the much broader general theory of human cognition proposed by Alan Newell (1990) and its architectural equivalent SOAR. The example microcosm serves to demonstrate as to what design experience actually is, how knowledge representation gets gradually changed with practice in a given design domain to influence the design process and why and how the learned knowledge is useful in producing better designs. The reader may refer to the superset of this work (Nath, 2000) for details of the learning algorithm or its application to a more involved real architectural design problem. Similar experiments with other cognitive architectures [8] can be conducted to gain an understanding of the design cognition mechanisms that work in the background to result in what is seen as expert behaviour. To emphasize again, this understanding, is with respect to the theory of cognition on which the chosen architecture is founded upon. Such an understanding is always beneficial, especially, when the objective is to support designers with advanced computational tools supporting one or more dimensions of design activity.

The reported experiment in this paper, for example, is relevant to supporting designers with computational tools that accumulate and subsequently reuse problem solving experience in the background, while solving problems in the foreground; the behaviour of the problem solver typically evolves with experience. Following is a description of how the author sees this mechanism could be translated into part of an effective design tool or a useful feature of an existing design tool. Design problem solvers often have rule-based means of incrementally generating or constructing a structure of the artifact through a sequence of transformations and criteria for evaluating which of the generated structures are suitable. If a trace of this generative and evaluative process is maintained, as soon as the evaluation leads to a good/bad design, the chunking algorithm of SOAR, or in fact another explanation-based learning method [9] can work in the background to derive useful goal-oriented designing strategies for the future. These strategies can be ones, which would direct future search to produce certain desirable features resulting in good designs, or they can be strategies that do not produce some undesirable features or bad designs. The theorectical concepts of 'good' and 'bad ' are of course user definable, what to do in order to produce the good and/or avoid the bad is found out by learning algorithm. The learning and the impact of such strategies on design solutions have been explored in detail by Nath (2000). The learned heuristics would be used when the designer attempts to solve the same or similar design problem with parameter changes, while maintaining the same representation of the artifact to be designed as also the associated computational design process. One of two things then happen when solving a similar problem the 'next time' with the learned heuristics enabled: (a) If previously learned knowledge match, it would be applied to bypass 'known' problem solving situations to either explicitly propose a path that leads to a good design (b) If there is no match then new knowledge will be learned to augment the knowledge base of the problem solver so that a previously unknown situation would henceforth be 'known'.

Both cases of positive and negative heuristic acquisition by the proposed tool will save considerable time for the designer, who would be using that tool in terms of quickly arriving at good designs or even designs that satisfy the designer's preferences and not repeating 'mistakes'. The tool could also support visualizing these machine-acquired pattern-matching strategies, editing and saving them, so that a subset of the learned strategies could be loaded and used at the next program run, as per the choice of the designer, leading to different classes of design solutions. Novice designers could also understand the nature of a complex design space, with the aid of such a tool. With these aspects of design activity taken care of, the focus of the designer shifts much more towards formulation and exploration with different search spaces for the design problem and experimenting with each one with the proposed tool.

## 8. Conclusion

This paper reported an experiment on the automatic acquisition of a kind of procedural know-how by a computer program while solving a small well-defined demonstrative design problem and then applying the learned know-how to produce good results for a slightly different design problem. The program was founded upon primitive problem solving and learning methods encapsulated in the SOAR architecture for general intelligence. What is interesting is that (a) the knowledge learned in the context of this problem surprisingly subsumes a common human problem solving strategy (b) the part of the learned strategy that does not match the common human strategy sometimes produces surprising results (c) the declarative concept representation of a good design is changed and distributed across strategy rules as procedural know-how. Because of the availability of know-how, the design reasoning method undergoes a change, from search-based it becomes knowledge-based, from uniformed to informed, from generative-evaluative to predictive, similar to what is postulated in the theories of skill acquisition in cognitive science. The work is useful for understanding the nature of design expertise and can be used as a feature in computational tools for supporting designers.

## Acknowledgements

This work was supported by an AUSAID fellowship and was carried out in majority at the Key Centre of Design Computing and Cognition, Sydney, Australia as a part of the author's PhD research.

## Notes

[1] SOAR: SOAR is a computational model of human cognition. It provides an architectural framework to model problem solving and learning. SOAR uses production rules to represent permanent knowledge, object-attribute-value representation of temporary knowledge, problem space representation of tasks and subtasks to be solved, automatic subgoaling as the single method for generating goals and chunking as the single learning mechanism.

[2] Production System: A production system consists of an unordered collection of production rules (Forgy, 1982). The data operated on by the productions is held a global database called working memory. All data in working memory is defined using object-attribute-value triples. Each such data is called a working memory element. Each production consists of condition patterns on the left-hand-side and actions, which specify addition or deletion of working memory elements on the right-hand-side. Usually an interpreter executes a production system by performing the following actions: (a) Match: Evaluate the left-hand sides of the productions to determine which of them are satisfied, given the current contents of working memory (b) Resolve Conflict: Select one production to apply (c) Act: Perform actions on the right-hand-side of the production chosen for application, after conflict resolution (d) Go to Match.

[3] SOAR-production system differences: In SOAR, a classical AI state-operator abstraction is used on top of a production system. However standard conflict resolution is replaced by the process of all productions firing in parallel until there are none to fire. These rule firings propose one of a small number of possible preferences on working memory elements. A fixed decision process evaluates the preferences on these working memory elements to arrive at the conclusion as to whether a new working memory element is to be added or an existing working memory element is to be deleted or the value of a working memory element is to be modified (a combination of the above).

[4] Pattern Match: When some variables pattern match working memory, other variables as a side-effect get bound to working memory data producing a complete and consistent match, as is common in production

system pattern-matching engines. When the left hand side of the rule matches, the right hand side changes working memory accordingly.

[5] Chunking: The chunking algorithm in SOAR is a method of summarizing an elaborate set of actions that were performed to achieve some subgoal/task that is a part of the global goal. This summary is the result that was achieved in the subgoal. A pattern is also learned, so that this pattern could be used to recognize an opportunity to apply this summarization result in the future. The motivation is that the process of problem solving can then be bypassed in the future and simply be replaced by the end result, resulting in efficiency. A chunk is rule associating the two, the latter being the left hand side and the former, the right hand side.

[6] Implicit Generalization: A generalization strategy that uses the data abstractions used in reasoning as a basis for generalizing concepts.

[7] Common Human Approach: Unfortunately this phrase is a bit subjective. We have no rigorous study of properly sampled human subjects and their problem-solving strategies on this problem that could conclusively establish that this is indeed a common approach.

[8] Cognitive Architecture: A cognitive architecture (Sloman, 2003) refers to the design and organization of the mind. Theories of cognitive architecture strive to provide an exhaustive survey of cognitive systems, a description of the functions and capacities of each, and a blueprint to integrate the systems. Such theories are designed around a small set of principles of operation. Theories of cognitive architecture can be contrasted with other kinds of cognitive theories in providing a set of principles for constructing cognitive models, rather than a set of hypotheses to be empirically tested.

[9] Explanation-based learning (EBL): The best way to understand explanation-based learning is to follow the example. Given a training example, C1 for learning, such that, C1 is light; is made of porcelain; has a decoration, a concavity, a handle, and a flat bottom. Given, theoretical background knowledge for the functional definition of a cup as follows: If an object is stable and enables drinking, it is a cup. If an object has a bottom which is flat, it is stable. If an object carries liquids and is liftable, it enables drinking. If an object is light and has a handle, it is liftable. If an object has a concavity, it carries liquids. Can it be proven that C1 is a cup ? If the answer is yes, the proof tree is the explanation of why C1 is a cup. From the explanation, EBL algorithm tries to find features, which if present in any object will result in the conclusion that the object is indeed a cup. The output of explanation-based learning applied to the above example is a new rule: If an object is light, has a concavity, has a handle, and has a flat bottom, it is a cup. Note that functional features are converted into structural features which enable cup recognition.

## References

Akin, ö (1974) AIM: Architectural Inference Maker, in Proceedings of the International conference and Exhibition on Computers in Engineering and Building Design(CAD'74), London, pp 506-521.

Aki, ö (1986) A Formalism for Problem Restructuring and Resolution in Design, Planning and Design, 13, pp 223-232.

Akin, ö (1990a) Necessary Conditions for Design Expertise and Creativity, Design studies 11, pp 107-113.

Akin, ö (1990b) Expertise of the Architect, in Rychener, M.D., Expert Systems for Engineering Design, Academic Press, NY, pp 173-196

Akin, ö and Akin, C. (1996). Expertise and Creativity in Architectural Design, in Proceedings of the First International Symposium on Descriptive Models of Design, Taskisla, Istanbul, Turkey.

Anderson, J (1983) The Architecture of Cognition. Cambridge, MA, Harvard University Press.

Anzai, Y. (1987). Doing, Understanding and Learning in Problem Solving, in Klahr, D., Langley, P. and Neches, R. (eds.) Production System Models of Learning and Development, pp 55-97. MIT Press, MA.

Anzai, Y. and Simon, H. (1979). The Theory of Learning by Doing, Psychological Review, 86, pp 124-140.

Archea, J. (1986) Puzzle-Making: What Architects do When Noone is Watching, in Kalay, Y., Computability of Design, pp 37-52, Wiley Interscience, New York.

Chase, W.G. and Simon, H.A. (1973) Mind's eye in Chess, in Visual Information Processing, W.G.Chase (ed.), pp 215-282, Academic Press, New York.

Coyne (1988) Logic models of design, Pitman, London.

Eastman, C (1989) New Directions in Design Cognition: Studies of Representations and Recall, in Design Knowing and Learning: Cognition in Design Education, Eastman, C.M., McCracken, W.M. and W.C.Newstetter (eds), pp 147-198, Elsevier.

Kolodner, J. (1983) Towards an Understanding of the Role of Experience in the Evolution from Novice to Expert, International Journal of Man Machine Studies, 19, pp 497-518.

Laird, J., Newell, A. and Rosenbloom, P (1986) Chunking in SOAR: The Anatomy of a General Learning Mechanism, Machine learning 1, pp. 11-46.

Laird, J., Newell, A and Rosenbloom, P (1987) SOAR: An Architecture for General Intelligence, Artificial Intelligence 33, pp. 1-64.

Langley, P. (1987) A general theory of discrimination learning, in Production System Models of Learning and Development, in Klahr, D., Langley, P. and Neches, R. (eds.), pp 99-161, MIT Press, Cambridge, MA.

Muller, W. and Pasman, G. (1996) Typology and Organization of Design knowledge, Design Studies, 17(2), pp 111-130.

Mitchell, W., Steadman, J. and Ligget, R. (1976), Synthesis and optimization of small rectangular floor plans, Environment and Planning, B, 3(2), pp 37-70

Nath, G. (2000) A model of situation learning in design, PhD Thesis, Department of Architectural and Design Science, University of Sydney, Australia.

Newell, A. (1990) Unified theories of Cognition, Harvard University Press, Cambridge, MA

Oki, A. and Lloyd Smith, D. (1991) Metaknowledge Reasoning in Civil Engineering Expert Systems, Computers and Structures, 40(1), pp 7-10.

Shell, P. and Carbonell, J (1989). Towards a general framework of composing disjunctive and iterative macro-operators, in Proceedings of the 11th Joint Conference in Artificial Intelligence, Detroit, MI, pp 596-602.

Sloman, S. (2003), Cognitive Architecture, http://cognet.mit.edu/MITECS/Entry/sloman, MIT Encyclopaedia of Cognitive Science(MITECS).

Smithers, T., Conkie, A., Doheny, J., Logan, B., Millington, K. and Tang, M.X. (1990). Design as Intelligent Behaviour: an AI in design research programme, Artificial Intelligence in Engineering 5(2), pp 78-108.

Vancza, J. (1991). Improving Design Knowledge by Learning, in Intelligent CAD-III, Yoshikawa, H., Arbab, F. and Tomiyama, T (eds), Elsevier, pp 289-305.