Research in Engineering Design (2004) 15: 77–92
DOI 10.1007/s00163-003-0042-4

ORIGINAL PAPER

Jarrod Moss · Jonathan Cagan · Kenneth Kotovsky

Learning from design experience in an agent-based design system

Received: 25 February 2003 / Revised: 5 May 2004 / Accepted: 5 May 2004 / Published online: 20 July 2004

Abstract A learning mechanism based on findings in cognitive science was added to an agent-based computational design system to determine if this mechanism would enable the system to learn from its experiences and transfer useful design knowledge to new problems. Learning and transfer were evaluated by examining how well knowledge learned while solving one problem could be applied to solve the same problem more effectively as well as how well this knowledge could be transferred to new design problems. An existing system, A-Design, was enhanced by giving it the ability to learn and store useful design knowledge so that this knowledge can be used in new design problems. Three electromechanical design problems were used to evaluate this new learning mechanism, and results indicate that this basic, cognitively based, learning mechanism is successful at transferring design knowledge to new problems with a few limitations. This knowledge transfer leads to a more effective design process.

Keywords Agents · Synthesis · Learning · Cognitive science

J. Moss
Department of Psychology, Carnegie Mellon University, Pittsburgh, PA 15213, USA

J. Cagan (✉)
Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA
E-mail: cagan@cmu.edu
Tel.: +1-412-268-3713
Fax: +1-412-268-3348

K. Kotovsky
Department of Psychology, Carnegie Mellon University, PA, Pittsburgh, Pittsburgh 15213, USA

1 Introduction

Human designers typically become better at designing devices in a domain as their experience in that domain increases. Computational design systems should also be able to learn from their design experiences so that they can learn to produce higher quality designs in a shorter amount of time. The work presented here is an attempt to enhance an existing design system called A-Design to give it the ability to learn from its experiences. In particular, the learning mechanisms that were added to the system are based on ideas about learning derived from the cognitive science literature.

The process of acquiring expertise is gradual and has been documented in a number of other domains (Chase and Simon 1973; Larkin et al. 1980; Reitman 1976; Richman et al. 1995). The reason for this performance improvement with experience is due in part to the expert's ability to utilize previous experiences. One aspect of using previous design experiences is the ability of the designer to transfer knowledge learned in a past design problem to the problem that is currently being pursued. Incorporating previous knowledge into a new problem allows designers to explore different design possibilities and prevents them from having to solve the same sub-problems multiple times. If a designer encounters a part of the current design problem that resembles a previously encountered design or part of such a design, the designer has the option to incorporate parts of that previous design into the current design. This process allows designers to reuse other designs and so prevents them from going through the effort of solving the same problem twice. Alternatively, if part of a design is performing poorly or is too costly, a designer may be able to substitute other parts which perform better through an analogy process based on other designs in memory. These are only a few of the ways in which the ability to transfer knowledge between problems is potentially useful in engineering design. However, transfer processes have only been implemented in a few computational design systems.

Case-based design systems such as CADET (Sycara et al. 1991), ARCHIE-II (Domeshek and Kolodner 1991), and Kritik2 (Goel et al. 1997) all have the capability of transferring to new problems the knowledge they have stored as cases. However, only some of the systems have the capability of indexing new cases into memory so that design experience can accumulate over time (Goel et al. 1997). While these systems are able to implement a form of knowledge transfer, there are other forms of transfer in which the knowledge being transferred does not consist of an entire, previously encountered design. There are some cases where transfer of more abstract knowledge may be beneficial. In these cases, another transfer process may be required such as the chunking process added to A-Design in this study. The chunks in A-Design are more abstract in the sense that they consist of embodiments and not actual instantiated components as described below. One type of abstract process that has received some attention in previous work is analogy.

Analogy is a powerful method that may be used to transfer knowledge from previous designs as well as from other knowledge and experiences from the designer's memory. Analogies could be used to increase understanding of a novel design by allowing the designer to map previous experience onto the new device. There are many other uses for analogy in design, and there has been some work on models of design by analogy (Bhatta and Goel 1996; Howe et al. 1986; Huhns and Acosta 1988).

However, there have been fewer attempts to incorporate knowledge transfer into more search oriented design systems. These systems usually employ a form of search that takes advantage of computational power to search a large number of possible designs, and they have been based on traditional AI search techniques (Ulrich 1989; Welch and Dixon 1994), genetic algorithms (Brown and Hwang 1993), and simulated annealing (Szykman and Cagan 1995). This paper describes an attempt to modify such a design system in order to incorporate some knowledge transfer processes. This modification allows a powerful search based system to take advantage of learned design knowledge.

A-Design is a multi-agent design system based on an iterative stochastic algorithm which in many ways resembles a genetic algorithm (Campbell et al. 1999, 2000, 2003). The work presented here is an attempt to augment A-Design with a basic learning mechanism based on human cognition that allows it to transfer knowledge across design problems. The idea is to give the system the capability to extract knowledge as it solves design problems which can then be used to improve performance when solving novel problems. This process allows A-Design to work with design knowledge at an abstract level so that the design knowledge is not necessarily tied to a specific design. In addition, using this knowledge does not require a complex analogical process. This work is motivated by findings from cognitive psychology which are described next. Following this description, an introduction to A-Design is presented followed by a description of the changes to the system that allow it to learn from its design experience. The results demonstrate that A-Design was able to transfer knowledge from one design problem to another, but there are some limitations to this process as it is currently implemented. These limitations and possibilities for overcoming them are discussed along with some of the reasons for studying the cognitive basis of engineering design.

2 Cognitive basis

Experienced designers approach a design problem with a large body of potentially relevant experience and background knowledge which helps them produce better designs than designers who have little or no experience. However, the presence and amount of this experience is likely to be only one of many differences between expert and novice designers. In general, the differences between experts and novices in a domain are characterized by the amount of relevant knowledge in memory, the representation of this knowledge, and the organization of this knowledge. These are the types of differences between experts and novices that have been found in domains such as chess, physics, electronics, Go, and medicine (Chase and Simon 1973; Egan and Schwartz 1979; Larkin et al. 1980; Patel and Groen 1991; Reitman 1976; van de Wiel et al. 2000). Results of these studies indicate that the organization of knowledge in memory differs between experts and novices in a domain just as much as the amount of knowledge does. These findings can be used as initial guides when conducting psychological studies of expert/novice differences in the domain of engineering design, but they can also contribute ideas about how to incorporate cognitive principles into computational design systems.

One of the most common findings in the study of expertise is that experts have the ability to group large amounts of information into a single memory unit or chunk. This chunking process has been a finding in a number of the domains mentioned above (Chase and Simon 1973; Egan and Schwartz 1979; Reitman 1976). For example, chess masters are able to recall the pieces on a mid-game chess board almost perfectly after only five seconds of exposure, but novice chess players can only recall five to seven pieces (Chase and Simon 1973). However, this finding is not due to extraordinary memory abilities, since both types of players recall only about seven pieces from randomly generated board positions. An analysis of the extraordinary recall ability of the chess master indicates that this recall is supported by the ability to group four to six memory pieces into a single unit of memory storage called a chunk. Novices appear to be unable to construct such chunks. This chunking ability is supported by a large knowledge base of common perceptual patterns of chess positions. Experts appear to be able to recognize such common

patterns from past experience, and then the location of the pattern only has to be referred to in long-term memory when recalling the pieces. This chunking ability in a domain has been used to explain how experts are better able to perceive, analyze, and act on situations in their domain (Chase and Simon 1973).

In the study presented here, the main issue is the kinds of mechanisms that must be employed in order to allow a computational design system to demonstrate the ability to learn and utilize familiar design patterns. There is no attempt to actually model the particular cognitive processes that human designers employ in order to learn such information; the idea is rather to supply a design system with a simple chunking ability modeled on findings from studies of cognitive processes used in other domains of expertise. An agent-based design system called A-Design was modified to include new processes and agents that allow it to store chunks from past design experiences and utilize these chunks when solving new design problems. This new learning mechanism is simple in comparison to the cognitive processes utilized by humans in order to transfer knowledge to new problems, but its successes and limitations provide insight that should aid in future attempts to incorporate learning processes into computational design systems.

3 Background: A-Design

An introduction to A-Design is required in order to explain how the agents in the system produce designs, and how the system was modified in order to include the new learning processes. A detailed description of A-Design can be found in works by Campbell (2000) and Campbell et al. (1999, 2000).

Punch Press problem

Input-device: Handle	Output-device: Punch
Input-force: 6 N	Output-force goal: 100 N
Input-displacement goal: 0 m	Output-displacement goal: .25 m

Objectives: minimize cost, minimize weight

Fig. 1 The specification for the punch press problem

A-Design is an agent-based design system that produces an array of conceptual design solutions in response to a set of input and output constraints for a design problem. A design problem is specified to A-Design by stipulating the input and output constraints of the desired device in A-Design's representation. For example, a punch press could be specified to the system by indicating that the input is a downward force on a handle, and the output of the device is a much larger downward force that drives a punch into some material (Fig. 1). Along with these general input and output specifications, a number of other input/output constraints are specified such as the desire to minimize handle displacement in the punch press (shown as a goal of zero meters of displacement in Fig. 1). This problem also specifies that the punch should only be displaced by .25 m. In addition to these constraints a number of other objectives can be specified such as minimizing the cost and weight of the device. Once a problem and all of the associated objectives have been specified, A-Design's iterative design process attempts to produce a design that optimizes these design goals by working at both the configuration and component levels. Components are taken from a user-defined catalog that contains information about actual parts.

3.1 Iterative design process

A-Design's iterative design process is similar to a genetic algorithm because in each iteration a number of new design candidates are produced by the system. The best of these designs are then chosen to be the basis for the production of new designs in the next iteration. The basic structure of A-Design's iterative design process is shown in Fig. 2 along with the set of agents associated with each part of the process.

The design process begins with a set of configuration agents (C-agents) who construct candidate designs using a library of embodiments. Each C-agent adds one embodiment to an incomplete design until either the design is complete or a maximum number of embodiments have been added to the design. A

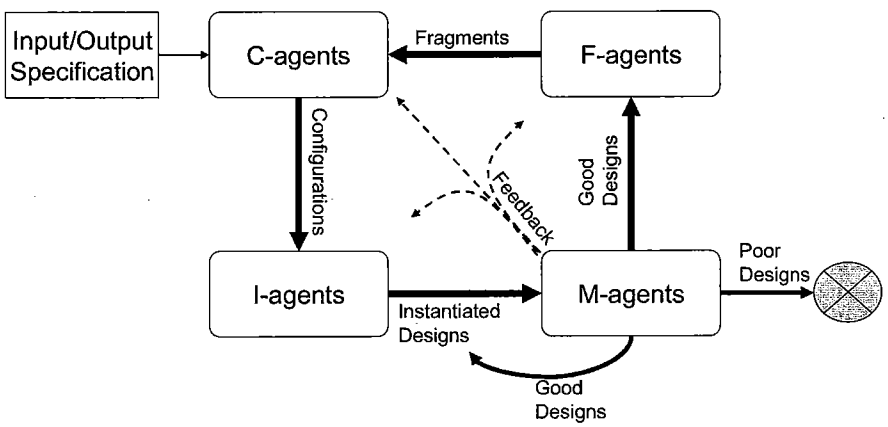


Fig. 2 A-Design's iterative design process (adapted from Campbell et al. 1999)

candidate design starts off as just a set of input and output constraints to which components can be added. These input and output constraints are represented in structures called functional parameters (FPs). An FP represents the characteristics of an interface between components, and a C-agent utilizes the qualitative information in an FP to determine which embodiments in the embodiment catalog can be connected to the incomplete design. The embodiment catalog contains information about types of components, but these components are not instantiated with actual parameter values such as length, resistance, etc. While this embodiment catalog is user defined, Chen and Brown (2002) have shown that it may be possible for A-Design to modify this catalog itself. There is also a second catalog, the component catalog, which contains a number of instantiated versions of each embodiment. This component catalog is used by the I-agents as described below. Once an embodiment is added to one of the FPs, the free ports of the embodiment are included as new FPs where future embodiments can be attached. A form of qualitative reasoning is used to update the constraints in the incomplete design as each embodiment is added. A candidate is complete once it has connected the input and output FPs and has qualitatively satisfied the input and output constraints. A C-agent chooses which embodiment to add to an incomplete design based on a set of preferences built into the agent, the current state of the incomplete design, and other influences originating from the manager agents in the system. For example, some C-agents may prefer hydraulic components while others prefer electrical ones or components connected in series over ones connected in parallel. The current state of the design can also influence which embodiment a C-agent selects. If a device should have a bounded displacement at the output such as in the punch press example in Fig. 1, then an agent would prefer components which accomplished this goal when added to the system. The C-agents' choices are also influenced by feedback they receive from manager agents in the system as discussed below. Once the set of C-agents have constructed a defined number of designs these designs are passed to a group of instantiation agents (I-agents).

I-agents take the configuration designs from C-agents and instantiate the parameters in the system with values obtained from a catalog of components. Each embodiment in the configuration has a set of parameters such as length, weight, or resistance. These agents also have a set of built in preferences. For example, some I-agents may prefer using components that are low cost while others prefer those that are low weight. Once all of the components in a design have been instantiated, equations that describe the behavior of the design can be extracted. These equations allow the design to be evaluated on how well it satisfies the constraints of the given problem. For example, the equations extracted from a punch press design are then used to evaluate how much the handle is displaced since that is one of the specified constraints. In this implementation, each device is evaluated along the dimensions specified in the problem, and these evaluations are combined into a linearly weighted sum. All of the devices are then sorted by this sum and Pareto optimal, good (based on weighting), and poor devices are separated based on this ordering. These design partitions are then passed to a set of manager agents (M-agents). For the purposes of the work in this paper, Pareto and good designs are both lumped into one category of good designs. The A-Design system still distinguishes between the two, but the chunking and memory mechanisms described below treat good and Pareto designs in the same way.

M-agents take the current design population and produce feedback that controls how other agents in the system operate. First, the agents which contribute to good and poor designs are examined and the probabilities controlling how often those agents contribute to designs are adjusted. A C-agent that contributed to a number of good designs will be called more frequently than one who contributed to many poor designs. The M-agents keep track of the number of good designs to which each C/I-agent contributes; the probability that a specific C/I-agent will be called upon in the future is a function of its past success. These statistics are used for all of the agents except for the M-agents. Managers also look for trends in the design population. Trends are groups of agents or connected embodiments that appear together in a number of designs (see example in Fig. 3). Good trends are found by examining the best designs, and bad trends are extracted from the worst designs. In this implementation, the six best designs and the six worst designs are used to extract trends. Good trends are placed on a "todo" list and bad trends on a "taboo" list. These lists work by encouraging agents to reproduce combinations of agents or embodiments that are on the todo list and discouraging agents from reproducing the groupings on the taboo list. In this manner, the todo/taboo lists allow the M-agent to influence the designs that are generated in the next iteration of the design process, but *they exert this influence only within a given run on a single problem.*

In addition to passing the good designs from the current iteration into the next iteration, all good designs are passed to fragmentation agents (F-agents), who take out one or more components of the design. These fragmented designs are then reconstructed and become part of the next iteration's design population. This fragmentation process allows good designs to propagate similar designs to the next iteration with the hope that the changes made to each design will improve it. The design population now consists of the good designs plus the newly reconstructed designs. In the next iteration the C-agents produce the number of new designs necessary to bring the design population back to the original number of designs. This design population level is a

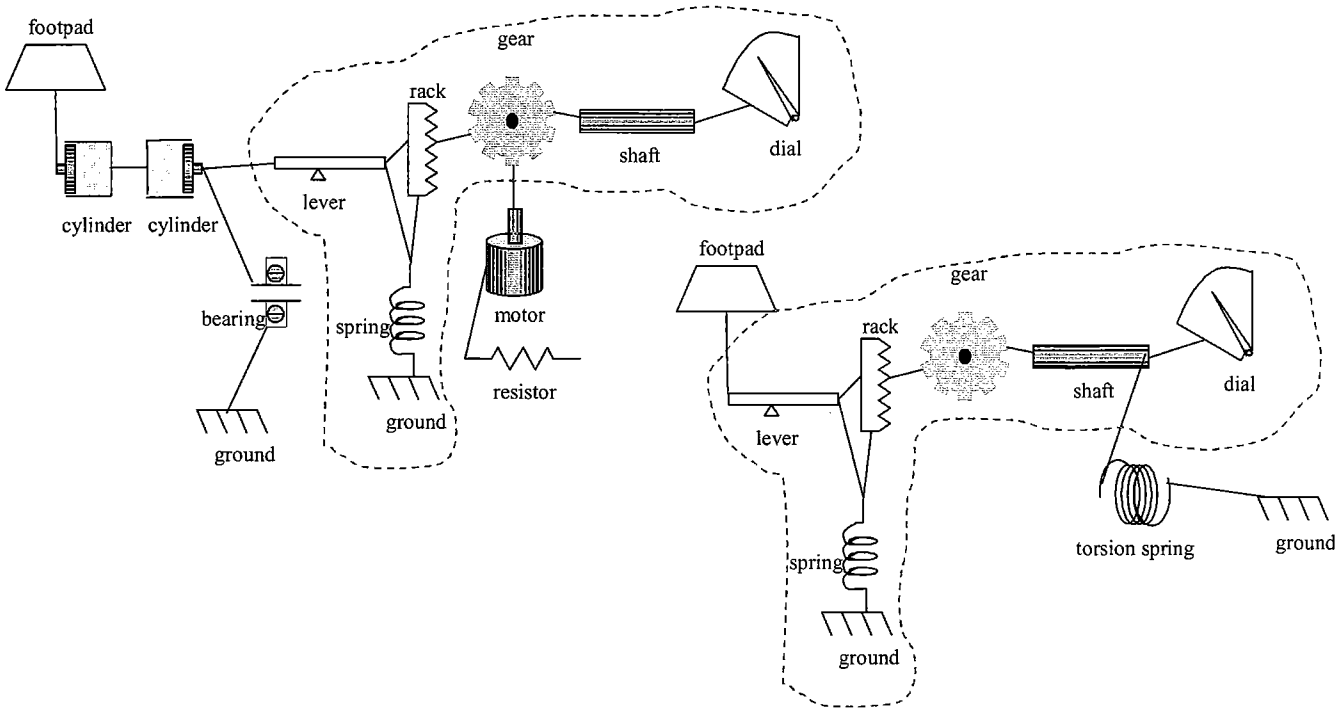


Fig. 3 An example trend found by intersecting multiple designs with the overlapping components circled

parameter of the system. This iterative process is the basis of the A-Design system.

4 Learning across problems in A-Design

In this work, we extended A-Design so that it could learn knowledge from its design experiences for application to *future* designs. In order to do this A-Design needed to be able to extract knowledge from its design problem solving activities, and once it had this knowledge there had to be some way of applying it to new design problems. One type of information that A-Design already knew how to extract was the good trends in a set of designs produced in a single iteration. The sets of interconnected embodiments that appear on the todo list are subsystems that appear in a number of good designs, and so it is likely then that these subsystems perform well in the current problem and may be worth remembering for future use.

A-Design's design process is a series of iterations in which the current designs perform the same as or better than those in the previous iteration, and so the subsystems found on the todo list in the final iteration are the ones from the best designs produced. These subsystems were chosen as the ones to be remembered in the memory store. Each subsystem appearing on this list is extracted and placed into memory as a chunk of knowledge. A subsystem from the todo list usually has many open ports to which additional embodiments can be attached. It is also desirable to be able to call one of these open ports the input to the subsystem and to call another port the output. In order to identify input and output ports, the new system randomly selects one of the designs that the subsystem appeared in before it was placed on the todo list. The embodiment closest to the input of the original design is labeled as the input to the subsystem, and the same process is completed to label one port the output. Each chunk is then placed in memory and indexed by its input and output constraints since this is the only information that is needed to determine if a component can be added to an incomplete design by the C-agents in the system.

An example chunk consisting of a connected belt and pulley is shown in Fig. 4. This chunk extraction process is only half of the knowledge transfer process. After a knowledge base of chunks has been constructed, a new set of C-agents is needed that has the ability to add chunks from memory to designs during the configuration part of the design process. These memory agents (Mem-agents) add components to incomplete designs in the same way as the C-agents except that they are adding

Design Chunk

Belt-pulley-chunk
Isa: design-chunk
Input-domain: translation
Input-interface: bolt
Output-domain: rotation
Output-interface: shaft-hole
Components: (belt pulley)
Connectivity: port-1 of component-1 is connected to port-1 of component-2

Fig. 4 An example design chunk

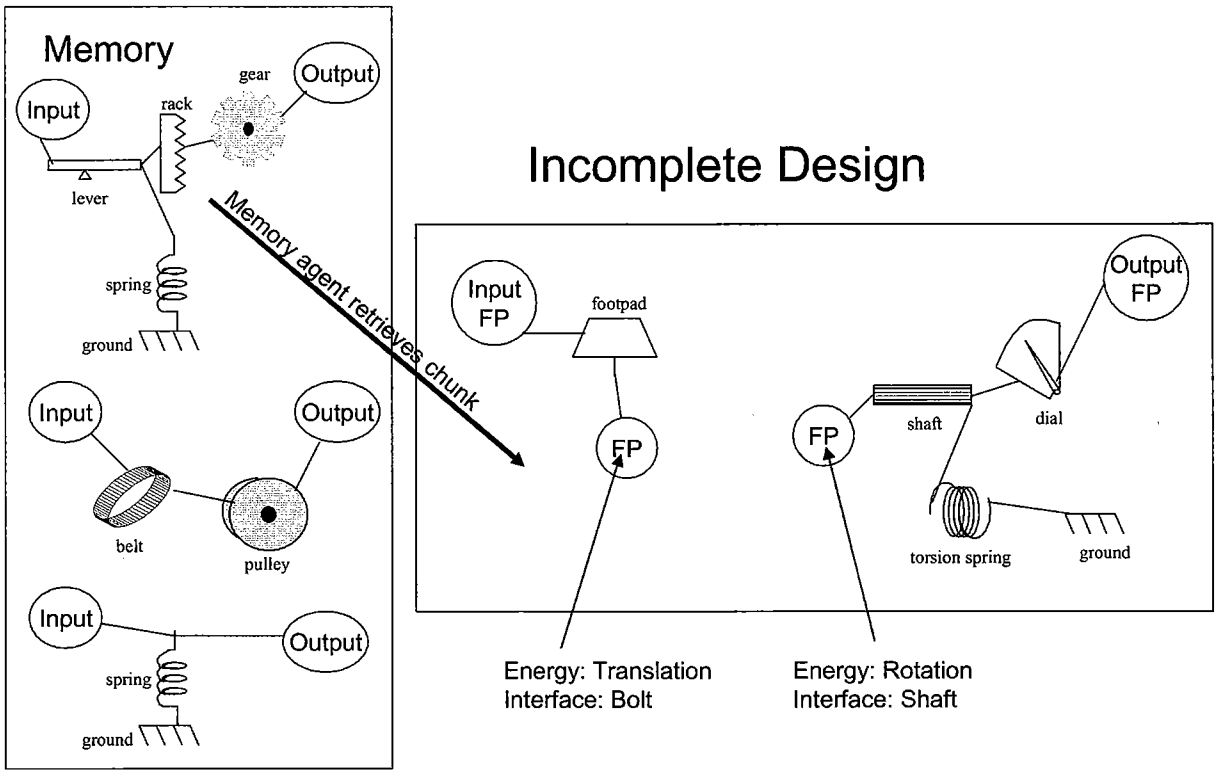


Fig. 5 Memory agents retrieve chunks from memory

chunks from memory instead of embodiments from the embodiment catalog. There is a three step process by which a chunk is added to a design (Fig. 5). First, a memory agent is called to work on an incomplete design. The memory agent then examines the open FPs in the system to determine where chunks may be added. The agent then looks in memory for a chunk that is compatible with one or more of the FPs in the design and adds this chunk. As mentioned before, all chunks are indexed in memory by their input and output constraints which are essentially the input and output FPs for the chunk. Chunks can then be retrieved from memory either based on their input, output, or both. Each of these retrieval methods is embodied in one Mem-agent. For example, the Mem-agent implementing the input retrieval strategy will search for chunks in memory whose input constraints match one of the open FPs in the design, but the input-output Mem-agent would search for chunks in memory whose input/output constraints match two open FPs, one for input and one for output.

There are some cases in which multiple chunks in memory are retrieved as potential chunks to be added to a design. In this case, a Mem-agent needs a way of determining which chunk to add. The chunk could just be randomly selected from all possible chunks, or there could be some form of learning that takes place within the memory agent that allows it to pick the chunk most likely to produce a good design. The second option was implemented and is based on learning mechanisms found in the ACT-R model of human memory (Anderson and Lebiere 1998). Each memory agent is informed about the consequences of its actions by the manager agent, and the memory agent can then use this feedback to choose among multiple chunks when it has to. When a Mem-agent adds a chunk to a design, the design can either end up being evaluated as a good design or a poor design as discussed above. The proportion of good designs that result from using a particular chunk is calculated by dividing the number of good designs the chunk was used in by the number of times the chunk was used. This proportion is calculated based on design evaluation data from the previous five iterations. When multiple chunks are retrieved, the probability that any particular one is chosen is just the proportion of good designs for that chunk divided by the sum of the proportions for all of the chunks retrieved from memory. If the proportion of good designs for a particular chunk is below some minimum level then this minimum proportion is used instead of the actual value so that every chunk has at least a small probability of being chosen regardless of its past performance. These proportions are updated dynamically after each iteration to include information about good and poor chunks from the previous five iterations. This mechanism is also similar to the strategy for move selection in the simulated annealing algorithm found in work by Hustin and Sangiovanni-Vincentelli (1987). Each of the three Mem-agents maintains a separate set of these statistics because different chunks in memory may be more suited to one of the three memory retrieval strategies outlined above.

These additions to A-Design give it the basic capabilities of noticing and storing common co-occurring

elements in chunks, noticing when a current design can be augmented with information from memory, and the ability to learn which chunks in memory perform the best in a given design problem. In general, one of these chunks serves as a way from getting from a certain type of interface and energy to another type of interface and energy. This allows for a type of abstract knowledge to be gained through the chunking of patterns of components. These basic mechanisms should allow the system to accumulate some amount of expertise in design problem solving. Similar chunking mechanisms are well documented learning processes in human cognition, and the chunking described above is similar to the way in which certain types of knowledge are acquired in the process of becoming an expert in some domains such as chess and Go (Chase and Simon 1973; Reitman 1976). There are also some similarities between A-Design's chunking of common design elements and the way in which certain types of cognitive architectures chunk together problem solving steps, e.g., SOAR (Newell 1990).

5 Testing learning

The chunk learning mechanism of A-Design was tested to see if it allowed the system to learn both within and between problems. Learning within the same problem is simply the case where A-Design works on a design problem, learns chunks, and then works on the same design problem again with the new chunks. Within problem learning should indicate whether the chunking mechanism is allowing the system to learn and apply useful knowledge about the design problem. Between problem learning occurs when A-Design applies chunks learned in one problem to a new design problem, and it is this type of knowledge transfer that the chunking mechanism was designed to accomplish.

Within problem learning was evaluated by running A-Design 20 times on one problem, and in each of these runs, a set of chunks was generated from the final design population. A-Design was then run again on the same problem 20 times, once with each of the 20 sets of learned chunks. The number of iterations in each run was 60, and the design population was set to 120 designs. A graph of A-Design's performance on a design problem can be constructed by taking the evaluation score for the best design at each iteration and averaging this evaluation across all 20 runs of the problem. This leads to a graph similar to Fig. 6 in which each line in the figure shows A-Design's average performance on one design problem. The performance difference between the initial problem and the second attempt at the problem was measured in three ways (see Fig. 6). The evaluations of the initial designs, the evaluations of the final designs, and the number of iterations until a specified evaluation level is reached can be compared. The comparison of the number of iterations can be looked at as the number of iterations that were saved by the presence of chunks and will be referred to as a measure of savings. The criterion evaluation level for each problem presented here is just the average evaluation score obtained in the final best design without using a set of stored chunks, i.e., just the performance of the A-Design system without the new learning mechanism. Any statistically significant differences in performance from the first attempt to the second can then be attributed to the chunks that were used in the second attempt.

This measure of savings was calculated by recording the iteration at which each run of A-Design reached the criterion level. If an individual run did not reach the criterion level by the end of 60 iterations, then the number of iterations to criterion was just taken to be 60 for that run. The average number of iterations for a given condition can then be calculated by averaging the number of iterations to criterion for each of the 20 runs in a condition. The average number of iterations to criterion can then be compared for any two conditions (i.e. for chunk and no chunk conditions). This yields a conservative estimate of the amount of savings since each run is capped at 60 iterations even if it would have taken longer to reach criterion.

Another potential metric that should be kept in mind is the actual running time for A-Design to finish a set of 60 iterations as outlined above. In general, the Mem-agents are computationally more expensive than normal C-agents. It is useful to determine if these extra costs

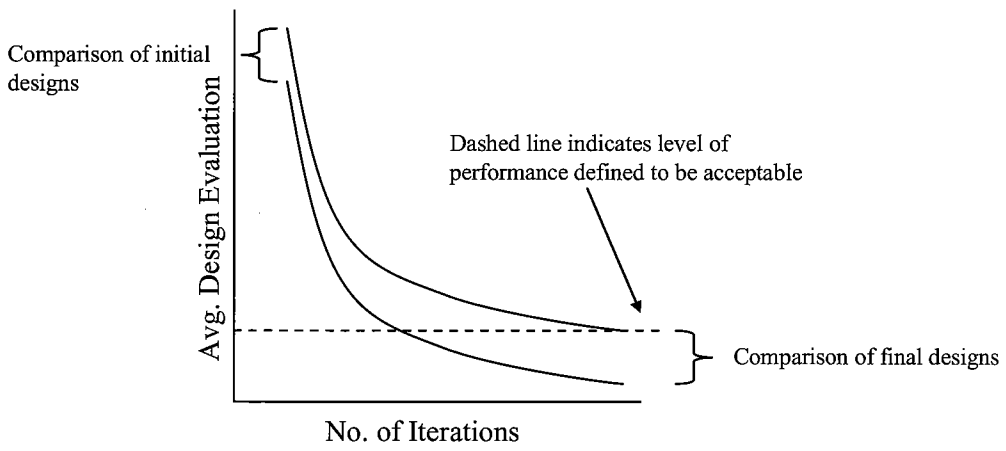


Fig. 6 An illustration of the three measures of performance

Fig. 7 An example evaluation function used for the pressure gauge

$$Evaluation = w_1 \, cost + w_2 \, mass + w_3 \, efficiency + w_4 \, accuracy$$
$$w_1 = 1, \; w_2 = 10, \; w_3 = 10, \; w_4 = 100$$

significantly impact the running time. Run time results were examined for all three problems, and it was found that the average time for the completion of one iteration did not differ significantly when Mem-agents were used. Therefore any benefits of the chunking processes do not come at the expense of significantly increased run time.

5.1 Design problems

Three electromechanical design problems were used to assess the benefits of the chunking mechanism: a punch press, a pressure gauge, and a weighing machine. The punch press problem is the same as described above, a handle is pulled which forces a punch through some material at the output. Punch presses are evaluated based on their cost, the amount of input handle displacement, and how closely they conform to the specified output displacement and force. The pressure gauge problem has an input pressure source and the output is a dial display that reflects the amount of pressure coming from the pressure source, and this problem is evaluated on the cost, mass, efficiency, and dial accuracy of the gauge. The weighing machine as defined by Campbell et al. (1999) takes a force input on a footpad and has a dial output, and is evaluated on the cost, mass, dial accuracy, and input displacement of the device. An example of the linearly weighted evaluation function is presented in Fig. 7. The evaluation function deals with real quantities like dollars, radians, and grams so the scales are difficult to interpret when combined with the weights. The relative weighting is the most important aspect of the evaluation function for our purposes. For example, the weights presented in Fig. 7 emphasize dial accuracy while placing little emphasis on cost. In this equation, characteristics like cost can be read from the cost of the components, but to evaluate accuracy a behavioral equation must be extracted from the design so that accuracy can be calculated. Some example designs produced by A-Design for these problems are presented in Fig. 8. These example designs are typical of the kinds of solutions A-Design produces for each problem. The component catalog used in this research is the same as the one that has been presented in previous A-Design work (Campbell 2000; Campbell et al. 1999, 2000). The only additions to the catalog were the input and output components for the new problems, such as the handle and punch for the punch press problem.

These specific problems were utilized to provide both a problem that was similar to the weighing machine as well as a problem that was significantly different from the weighing machine The pressure gauge is a measurement device with a dial output just like the weighing machine. However, the goal of the punch press is to amplify the small input force so that it is sufficient to drive a punch through some material. It was thought that this problem shares little with the weighing machine so it was used to evaluate how context dependent A-Design's new knowledge capabilities would be. For example, if A-Design learned design chunks from the weighing machine problem, then these chunks might be easier to apply in the pressure gauge problem since this problem is similar to the weighing machine. On the other hand, applying these same design chunks to the punch press might be more difficult and potentially less useful because the punch press does not have as much in common with the weighing machine. This means that the similarity between the devices probably has an impact on which learned chunks are transferred successfully.

The specific evaluation function used to evaluate a device does have an impact on the chunks learned from that design experience. For example, a pressure gauge design that minimizes cost above all else would be different from a design which primarily emphasizes the accuracy of the gauge. These different designs would lead to different chunks, and it may be that these chunks will transfer best to designs that have a similar emphasis in their evaluation functions. This aspect of chunking is not explored in the current work. Our primary goal was to evaluate the viability of the simple chunking mechanism described above. We tried to keep the emphasis in the evaluation functions as constant as possible. For example, both the weighing machine and the pressure gauge evaluations emphasize accuracy more than any other criteria.

Two sample chunks are shown in Fig. 9. Figure 9a shows a chunk learned by A-Design while solving the weighing machine problem. This chunk was then used in the production of pressure gauge designs, and in particular, this chunk was used in the design of the pressure gauge shown in Fig. 8. Figure 9b is another example of a chunk learned in the weighing machine problem, and it was used in the production of the punch press design seen in Fig. 8. The first chunk is larger as it contains seven components as opposed to the smaller chunk composed of two components. Larger chunks were more frequently transferred between the weighing machine and pressure gauge problems than between the punch press and other problems. This finding was not unexpected and is due to the degree of similarity among the problems as noted above.

6 Results

6.1 Within problem results

Within problem results for the three design problems can be seen in Figs. 10, 11 and 12 and Tables 1, 2 and 3.

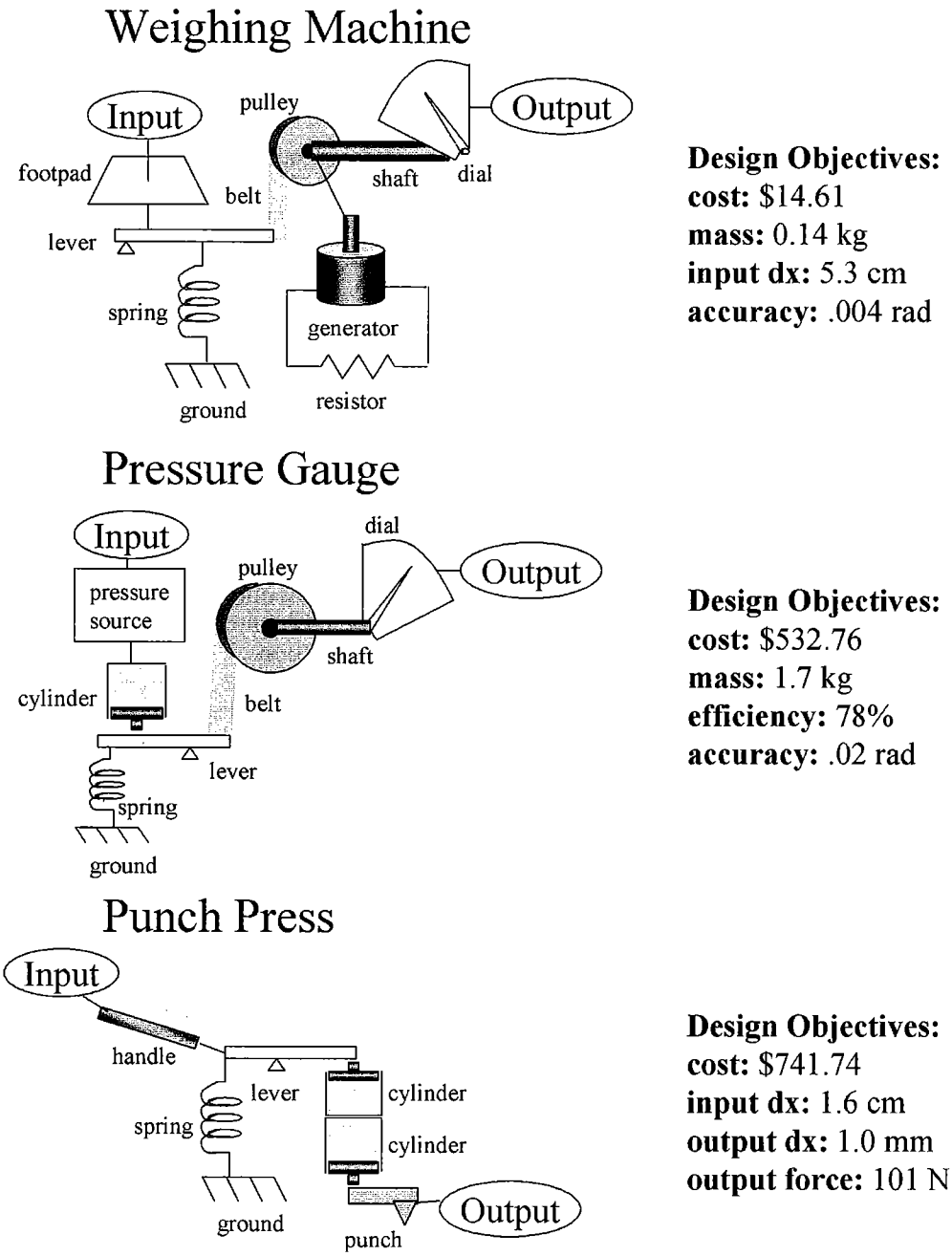


Fig. 8 Sample solutions produced by A-Design for each of the three design problems

There appears to be within problem learning in each of the three problems (lower evaluation scores are better). A series of paired t-tests was run to assess the statistical significance of any differences in the three performance measures discussed above. In some cases the assumptions of the t-test did not hold (such as equal variances). In these cases nonparametric statistical tests were also run, but the results from these tests did not differ from those of the t-tests.

6.1.1 Weighing machine

In the weighing machine problem only the comparison of the initial designs was not significantly different, $t(19) = 1.19$, $p = .12$, but both the comparison of the last designs, $t(19) = 2.47$, $p = .01$, and the amount of savings, $t(19) = 3.73$, $p < .001$, were significant. On average, the final design produced without chunks had an evaluation of 21.7 as compared to 19.3 with chunks, and the number of iterations required to reach an evaluation of 21.7 was 45.6 without chunks and 30.7 with chunks, namely, 33% fewer iterations.

6.1.2 Pressure gauge

In the pressure gauge problem the comparison of initial designs, $t(19) = 2.14$, $p = .02$, and savings, $t(19) = 1.82$, $p = .04$, were significant, but the comparison of final

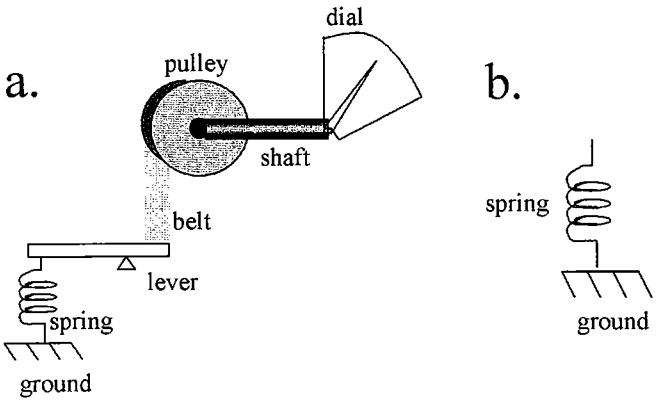


Fig. 9 Sample chunks learned from the weighing machine problem and used in the pressure gauge and punch press problems in Fig. 7

design evaluations was not, t (19) = 1.19, p = .12. On average, the initial design produced without any chunks was 50,302 as compared to 35,358 with chunks, and the number of iterations required to reach an evaluation of 3,258 was 44 without chunks and 32.7 with chunks, namely, 26% fewer iterations.

6.1.3 *Punch press*

In the punch press problem the comparison of initial designs, t (19) = 3.32, p = .002, final designs t (19) = 3.34, p = .002, and savings, t (19) = 2.73, p = .007, were significant. On average, the initial design evaluation produced without any chunks was 9,046 as compared to 6,675 with chunks, the final design produced without any chunks was 1,658 compared to 1,016 with chunks, and the number of iterations required to reach an evaluation of 1,658 was 38.9 without chunks and 23 with chunks, namely, 41% fewer iterations.

6.2 Between problem results

The results for between problem transfer can be seen in Figs. 13, 14 and 15 and Tables 4, 5 and 6. For each problem, a one-way ANOVA was run for each comparison followed by a series of planned contrasts if the ANOVA indicated any significant differences. In some cases the assumptions of the ANOVA did not hold (such as equal variances). In these cases nonparametric statistical tests were also run, but the results from these tests did not differ from those of the ANOVAs.

6.2.1 *Weighing machine*

Figure 13 indicates that the three conditions perform about the same, and there were no significant differences in initial evaluations, F (2,57) = .74, p = .24, final evaluations, F (2,57) = .06, p = .47, or savings, F (2,57) = .804, p = .23, for the weighing machine problem. Thus neither the pressure gauge nor the punch press chunks produced significant improvements in the weighing machine problem. In this case, the within problem transfer results outperformed all cases of between problem transfer.

6.2.2 *Pressure gauge*

Figure 14 indicates that the two conditions with chunks perform better than solving the problem with no previous knowledge. There were significant differences in final evaluations, F (2,57) = 2.94, p = .03, and savings,

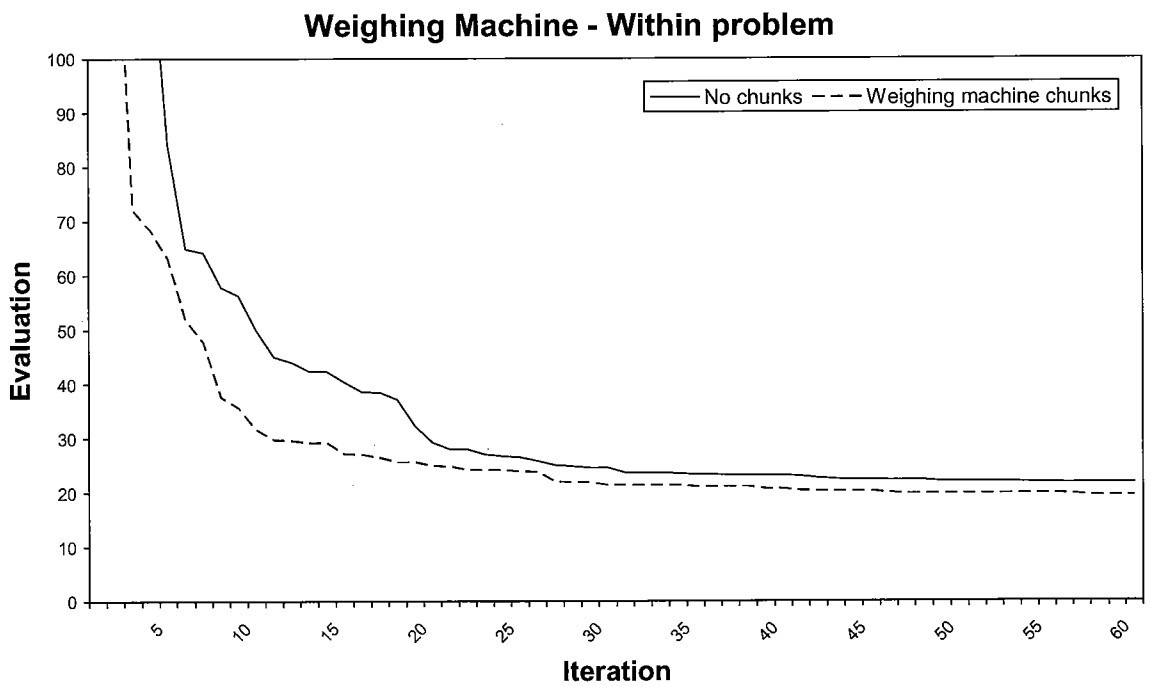


Fig. 10 Within problem results for the weighing machine problem

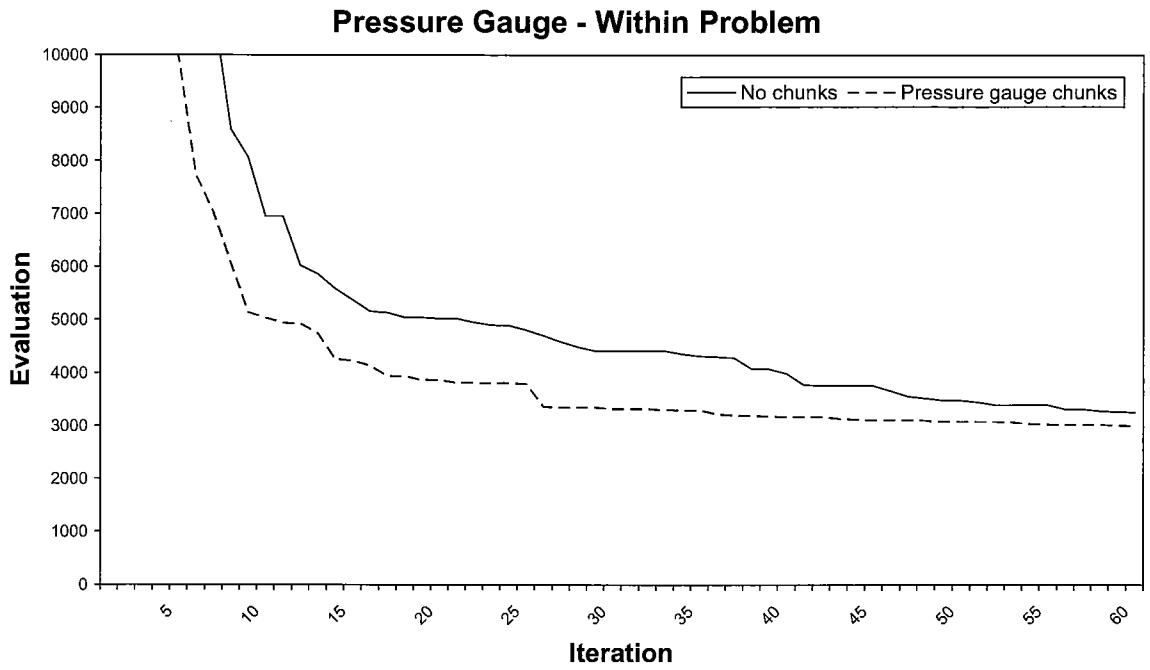


Fig. 11 Within problem results for the pressure gauge problem

$F(2,57) = 4.53$, $p = .008$, but not for the initial evaluations, $F(2,57) = .9$, $p = .21$. A series of contrasts reveal that chunks from the weighing machine problem allow the system to produce better designs than the no chunk condition, and both the weighing machine and punch press chunk conditions produce significant savings when compared to the no chunk condition. Final design evaluations in the weighing machine chunk condition had an average of 2,766 while the average in the no chunk condition was 3,258. Chunks from the weighing machine problem help the system to reach an evaluation of 3,258 in 28.8 iterations as compared to 44 iterations for the no chunk condition, and chunks from the punch press problem also allow the system to reach the criterion evaluation in 29 iterations, 34–35% fewer iterations in both cases. It can also be seen that within problem transfer produced similar quality final designs and reached criterion in a similar number of iterations as did the two between problem transfer conditions (Figs. 11 and 14). In fact, the two between problem transfer cases appear to outperform the within problem transfer, but these differences were not statistically significant.

Fig. 12 Within problem results for the punch press problem

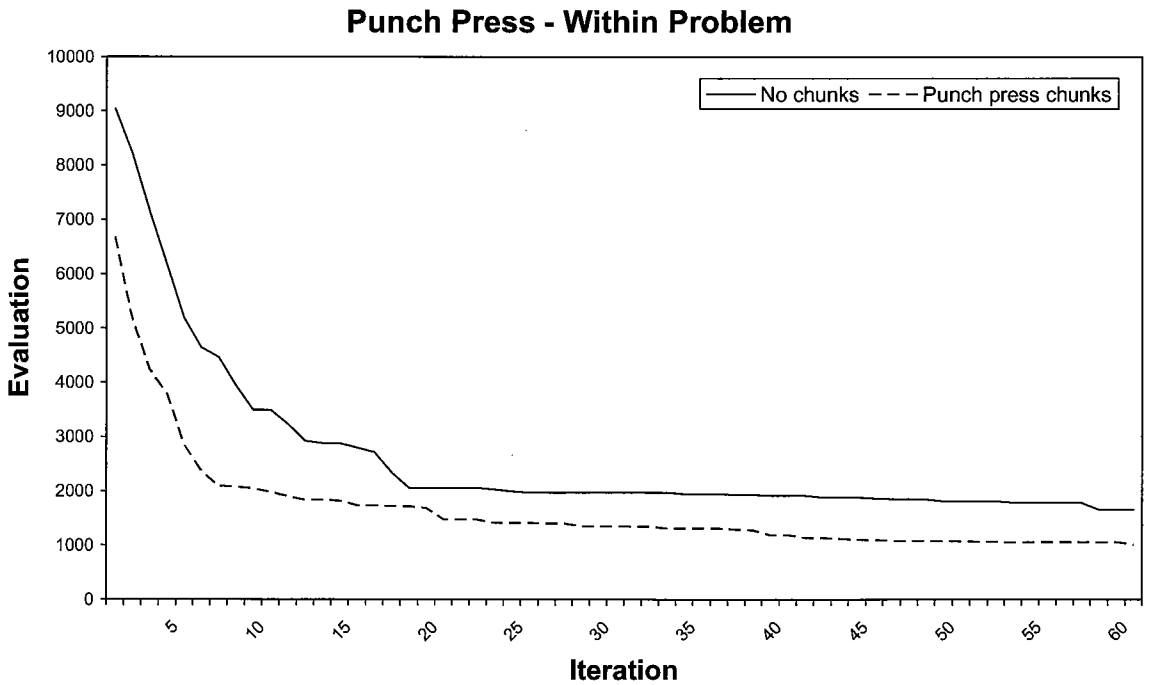


Table 1 Within problem performance measures for the weighing machine

	Initial evaluations		Final evaluations		Iterations to criterion	
	μ	σ	μ	σ	μ	σ
No chunks	773	1001	21.7	4.90	45.6	16.0
Using within problem chunks	465	632	19.3	3.29	30.7	20.3

Table 2 Within problem performance measures for the pressure gauge

	Initial evaluations		Final evaluations		Iterations to criterion	
	μ	σ	μ	σ	μ	σ
No chunks	50,302	25,775	3,258	807	44.0	18.0
Using within problem chunks	35,358	26,275	3,005	643	32.7	23.5

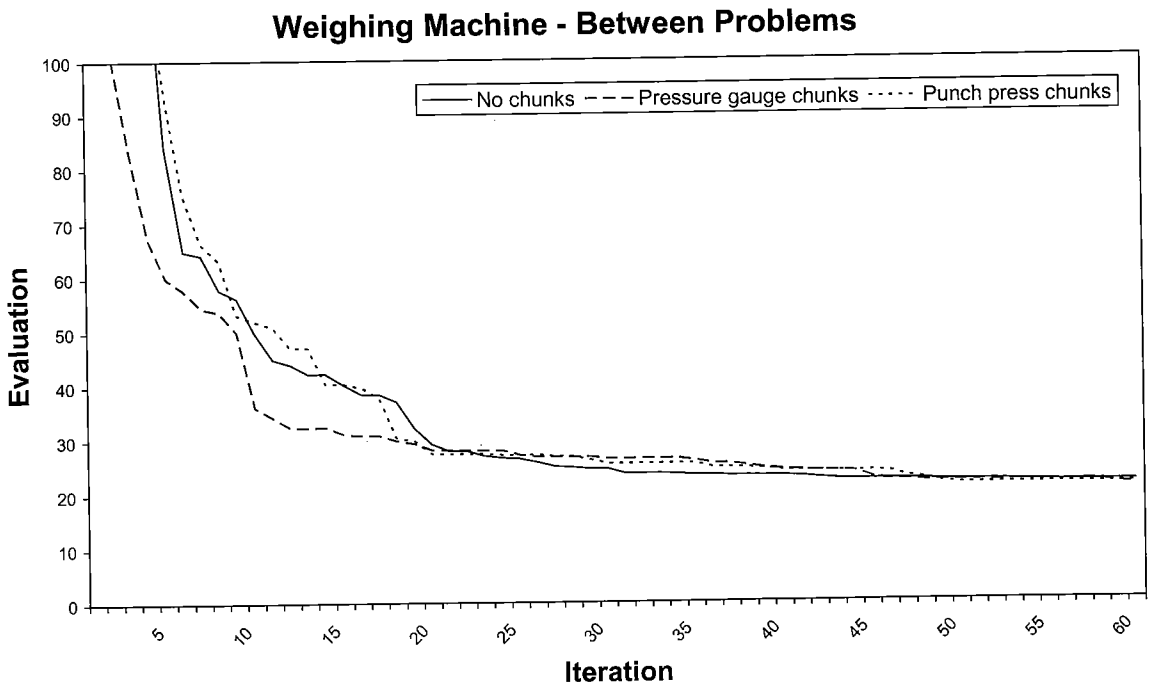
Table 3 Within problem performance measures for the punch press

	Initial evaluations		Final evaluations		Iterations to criterion	
	μ	σ	μ	σ	μ	σ
No chunks	9,046	1,327	1,658	524	38.9	21.4
Using within problem chunks	6,675	2,762	1,016	517	23.0	18.7

6.2.3 Punch press

Figure 15 indicates that the two conditions with chunks perform better than solving the problem with no previous knowledge. There were significant differences in final evaluations, F (2,57) = 4.09, p = .01, but not for the initial evaluations, F (2,57) = .65, p = .26, or savings, F (2,57) = 1.37, p = .13. A series of contrasts reveal that both of the chunk conditions produce significantly better final designs when compared to the no chunk condition. Final design evaluations in the weighing machine chunk condition had an average of 1,298 and in the pressure gauge chunk condition the average was 1,271, while the average in the no chunk condition was 1,658. The within problem transfer case (Fig. 12) outperformed between problem transfer in this problem.

Fig. 13 Between problem results for the weighing machine problem



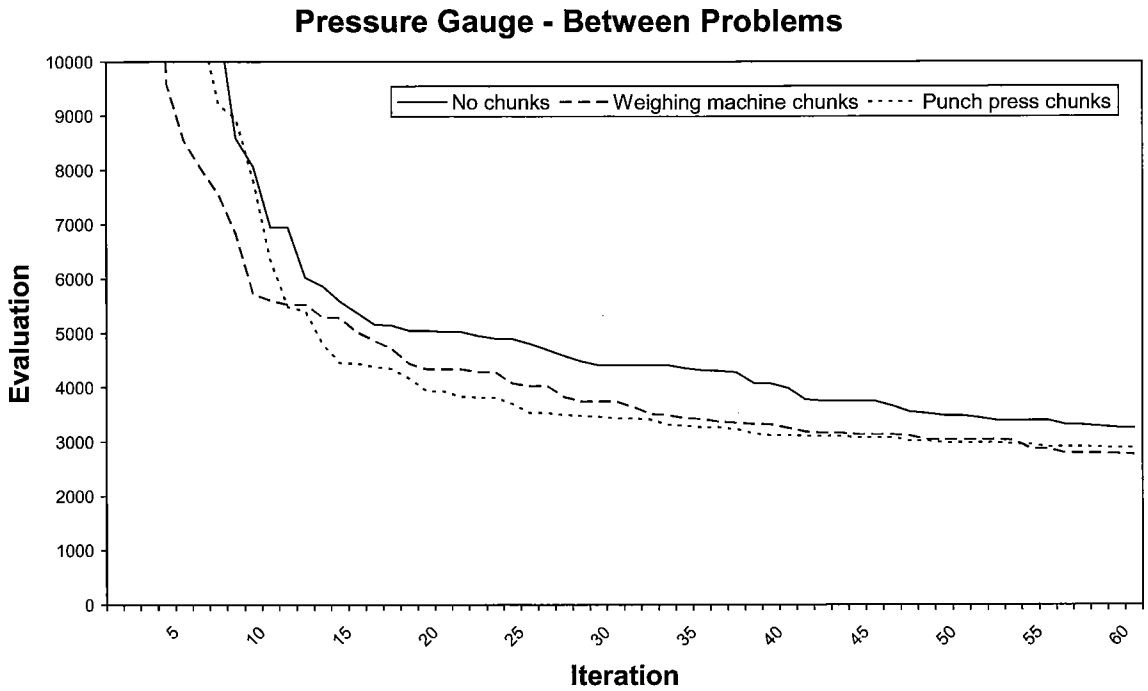


Fig. 14 Between problem results for the pressure gauge problem

7 Discussion

The chunking mechanism did produce successful knowledge transfer in most cases. Transfer was found in most cases where it was expected, but interestingly there was no significant between problem transfer when working on the weighing machine problem. This was unexpected because of the similarity of the weighing machine and pressure gauge problems, and because there was transfer from the weighing machine problem to the pressure gauge problem. Performance on the punch press problem improved regardless of the types of chunks used which may indicate some unexpected similarities between the punch press and the other problems or the existence of some chunks that work regardless of the problem context in which they were learned. In general, within problem transfer was the most effective

Fig. 15 Between problem results for the punch press problem

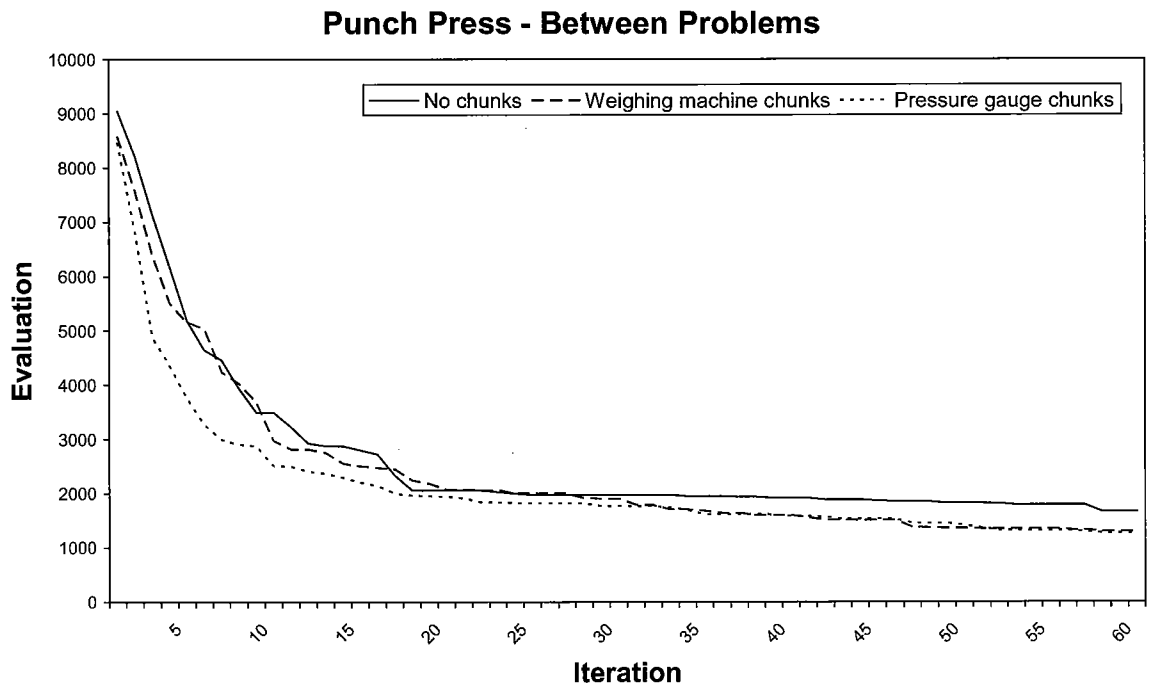


Table 4 Between problem performance measures for the weighing machine

	Initial evaluations		Final evaluations		Iterations to criterion	
	μ	σ	μ	σ	μ	σ
No chunks	773	1,001	21.7	4.90	45.6	16.0
Using pressure gauge chunks	700	1,397	21.2	4.21	49.5	12.6
Using punch press chunks	405	352	21.4	4.73	43.2	18.4

Table 5 Between problem performance measures for the pressure gauge

	Initial evaluations		Final evaluations		Iterations to criterion	
	μ	σ	μ	σ	μ	σ
No chunks	50,302	25,775	3,258	807	44.0	18.0
Using weighing machine chunks	42,190	23,483	2,766	441	28.8	18.7
Using punch press chunks	52,670	2,8337	2,889	705	28.9	18.4

Table 6 Between problem performance measures for the punch press

	Initial evaluations		Final evaluations		Iterations to criterion	
	μ	σ	μ	σ	μ	σ
No chunks	9,046	1,327	1,658	524	38.9	21.4
Using weighing machine chunks	8,583	1,493	1,298	413	28.7	20.0
Using pressure gauge chunks	8,472	2,139	1,271	491	30.7	20.9

type of transfer as none of the between problem cases significantly outperformed the within problem case in any of the problems. Knowledge transfer within the same problem produced improvements in two to three of the performance measures in each problem. In the cases where significant between problem transfer did occur, the performance measures that improved were the final design evaluations and the savings measure. So while the knowledge from another problem did not produce better initial designs in any of the three problems, it was able to improve performance throughout the iterative design process.

This eventual but not immediate improvement due to transfer could indicate that the chunks learned by the system could only be applied to a new problem at some late or intermediate stage of the design process. Another explanation is that only some of the chunks learned in another problem are beneficial to the current problem, and it takes many attempts to find which of the chunks are beneficial when applied to the current design problem. An analysis of the data produced by the system indicates that certain chunks did come to be preferred by the system in a certain problem, and this was mostly due to the success statistics that are accumulated by Mem-agents as discussed previously. It is also possible that some of the chunks had to be slightly modified in order to produce improvements in a particular problem, and this modification process did appear to operate as discussed below.

It was hypothesized that most of the between problem transfer would occur between the weighing machine and pressure gauge problems due to their similarity. In the pressure gauge problem, the within problem chunks lead to the most improvement followed by the chunks from the weighing machine problem and finally the punch press chunks. This graded transfer makes sense as it is easiest for the system to apply knowledge from the same problem and somewhat more difficult to apply knowledge from a similar problem. However, in the weighing machine problem there was no significant between problem transfer at all. This behavior appears to be caused by the embodiments that occur in the chunks learned from the pressure gauge problem. Every pressure gauge takes some pressure source as its input, and, given the catalog that was supplied, the only way A-Design has of transforming this pressure source into a translational motion is by using a hydraulic cylinder. So all pressure gauges have cylinders in them, and since chunks are found by extracting commonalities, a large portion of the chunks learned by A-Design in this problem have the cylinder embodiment in them. None of the best weighing machine designs produced had cylinders in them because of the high cost of this embodiment, and so these pressure gauge chunks do not appear to be very useful in the weighing machine problem. This seems like the most likely cause of the asymmetric transfer.

The results from the punch press problem were also a little surprising. It was originally thought that this problem had few similarities to either the weighing machine or pressure gauge problems, but in both cases transfer to the punch press improved results. One