Artificial Intelligence Review 7, 43–70, 1993.

Creativity: a survey of AI approaches

JON ROWE AND DEREK PARTRIDGE

Department of Computer Science, University of Exeter, Exeter, EX4 4PT, U.K.

Abstract. In this paper we critically survey the AI programs that have been developed to exhibit some aspect of creative behaviour. We describe five necessary characteristics of models of creativity, and we apply these characteristics to help assess the programs surveyed. These characteristic features also provide a basis for a new theory of creative behavior: an emergent memory model. The survey is concluded with an assessment of an implementation of this latest theory.

Bill sings to Sarah. Sarah sings to Bill. Perhaps they will do other dangerous things together. They may eat lamb or stroke each other. They may chant of their difficulties and their happiness. They have love but they also have typewriters. That is interesting.

1. INTRODUCTION: RANDOMNESS AND RIGIDITY

Intelligence involves creative behaviour. Few would challenge this statement, but agreement on what is meant by "creative behaviour" would be much harder to find especially if the meaning had to be couched at the level of programmable specifics. Nevertheless, the historical record in AI is dotted, albeit sparsely, with attempts to build creative programs. But it should come as no surprise that these programs tend to differ greatly in both what they attempt to do (to exhibit creativity), and how they attempt to do it. In this paper we shall critically survey a wide range of these programs designed to exhibit creative behavior.

The opening quotation was written by the computer program Racter which, in turn, was written by William Chamberlain (Racter 1984). Racter works by constructing sentences, poems and stories according to "syntax directives" using random words and phrases. This is modified by an ability to adapt previously used phrases to give a sense of continuity. The effect of reading Racter's compositions is very interesting. Initially, they are too bizarre to be taken seriously. After a while, though, the general surrealism becomes appealing and some parts quite brilliant: "reflections are images of tarnished aspirations." This effect, however, is also short-lived and soon gives way to dissatisfaction and eventual boredom. What is missing? If creativity is about producing something new then Racter should certainly qualify, but it seems that novelty is not enough. The impact of Racter's prose can best be described as superficial. An interesting comparison here is with some modern art. This has the same kind of shock value at first sight but, when this has worn off, it is replaced by a longlasting appreciation. It bears examination and re-examination. It is easily detected that Racter's work has no semantic relevance. Any interest it does provoke is almost entirely due to the work the reader's mind does on the words that Racter produces. This is aided slightly by Racter's habit of repeating words and phrases. The reader, though, becomes hard-pushed to construct any detailed model of the narrative. Randomness, then, is not enough, even though it (trivially) generates flexible and ambiguous behaviour. Because there are no semantics behind the output, there is no way for Racter to avoid useless products or to assess what is produced. If randomness is a necessary element of creativity, it must operate much more subtly than this.

Another early program that generated stories was Meehan's TALE-SPIN (Meehan 1977). The theory here was that a story is about a problem and how it gets solved. Characters were set up with different goals which they tried to meet by forming plans and taking appropriate actions. The story was a natural language trace of the events produced by this process. This method is almost a directly opposite approach to that of Racter. The program is full of knowledge about the world in which the characters move, about their goals and their social relationships. The problems were solved using a planner. Thus, for the most part, only sensible stories were produced. However, much flexibility was sacrificed, not so much with the events that could occur (although these were also limited) but with the structure of the stories that could be written. They all followed from how a top-down, subgoaling recursive planner operates. Representation of knowledge (about stories) was fixed. The only ambiguities that arose were in "mis-spun" stories (that is, those that came from bugs in the system). Clearly there is a need for both flexibility and control. Finding a way to combine these properties has presented many problems for AI programs.

2. FIVE CHARACTERISTICS

Our study of the history of creativity in AI, together with a survey of psychological theories of human creative behaviour, has led to a formulation of five necessary characteristics of computational creativity. These five will be used as a basis for the critical survey. The characteristics are as follows.

Firstly, it is necessary that knowledge is organised in such a way that the number of possible associations (the creative potential) is maximized. Thus programs require a *flexible* knowledge representation scheme. Any individual concept should be related to as many others as possible with only small variations in the relative strengths of associations. If two concepts are much more strongly related than any others then they will always be activated together; a rigid, uncreative response. Rather than using a spreading activation technique (which would be computationally expensive on a sequential machine), a probabilistic selection method could give the same behaviour, statistically speaking, in the long run. Creative programs should allow for this kind of behavior.

Secondly, it is necessary to tolerate *ambiguity* in representations. Thus programs should, under appropriate circumstances, allow the generation of seemingly incorrect associations in order to build connections between concepts. In other words, in should be possible to relax constraints on concepts to enable the activation of more disparate ideas. The control of constraint relaxation needs analysis. The decision on which constraints to relax is a hard one, and similarly difficult is the problem of when to do it, and when the constraints need to be tightened up again. This process is analogous to the regression stage of the creative process, and thus has both external and internal influences.

Thirdly, there is a need for multiple representations. A single concept should not just be applicable in a single situation. Rather, a concept should be indexed to many situations. This would help avoid the problem of *functional fixity*, where people have a fixed idea as to the use of an object (or concept) and this prevents it from being applied elsewhere. It has been shown that creative people are less prone to functional fixity than uncreative people (Duncker 1945).

Fourthly, the usefulness of new combinations should be *assessable*. It is of no use of create many combinations if they are all useless. It is too computationally expensive to generate many combinations and then try to filter out the good ones (see, for example, the later discussion on AM and Eurisko). Rather it should be generally expected that most new combinations will be worthwhile and only in a small number of cases need adjustment or rejection.

Lastly, any new combinations need to be *elaboratable* to find out their consequences. It may be expected that this would involve more logical problem solving techniques (analogous to secondary processing). This process of verification should assess whether or not the new idea has been successful or if more combinations (primary processing) need to be made. The consequences discovered should be applied to the problem situation; the usefulness of a discovery is decided by its applicability.

3. GENERATIVE GRAMMARS

A number of programs that attempt to be creative have been based on generative grammars. A grammar consists of a set of production rules that re-write strings of symbols. Symbols that can be re-written are called non-terminals; those which cannot are terminals. Starting with a distinguished non-terminal, the rules are applied until a string of terminals is obtained. The set of all producible strings is called the "language" defined by the grammar.

For example, Rumelhart (Rumelhart 1975) has written a story grammar with such rules as:

story → setting, episode
setting → time, place, characters
episode → event, reaction

and so on. The terminals are the actual words and phrases that make up the story. The question to ask here is: what is the relation of the language defined by this grammar to the set of all stories? In other words: how plausible is this system as a story writer? Unsurprisingly, the system does not seem to write good stories. A great limitation is the scope of known words and phrases; it is small and fixed. Worse, as with TALE-SPIN, the known form of story structure is rigid. It does, however, incorporate a random element into the generation, in the choice of phrases used to fit the sturcture. But this flexibility seems to be entirely on the wrong level. It is knowledge about what constitutes a story that needs to be flexible.

This kind of criticism applies, in general, to other grammar systems. One such system, a jazz bass improviser, was written by Johnson-Laird who claims that randomness is essential to overcome the deterministic nature of computer programs (Johnson-Laird 1987). However, this view misunderstands the nature of determinism. It is quite possible that a deterministic system be unpredictable (say, if the system is a non-linear dynamical one). Behaviour can appear random without having to posit an explicit random source. More usefully, Johnson-Laird looks at three classes of creative algorithm: neo-Darwinian, neo-Lamarckian and multi-stage. Neo-Darwinian creativity is when all possible combinations are made and then filtered. Neo-Lamarckian makes only those combinations that satisfy the constraints and then arbitrarily selects an output. As usual in such cases, the preferred solution, multi-stage, lies somewhere between the two. Neo-Darwinianism is clearly a suppressor oriented system whereas neo-Lamarckian creativity employs censoring. Multi-stage uses partial censoring and suppression in turn.

The limitation of grammar-based systems is in the rigidity of the rules. Narayanan suggests that a grammar could be given an "open" re-write production as a sort of *carte blanche* (Narayanan 1983). This would allow the introduction of any new string at a suitable time which could be checked for consistency and applicability. If it passes these tests, it can be added as an axiom. Though this does provide a means for accumulating and adapting rules, Narayanan does not give any criteria by which a suitable string may be chosen. A purely random choice would almost certainly produce rubbish. Again we have the issue of control versus randomness.

Both Rumelhart's and Johnson-Laird's programs produce orginal output, but, as it is not very domain plausible, their novelty is not worth much (one wonders why Johnson-Laird chose modern jazz as a domain). They have a similar shock value to the productions of Racter. Constraining the domain by tightening the possible non-terminals could help. Steedman has produced a grammar for jazz chord sequences based entirely on the 12 bar blues (Steedman 1984). Here the results are, apparently, more acceptable but the originality of the output suffers.

In summary, grammar systems are too restrictive and inserting a random element does not seem to be an appropriate solution as it breaks down the essential control.

4. DISCOVERY PROGRAMS

A classic program in the field of computational creativity is AM, a program that discovers mathematical concepts (Davis and Lenat 1982). AM starts with an initial set of concepts arranged in a specialisation hierarchy. Each concept is represented by a set of slots containing information such as: definition, examples, domain and range (for operations), specialisations, worth and so on. An example of a concept can also be a concept, for example, the concept *multiply* is an example of the concept *number-function.* To start with, many of the slots are empty and AM, guided by heuristics, attempts to fill them. There are four types of heuristic: fill, check, suggest and interest. Fill rules attempt to fill in a slot. For instance, to fill in examples of a concept, one rule might look for functions which have that concept as the range and run them over their respective domains. Check rules make sure entries are correct and also watch regularities. They may have side-effects such as creating new concepts or proposing new tasks. Suggest rules are purely side-effects and are useful when the program is running low on interesting tasks. Lastly, interest rules are used to gauge the worth of a concept. The program is controlled by an agenda which maintains a list of tasks sorted according to the worth of the concepts involved, the number of reasons for a task's suggestion and their worths and also the type of the task. The top task is chosen, relevant rules are gathered and operated, possibly with side-effects such as new concepts and new tasks.

AM's initial concepts included sets, lists, ordered pairs, some functions such as union, intersect and some more general operations such as compose, coalesce and canonise. On a "good run", AM discovers natural numbers, addition, multiplication, primes, prime factorisation and Goldbach's conjecture. It is important to note that AM never proved any of its results; it made conjectures based on the examples which it generated. AM came to a halt soon after Goldbach's conjecture, failing to produce any more new, interesting concepts and eventually proposing only boring tasks. Lenat conjectured that this happened because of the nature of the heuristics. These were applicable to general concepts such as sets but less powerful with more specialised concepts such as numbers. By the time AM had reached prime numbers the heuristics were virtually useless and pottered about making small and ineffectual conjectures. Though it discovered integer division, it never found the rationals. It knew about perfect squares and prime factorisation but didn't conjecture that the square root of a non-perfect square isn't rational. In fact, it didn't consider looking at such things, let alone conjecture about them.

The solution, as Lenat saw it, was to automatically synthesise new heuristics to go with the new concepts. Lenat attempted this by allowing AM to work on its own heuristics as well as on maths concepts. The results were disastrous. A vast number of completely useless heuristics were created. What had gone wrong? Lenat re-examined the method by which new concepts were made (Lenat 1983). Many heuristics generated new concepts by syntactically manipulating the definitions of old concepts. When these were definitions of maths concepts they were implemented as small pieces of Lisp code. A minor modification to the Lisp code produced a (usually) meaningful result. This was attributed, by Lenat, to the close correspondence between lisp and elementary mathematics. However, when similar modifications were made to the lisp code representing a heuristic, the results were virtually meaningless. This is because there is no close link between lisp and heuristics; heuristics operate at a much higher-level than lisp, and many lines of lisp code are required to implement each heuristic. What was needed, according to Lenat, was a new representation language that did have such a link. Then, syntactic mutations would correspond to meaningful semantic changes. In a new program, Eurisko (Lenat 1982), heuristics are represented as frames with many slots, each slot a small piece of lisp. Now mutations in the values of these slots produce a meaningful change in the heuristic. New slots could be added and old ones merged as the system experimented with the utility of a rule. Eurisko was run in several domains: VLSI design, space-ship fleet design and elementary number theory. It worked best when interacting with a user that could save time in checking obviously (at least to the user) bad heuristics. Surprisingly, in the maths domain, Eurisko got no further than AM. Lenat attributed this to the fact that elementary number theory is already well-explored. It seems more likely, though, that the heuristics were again insufficient. To advance in number theory requires tools from many other advanced domains of mathematics (for example, algebra, geometry, graph theory, complex analysis).

AM has been criticised on several grounds. Ritchie and Hanna have highlighted methodological problems, particularly concerning Lenat's reports on AM (Ritchie and Hanna 1990). Firstly, they question whether the control structure was as uniform and minimal as Lenat suggests. For example, are the interest rules a separate body of rules or are they really concerned with tasks of the form "fill in worth of C", for some concept C? There are some indications that the rules are not stored in such a well-organised manner as Lenat states; many seem to be "lumped together" in large pieces of lisp. According to Lenat, rules are stored in subfacets of the slot to which they apply, but this does not seem, generally, to be the case.

Secondly, the organisation of concepts is not uniform. One particularly confusing notion is the concept *non-concept* which is supposed to hold information about things which are not concepts. There are two kinds of links relating concepts: generalisation-specialisation and isa-example. For instance, *multiply* is a generalisation of *square* but both are examples of *numeric functions* which, in turn, is a specialisation of *operations*. Such a set up is the cause of great confusion about the relationship between concepts.

Many rules contain "special purpose hacks" which Lenat does not describe. Unfortunately, some of AM's most interesting results depend crucially on such rules. It can easily be conjectured that these hacks were written for the special purpose of making just these rediscoveries.

The role of the user is also unclear. When a user renames an AM concept, it receives an increase in worth. So the user effectively guides the search. This makes it hard to evaluate the exact power of the heuristics. In a way, the user can be seen as fulfilling the role of a teacher of the program, supervising the path of discovery.

Ritchie and Hanna are not arguing that a program could not be written to

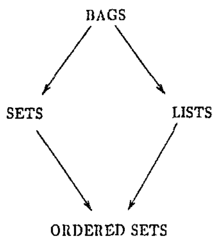


Fig. 1. The relationships between bags, sets, lists and ordered sets in AM.

achieve what Lenat describes, rather that Lenat has not done so and that this somewhat invalidates his claims.

A more methodologically acceptable version of AM has been developed by Haase (Haase Jr. 1986), who makes the following points about discovery systems in general. Firstly, they may be seen as "search processes that reconfigure their own search space." Whenever a new definition is made, or a new operator introduced, some new vocabulary is added which alters the space of concepts to be searched. Second, concept formation must be a functionally modular process: there must be a clear beginning and end to each concept creation. AM clearly falls down on this point, as well as on the next which is that concept formation must be *consistent.* This means that the form of a new concept must be the same as that of the old ones, as this will allow them to be used again in the discovery cycle. Lastly, any inquisitive process must eventually be introspective. This corresponds to Lenat's aim of trying to synthesise new heuristics.

Haase has implemented a system, Cyrano, along these lines. Cyrano maintains its concepts as a lattice of generalisations and specialisations. New concepts are inserted appropriately into the lattice, guaranteeing consistency. However, things cannot be this simple. Consider the lattice in figure 1. A bag is an unordered list. Thus bag [2, 1, 3] and [1, 2, 3] are equal, while the lists (2, 1, 3) and (1, 2, 3) are not. Sets are those bags without repeated elements. Now, just what is the relation of lists to bags compared with sets to bags? Clearly, any set is also a bag so:

$$\text{A isa set} \Rightarrow \text{A isa bag}$$

but is the same true of lists and bags? Rather, lists and bags deal with the same kind of object, but partitioned by different senses of equality. Sets are still partioned by bag-equality, lists arise by specialising this to ordered-equality. A bag is thus an equivalence class of lists:

$$\text{A isa list} \Rightarrow \text{A isa representative of a bag.}$$

This is a very different kind of relation. AM's discovery of numbers comes about by generalising bag-equality to same-length. A number is an equivalence class of bags. Is the concept *number* then a generalisation of *bag*? This would

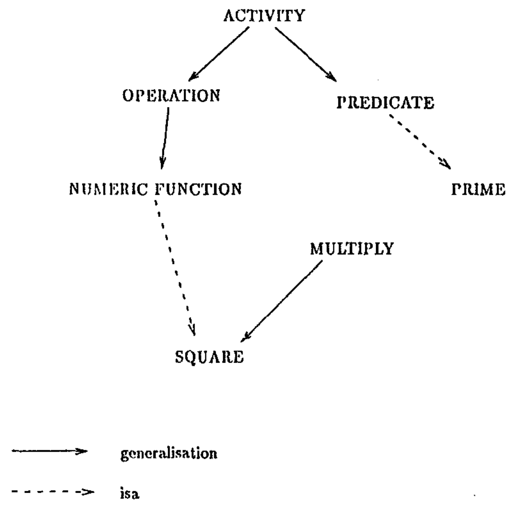


Fig. 2. A subsumption lattice with *isa* links showing instances.

imply that *set* is a specialisation of *number.* It would be clearer to maintain the distinction of generalisation by subsumption and generalisation by partition. The situtation is made worse by the *isa* links previously mentioned. Consider figure 2). *Multiply* is a generalisation of *square* which *isa* (that is, it is an example of) *numeric function.* This makes the categorisation of *multiply* problematical. A simple subsumption lattice is clearly insufficient to keep track of these details.

A further point made by Haase is related to Lenat's comments on representation. Haase describes the closeness of syntax and semantics as the "tightness" of the representation. In a tight representation, syntactic changes correspond to sensible semantic changes. This is surely the wrong way of looking at discovery. Any changes should be semantically motivated in the first place. The syntactic description should ease the process of making semantic changes, not determine it. The whole approach of generating concepts by manipulating the definitions of old ones seems inappropriate. New maths concepts should be made in mathematical ways for mathematical reasons. The discovery program should know about composition, substitution and recursion of functions explicitly. Partitioning should be explicitly recognised and motivated. AM was playing in the dark with bits of lisp code and was lucky because lisp is tightly bound to maths. However, the general solution is not to ensure a tight representation but to stop playing in the dark.

Some work has been undertaken on a re-implementation of AM along these lines. In this version, all new concepts arise as mathematical objects: compositions and substitutions. Recursion and partition have not been implemented as a further problem has been uncovered, that of motivation. A discovery process cannot make do with fixed "interest" rules, any more than other rules can remain fixed. It needs to learn to be intersted; it should develop "intuition".

AM and the related programs offer some interesting insights into cognitive modelling. First, there is the question of how a mathematician perceives mathematical objects. Doubtless such information as domain and range are closely linked with the idea of a function but the list of slots does not fully capture the sense of "mapping", "transfer" and "transformation". These ideas are more dynamic: a function *sends* an argument to its value. A fuller account of the concept *function* is necessary. In other words there is a need for multiple representation of concepts.

A second psychological phenomenon which AM highlights is *functional autonomy*. This happens when a subgoal becomes more important than the task which set it up. Imagine a baby reaching for a toy. The current goal is to get the toy for which the subtasks of learning to move the arm and grasp with the hand are set up. Of course, these skills become much more important than the original goal: they have outgrown their originator. In AM, the agenda control mechanism produces similar behaviour. Suppose the current task is to find examples of the function *first element*. If few domain examples exist (lists) then the task "fill in examples of lists" will be proposed. However, there is no backtracking. Though it is likely that the task "fill in examples of first element" will be suggested again, once some lists have been found, it is by no means necessary. In the meantime, many interesting properties of lists may be found, making them far more interesting than *first element*. An added feature of the agenda system is that, once a task has been dealt with, the values of any other tasks involving the same concept are temporarily raised. This gives a focus of attention to the program's behaviour. While attention is focused on one concept, other tasks involving other concepts will be building up. Eventually one of these will overtake the tasks in focus and become the new task; the focus will switch accordingly. Ideas and suggestions seem to bubble up the agenda, occasionally "bursting" others. The visible behaviour is, for the most part, purposeful and directed, with the occasional random flash. However, there is no attempt to control the broadening of the focus of attention in order to draw concepts together in useful ways, nor to relax constraints on concepts to allow for ambiguity.

The contrasting roles of the interest and suggest (sometimes check) rules bear some resemblance to the actions of suppressors and censors. These rules decide when to create a new concept (suggest and check) and when to stop using a concept (interest). There are some differences, however. When a suggest rule activates the creation of a new concept, it is a constructive action. There are many possible ways of making new concepts and the rules suggest the best possible type of concept to make. A censor, however, guards against certain types of combination. It would be interesting to see how the implementation of censors per se would affect the quality of the output. Interest rules are more literally suppressors, though they also have the task of encouraging good behaviour. For example, if a function turns out to be the identity, its worth slumps dramatically, making it an unlikely candidate for future combinations.

However, if a function is found to have identical domain and range, its worth is increased. A delineation of these two roles would be helpful, to show their relative importance and actions.

Eurisko has a lot to say about the accumulation and adaptation of heuristics, though as its results are fairly indecisive, it is difficult to draw any conclusions. An achievement of some significance is the representation of heuristics in an adaptable fashion. Slots can be created and destroyed; indeed each slot has its own descriptive frame. A program needs to know about a heuristic, not just be able to use it. The problem of how certain heuristics can control the development and action of others is very important. To a great extent, this factor governs the flexibility of a system: the trade-off between completely directed search and completely random search.

The originality of the behaviour of these programs is hard to assess in the light of Ritchie and Hanna's criticisms. Attempts to clean up AM have yet to produce any significant discoveries. Most discoveries are new to the program but known to the programmer. AM did, apparently, produce one idea that was new to Lenat: maximally divisible numbers. However, AM had little to say about them and subsequent work was done by Lenat himself.

In conclusion, the AM family of programs have not been too impressive in terms of actual results, but they give hope in that they are displaying that right kind of behaviour.

A more recent discovery program, GT, has a sufficiently novel representation scheme to warrant separate discussion. GT, the Graph Theorist, reasons and conjectures about finite undirected graphs (Epstein 1988). Graph theory is, in some senses, a good domain for computational study as it does not require a large amount of knowledge from other fields as does, say, topology. The representation system used, though, does not, at first glance, compare with that of the mathematician. However, it is sufficiently powerful to build up some "intuitions". Graph theoretic concepts are represented by triples of the form (f, S, s). S is a (minimal) set of graphs with the property (the seed set); f is a function of graphs and s is a selector, giving conditions on f. Repeated iteration of f on S under s is guaranteed to completely and correctly generate those graphs which have the appropriate property. Extra information is held in additional slots: examples, origin, pointers to generalisations and so on. One clear advantage of this representation is that as many examples as are required can be generated. Another advantage is that it allows certain proofs to be constructed. The two basic proof methods are subsumption ($p \Rightarrow$ q) and merger (the intersection of p and q). These can cover four basic theorem forms:

1. $p \Rightarrow q$: q subsumes p
2. $p \Leftrightarrow q$: p subsumes q and q subsumes p
3. $(p \wedge q) \Rightarrow r$: r subsumes the merger of p and q
4. $p \Rightarrow \neg q$: the merger of p and q is empty.

There exist methods for constructing mergers and proving subsumptions using f, S and s for each concept. It can be appreciated that GT has an inductive approach to graphs, based on testing conjectures made through observations on small graphs with certain properties. However, whilst mathematicians do use induction, there is something inescapably visual about graph theory. As work progresses, developing new proof techniques, maybe the overall view of a graph will begin to play a part.

A different approach to discovery is found in the BACON family of programs (Langley *et al.* 1987). These operate on a data-driven basis. The underlying philosophy is founded on ideas similar to those of Francis Bacon. Starting with a body of data, the goal is to find a theory that adequately describes or explains it. The BACON programs are good describers: given data, suitable laws are sought and found by a heuristically-guided process. Thus, given information about planetary co-ordinates and trajectories, Kepler's laws are deduced. The whole process is an ad hoc curve-fitting exercise. BACON never constructs any models or structures to help its predictions, rather it plays "syntactic number games until it has completely summarised the data." To remedy this, the programs GLAUBER, STAHL and DALTON were written. GLAUBER examines qualitative data and produces classifications that allow general laws. For example, given the result from a chemistry experiment:

(reacts input {HCL NaOH} output {NaCl})

GLAUBER suggests the law:

(reacts input {acid alkali} output {salt})

In contrast, STAHL uses similar inputs but with the aim of model building. STAHL works in the domain of phlogiston theory — the idea that fire is composed of a substance called "phlogiston". Reactions are studied using three basic rules: INFER, SUBSTITUTE and REDUCE. For example, consider the following initial information:

(reacts input {charcoal air} output {phlogiston air})
(reacts input {calx-of-iron charcoal air} output {iron ash air})

indicating that charcoal and air react to produce phlogiston (in the flames) and ash. REDUCE produces:

(reacts input {charcoal} output {phlogiston ash})
(reacts input {calx-of-iron charcoal} output {iron ash})

INFER (components charcoal are {phlogiston ash})
SUBSTITUTE (reacts input {calx-of-iron phlogiston ash} output {iron ash})
REDUCE (reacts input {calx-of-iron phlogiston} output {iron})
INFER (components iron are {calx-of-iron phlogiston})

DALTON is a similar program that deals particularly with atomic modelling.

The major problem with all these data-driven programs is that an idea of what to look for is built into the heuristics. Given the notion that the results of a chemical reaction are determined by the participating chemicals, compositional deduction becomes largely a matter of experimental accuracy. The really creative insight was that this could be done at all. BACON plays with the data until it has fitted an accurate formula, but this has nothing to do with the great effort Kepler made in overcoming his prejudices against ellipses. Until Kepler's time, it was assumed that all heavenly motions were circular. Of course, it was to prevent this kind of thing that Bacon argued for an unbiased view of data. In practice, this is impossible. An open-minded astronomer would look impartially at the different figures for radii and circumferences in vain. Why does STAHL never conjecture that phlogiston does not exist? The best it does is to go into an infinite loop. It cannot break out of the initial presentation of the data.

All the programs in the BACON family suffer from extreme rigidity of behaviour, built into the heuristics. The only creative characteristic which they have is the ability to derive only sensible discoveries from data. There is no flexibility, ambiguity, multiple representation or elaboration of ideas. It seems that when heuristics are tightened to avoid the generation of trivialities (as both AM and Eurisko were prone to do in the long run), all flexibility of representation is lost.

A final example of a discovery system is the program SPARC/E (Dietterich and Michalski 1986). This worked in the domain of sequence prediction using playing cards as the elements of sequences. Rather than be restricted to a single way of interpreting the data (as with the BACON programs), there are three different models which the program attempts to fit to the data. This proceeds by simultaneously generalising the data and specialising the models. The models are the *disjunctive normal form* (DNF) which express rules for sequences as disjunctions of card properties, the *decomposition* model which lists a set of mutually exclusive implications, and the *periodic* model, which deals with sequences of fixed period. While this method extends the applicability of discovery heuristics away from a single model, the fundamental problem remains and is quickly revealed when the program is presented with a sequence that does not fit any of its models: it cannot adapt its knowledge of sequence forms in any flexible manner.

5. META-RULES FOR CREATIVE FLEXIBILITY

The rigidity of rule-based systems has led some researchers to advocate the use of meta-rules: rules which can reason about and create rules. In some senses, Eurisko was performing such meta-activities, but in a true meta-rule system, the idea is that each layer of rules maintains a strict control on its domain. The data, rule, meta-rule hierarchy is fixed. One proponent of this approach is Yazdani (Yazdani 1989), who, during work on a story generating program, noticed that some of the more inventive stories were found in unwanted outputs produced when the program went wrong. This led to a description of creativity as an "almost random" generative procedure coupled with a validation process. The generation is "almost" random as only small variations on already known themes are used. To control these variations, a set of meta-rules is required to work on the rules producing the known theme.

An example of this approach is the program MUSCADET, a knowledge based theorem prover for topological linear spaces (Pastre 1989). Its proofs are built by heuristics which, in turn, are constructed by meta-rules supposed to capture the reasoning of a mathematician. However, this approach makes the program more concerned with proof structure than with the mathematical objects themselves. It cannot distinguish between important and trivial issues from a mathematical point of view. This seems a very artifical method of providing adaptability.

A variation on the meta-rule idea is seen in the DAY-DREAMER program (Mueller 1987). This is a planner that operates in two interacting domains: the personal world and the objective world. Both these domains are controlled by their own goals. Events in the objective world (say a failed plan) can affect the personal goals (say the need for comfort, revenge). The program then "day-dreams" by fantasising different situations in which the personal goals are met. Successful plans are abstracted and stored in memory. When the program next encounters a suitable objective world situation, these plans can be re-instantiated and applied. These day-dreamed plans can be adapted to fit into different objective goal situations by making appropriate variable changes. This provides some flexibility in the plan representation, but there is a fundamental problem of how to decide which variables can be changed and how. The abstraction process could benefit from multiple representation: methods describing how a plan could be used in different types of situation. One advantage of having these plans built in "dream-time" is that it allows them to be elaborated to discover if they are of any use. If not, the program can continue its fantasy with other possibilities.

The trouble with the meta-rule approach is that, whilst rules are given some scope for flexibility, it is pre-determined by the meta-rules. This requires that the programmer foresee the ways in which rules are allowed to be flexible, and this knowledge is then coded in *a priori*. Thus the issue of flexibility is really only hidden by this division of rules into layers. Perhaps what are needed are "meta-meta" rules? But, again, this would just bury the issue further. According to Douglas Hofstadter, we should

> side-step the topless tower of bureaucracies and meta-bureaucracies above by making rule-like behaviour emerge out of a multi-level bubbling broth of activity below. This means that you give up the idea of trying to explicitly tell the systems as a whole how to run itself. Instead, you content yourself with defining explicit micro-behaviours that will interact in vast numbers . . .

(from (Hofstadter 1986a)). In other words, the structure of representations need to be examined to see how they can be broken down into components. These components could then be used as building blocks for further representations. If two representations are built from similar building blocks, then an *analogy* exists between them. It may be possible, then, to deduce the components of a representation by examining its roles in analogies.

6. ANALOGY IN CREATIVITY

It would seem that analogy has much to do with the problems of creativity. In particular it addresses the problem of how links can be forged between two schemata, once they are activated simultaneously.

A standard method of approach to analogy is Gentner's Structure Mapping Theory (Gentner 1983). An example of the kind of situation which can be handled by this method is deriving an analogy between the relation (*planet, sun*) and (*electron, nucleus*). Each object has a number of properties stored in a frame-like structure, and there are relational links (such as *sun is-hotter-than planet*) between the object in each domain. The problem addressed is that of which properties and relations in one domain can be mapped on to the other domain. Notice, though, that the representation of each object and domain is essentially a rigid, syntactic structure. There is no question of these structures becoming flexible, or ambiguity about the properties tolerated. As a simple example of this problem, consider the vast number of properties and relations which are left out of the representation, for example, *sun is-more-yellow-than planet* or *sun only-seen-at-day-time*. It is clear that people can easily associate such properties (with a little imagination) but with a fixed structure, the programmer has to decide which properties are in and which are out. Further, there is no way of distinguishing which properties are more essential than others (for example, the fact that the sun is further away from human observers than the Earth, is purely relative). Consequently, analogy research (of this kind) does not have much to say about creative behaviour.

One positive contribution that is made concerns the elaboration of mappings. Analogies can be constrained or guided by pragmatic considerations (for example, see (Keane 1988)). In such systems goals, plans and roles drive the mapping process. This helps avoid useless maps for semantic reasons rather than syntactic ones. However, the actual mapping process is still between rigid structures.

7. TOWARDS FLEXIBLE REPRESENTATIONS

One of the classic rigid representations is the script (Schank and Abelson 1977). This is a generalised description of a sequence of events that is expected to take place in a given situation. The most famous example is that of the restaurant script, illustrated in figure 3. A script drives expectations and provides explanations when things go wrong (for example, the diners have no money; the food is disgusting). The rigidity of scripts is demonstrated when trying to apply the restaurant script to a fast-food restaurant, a superficially similar activity but with very different expectations. Should a new fast-food script be invented or can the old script be adapted?

To help make scripts more flexible, Schank developed the notion of an explanation pattern, or XP (Schank 1986). An XP is a "core" explanation that can be fitted to many circumstances. As well as the central script it carries a

```
RESTAURANT SCRIPT

actors: diners, waiters, chefs, managers.

props: tables, chairs, food, menu, money, bill, cutlery, plates.

initial state: diner hungry, empty chair, diner has money.

sequence: diner enters, diner sits, diner reads menu, diner orders,
          waiter brings food, diner eats food, waiter brings bill,
          diner pays waiter, diner leaves.

final state: diner has less money, manager has more money,
             diner is not hungry.
```

Fig. 3. A Restaurant Script.

number of indices to the kind of situation to which it could apply. Indices exist for the type of failure needing explanation, the type of event involved and the goals motivating the explanation. The explanation process starts with a question about an event. By determining the associated indices a standard question is generated and all the XPs fitting this form are accessed. The XPs are "tweaked" to fit the exact circumstances, if possible, by changing round actors, props, time ordering etc. A collection of explanations specific to the event is thus generated. The indexing system makes the central script much more widely applicable than before, so the representation is acting dynamically. However, the tweaking technique is basically driven by heuristics, so there is only a partial escape from rules. The question to ask is: how do the tweaking rule arise? The obvious temptation is to suggest meta-tweakers to tweak the tweakers! However, even with this partial flexibility there are some gains.

Explanations are often based on memories of similar events. If an event is unusual and an explanation found it is remembered when a similarly unusual event occurs. Such remindings are the basis of many analogies which are often used in common sense reasoning, which is always creative in a small way. The XP system captures this kind of creativity. However, to be truly adaptable, the core script of an XP would have to change. Schank has achieved flexibility by adding on indices to minimal scripts. To follow Hofstadter's philosophy fully would require the scripts to be broken down into "micro-explanations". This core rigidity is manifested in the kinds of explanations produced by the implemented system — they are acceptable but somewhat obvious. There are never any highly original explanations produced.

The indexing system begins to show how primary and secondary process interaction can be modelled. A specific event is encountered during secondary processing: as part of the focus of attention. The indices fired by the anomalies in the event match various XPs in memory: a primary process. The XPs are then made specific (made conscious) by being tweaked. Again, the two flaws in this model are the rule-like nature of tweaking and the rigidity of the memory of XPs.

What happens when several XPs meet the required conditions? In the current implementation, all XPs are treated sequentially. However, a system can be imagined where XPs compete for notice by "consciousness". This could be on the basis of recency in memory; the amount of tweaking that has to be done; the popularity of each XP and so on. As a result, only one XP will be used unless others are then "consciously" sought. This would produce the bubbling up effect present in AM's agenda mechanism, which is related to constraint relaxation and the defocusing of attention. If this processing were done in parallel then various factors (for example, time-delay, limited resources) could produce an "almost random" behaviour.

As explanations are related to memories, the stock of XPs grows with experience. A question not considered by Schank is when to make a new XP, rather than heavily tweaking an old one. This problem, in some ways similar to AM's concept creation problem, would need control to prevent the creation of many useless or trivial scripts.

A different approach to flexible representation is found in *classifier systems* (Holland 1986). These consist of a collection of rules (called *classifiers*) and a message list. If a rule's if-part matches a message on the message list then it can post its then-part as a new message. The basic cycle is as follows:

1. Place input messages on current message list.
2. Find all rules that can match messages.
3. Each such rule generates a message for the new message list.
4. Replace current message list by new message list.
5. Process new message list for any system output.
6. Return to step 1.

If a rule posts a message which another rule can match, then they are *coupled*. Using this technique, chains of inferences can be set up. The power of classifier systems comes by assigning weights to each classifier in the rule list. This can be done in such a way that the most successfully used classifiers get the strongest weights. Classifiers can then compete for the right to post their messages in response to other messages. A classifier's bid depends not only on its strength but also on how specific it is. More specific rules outbid more general ones (weights being equal). Because the chains of inference are not built-in but arise through selective competition, the high-level inferencing behaviour is very flexible. Such bid-passing would happen in step 3 of the above cycle. A rule would pass its bid to the rule which posted the message to which it is responding.

This kind of behaviour is reminiscent of *blackboard systems* (Craig 1988). However, classifier systems have the advantage of learning from their environments. By using a suitable credit-apportionment scheme (the so-called *bucket-brigade* algorithm) a classifier-system can learn appropriate chains of response for different situations, whilst still retaining general rules for situations not yet encountered. Thus classifier systems have all the right characteristics for creative behaviour: they have flexible representation; can tolerate ambiguity; they avoid functional fixity (by having multiple competing representations) and learn from their environment (that is, elaborate the consequences of their actions). Further, by representing classifiers as bit strings, *genetic algorithms* can be employed to enable the system to learn an appropriate classifer set. These algorithms are designed in such a way that useful combinations of classifers (creating new ones) are more likely to be produced than useless ones. Going by the strengths of the classifiers, there is an evolutionary effect, whereby only the "fittest" classifiers will survive.

8. DECENTRALISED SYSTEMS

One of the sources of the flexible behaviour of classifier systems is in their lack of a central controller (for example, a planner). Each classifier operates at a local level and it is only as chains are constructed that high-level characteristics become apparent. This kind of behaviour is called *emergent*, since global, high-level properties emerge without explicit control from the interaction of many local, low-level actions. This kind of approach avoids the structural limitations which get imposed by central controllers (for example, the fixed set of rules governing grammar based systems), and allow for flexibility as a high-level property.

Minsky has argued for such a distributed theory of mind, describing the mind as an integrated society of agents each with its own specialised task (Minsky 1985). Such societies are built up by learning from experience. Memories are controlled by special agents called *k-lines* which, in appropriate conditions, will reset a group of agents to the state they were in when last these circumstances were experienced. Minsky does not make clear how such agents could be implemented, or any learning algorithm by which k-lines could be established. His work serves rather to motivate the ideas of distributed control, and makes the important connection between memory, thinking and perception. This follows Hofstadter's view that "Cognition equals re-cognition", (Hofstadter 1983).

One approach towards decentralisation of control and emergent behaviour is *connectionism* (see (Pollack 1989)). These networks are generally used to model very low-level behaviour such as vision. Whilst they are good at generalising from a given data set, the learning procedures are very artificial and much research is still needed before connectionism could be applied to the larger problems of creativity. Some relevant progress has been made in the domain of sequence prediction, though so far only relatively simple sequences can be handled (Elman 1988).

Emergent behaviour is very much at the centre of research into *artificial life* (Langton 1989), whose aim is to discover how life-like behaviour can emerge from large collections of local agents. These systems thus simulate the development of societies of individuals. By changing the level of description, they can be seen as models of how mental agents interact within a mind.

A leading researcher in this approach to AI and creativity is Douglas Hofstadter (see, for example (Hofstadter 1986b)). One of Hofstadter's early programs, Jumbo (Hofstadter 1983), introduced the idea of *temperature-*

controlled randomness. The *temperature* of a system is a measure of the amount of order that has already been discovered. Little order corresponds to a high temperature (and a large amount of randomness) and high order is represented by low temperature (little randomness). The image suggests that a system will "freeze" into states of high order. When not much information has been gathered, the large random factor allows unusual paths to be explored. Once some order is found, however, the random factor diminishes and the system works on standard lines to complete the model.

Jumbo's domain was the solving of anagrams (or "jumbles"). There were various operators for swapping letters and syllables around, and rules for assessing what combinations of letters made good (that is, English-like) syllables and which syllables went well together.

An extension of these ideas is found in the Seek-Whence program (Meredith 1986), which tried to analyse integer sequences arranged in patterns. For example, consider the following sequences:

a. 1 2 3 4 5 5 4 3 2 1
b. 1 2 3 4 4 3 2 1

What is to **b** as 4 is to **a**? The problem is to determine the role that 4 has in **a** and to find a filler for the counter-role in **b**. Suppose the role in **a** is determined as "the fourth digit in the sequence". Then the answer is 4. However, an alternative role: "the number surrounding the central pair" gives the answer 3. This seems a much more satisfactory solution as it takes into account the structure of the sequences. But what fills the counter-role in the following sequence?

c. 1 2 3 4 4 4 4 3 2 1

Seek-Whence construct hypotheses about such sequences (seeking whence they came) from primitive structural units (cycles, constant, successor, . . .) which it uses to make predictions. It operates on three levels. The *cytoplasm* contains the elements of the sequence and groupings of such elements. These are seen as "manifestations" of "ideal" concepts in the *platoplasm.* Useful groupings are preserved as templates in the *socratoplasm.* Such templates may eventually become hypotheses. There is a cycle of information flow between these levels (new data from the cytoplasm and predictions from the socretoplasm). If a hypothesis is rejected, it dissolves releasing its components to break up and rearrange themselves. Again there is a *temperature* measuring the amount of order that has been discovered and controlling the randomness allowed in the building of hypotheses.

A major problem is that old hypotheses that have been rejected keep on getting rebuilt (functional fixity). To deal with this, hypotheses are *freeze-dried* in order to provide a check against recycling. However, this solution is out of the spirit of decentralisation as it requires a central memory. A further weakness is shown up in the limited number of sequences the system could actually parse. The set of primitives used for representation was too narrow, but there could be problems with scaling as this would increase the number of possible combinations and lead to even more useless hypotheses swamping the system.

Apart from this major problem, Seek-Whence represents a positive move towards flexible representations (at the high level) emerging from low-level, local interactions.

The program Copycat uses similar methods for constructing analogies (Mitchell and Hofstadter 1990). Copycat tries to construct analogies between strings of letters. For example:

> **if** *abc* → *abd*
> **then** *ijk* → *?*

The program tries to find structures in the first mapping that have analogues in the second and so derive a suitable answer. In this example, the first mapping might be perceived as having the form "replace the last letter by its successor", in which case the answer *ijl* would be produced.

There are three parts of the Copycat architecture: the *Slipnet*, the working space and a pool of structuring agents called *codelets*. The working space represents the system's current understanding of the situation: the structures it has found and the maps between them. The Slipnet is a network of the system's permanent concepts such as *successorship*, and their semantic relations. The links in this network are weighted and activation can spread along the links in proportion to these weights. The weights vary according to the context. Thus, if the idea of *opposites* has already been useful then the link from *leftmost* to *rightmost* will be strong (being an *opposite* relation). The activation of the concepts in the Slipnet provides guidance for the kinds of structures that should be tried out in the workspace. These are constructed by the codelets. Each codelet also has a weight (adjusted by the Slipnet) which determines its urgency. These weights provide a probability distribution by which the codelets are selected for running. Effectively, the tasks are run in parallel at rates proportional to their urgency. This distribution is adjusted by the *temperature* of the system; the measure of disorder. At low temperatures only the most urgent codelets will be picked (the strictly relevant ones), but at high temperatures all codelets are more or less equally likely. Once more, there is a feedback between levels of description: the high-level emergent structure influences the low-level building activity.

The major development of Copycat from Seek-Whence is in the introduction of the Slipnet. This is used as a guide by which to avoid useless combinations and to encourage useful ones. It also explicitly allows for the toleration of ambiguity by showing which concepts are allowed to slip into others. However, it is surely a disadvantage to have this built in as an *a priori*. It might be better if the system learned which concepts could slip and so be able to invent slippages of its own. It is unclear whether or not Copycat requires "freeze-dried" memories in order to prevent the repetition of bad ideas. Perhaps these get reflected in the activations in the Slipnet.

It seems that most of the characteristics of creativity are beginning to emerge in the Copycat program. However, it is very limited in its domain of application. In particular the kinds of maps which can be handled are of a very simple form, that would only need context-free grammars to describe them. It is interesting to consider how it would be able to scale up to context sensitive rules. This would require the addition of a memory in order for the system to move about over the structures. If the sequence is regarded as written on a Turing Machine tape, a parallel tape is required, and the ability to move in both directions, reading and writing.

9. CREATIVITY THROUGH AN EMERGENT MEMORY MECHANISM

A new theory of creativity has been formally defined, implemented in several variations, and tested extensively (see (Rowe 1991) for full details). In this theoretical model, which can be viewed as a concrete realisation of Minsky's "Society of Mind" ideas, autonomous agents can combine to generate solutions to problems — either (relatively) unconstrained new proposals (output creativity), or possible solutions to a highly constrained problem situation (input creativity). Successful representations that emerge can be kept intact, and may subsequently function as agents that cooperate in further problem solving (corresponding to Minky's k-lines, see below).

This kind of mechanism is an *emergent memory.* Instead of concepts or categories being recalled whole, because they match a certain input, concepts are reconstructed in appropriate situations. If the same situation is encountered again and again, eventually one memory link may come to represent the whole, but the partial memories would still exist to be used in combination with others. This provides flexibility, as often an experience is similar, but not identical, to a previous one. In such a case, a substantial part of the previous representation may be constructed (in as far as it conforms to the external constraints of the new situation) and then smaller memories and the construction agents themselves, may be used to fill in the gaps.

Minsky proposed the idea of agents that form links between other agents, so as to create memories (Minsky 1985). He called such agents *k-lines*, but gave no details as to how such agents could operate. A formal theory of learning based on this idea has been developed, and it provides the basis the model to be described below.

Cognitive units are thought to have *activation levels* which determine how active they currently are, how much attention is focused on them. This corresponds to the notion of bidding. An agent's bid represents its claim to be processed. The higher the bid, the more chance of it acting. Thus the process of task selection is a theory of attention.

The emergent-memory model has been implemented (in a program called 'GENESIS'). This implementation is based on the k-line idea, and has been tested extensively. In this system, there are two kinds of agents: *builders* that construct representations and *k-lines* which group together successful agents. The cue for the creation of a k-line is when some subgoal has been reached in the current problem situation. The k-line collects together all agents which have worked towards this subgoal and becomes an agent in its own right. It may be activated in future if half (or some other threshold) of its component agents are used together. It responds by activating the remaining agents. In this way, memories are reconstructed in a bottom-up fashion but with higher level representations able to guide the process. All agents have weights which help determine which potentially relevant agents become activated. These weights are altered according to the *bucket-brigade* algorithm used in classifer systems (Holland 1986). Following this algorithm, successful sequences of agent activity are rewarded by a pay-off whereas unsuccessful sequences are gradually reduced in weight.

This computational model of cooperating, autonomous agents gives us a general framework to support creative behaviour. And, moreover, within this framework two classes of psychological theory can be directly compared. The crucial difference resides in just one component of the model: the task selection strategy.

9.1. *Task selection strategy: the point of comparison*

One class of psychological theory of creativity centres around the notion of a four-stage model of creative behaviour. The four stages are: preparation, incubation, illumination and verification.

Preparation is the stage of concentrated work, accumulating data and trying out various approaches. Incubation is the time when the mind is relaxed or working on a different problem. The third stage, illumination, is the moment of insight when the solution found is made conscious. This stage seems impossible to predict or control. Verification consists of a further period of conscious work when results can be checked out, for example, against new data.

A modern theory based on the four-stage model has been put forward by Colin Martindale based on various theories and experiments connecting creative problem solving with levels of *cortical arousal* and *focus of attention.* Measures of cortical arousal can be displayed, for example, as an electroencephalogram (EEG) which depicts brain-wave patterns picked up as voltage changes on various parts of the head. The focus of attention is the extent to which short-term memory and consciousness are concentrated (Martindale 1981). In low states of cortical arousal, attention is unfocused. This corresponds to primary processing: the activation of many cognitive units to a small degree. At high arousal levels, attention is highly focused, corresponding to secondary processing (high activation of a small number of units). It has been shown, for example, that creative people exhibit a marked decrease in cortical arousal whilst solving problems creatively (Martindale and Hines 1975). This was measured by alpha-wave activity. However, when given uncreative tasks to do, or when elaborating an idea, creative people do not show this distinctive arousal pattern.

The four-stage model of the creative process has been challenged by Robert Weisberg, who argues that this model is not supported by experimental evidence and its continued acceptance is due to the propagation of myths about the creative process in psychological folklore (Weisberg 1986).

Weisberg's criticism of the incubation stage, in particular rests on the results of a number of experiments that have tried to isolate incubation in laboratory conditions. One such experiment, conducted by Olton, involved observing expert chess players solving chess problems (Olton 1979). One group was allowed to take a break (to encourage incubation) while the second worked continuously. However, there was no significant difference between the performance of the two groups.

In a second experiment, by Read and Bruce, subjects were shown pictures of actors from the 1960s whose names were not easy to recall (Read and Bruce 1982). During the following week, subjects kept a diary of their attempts to recall these names. Only about 4 percent of successful recalls involved spontaneous remembering of the names and most of these incidents were reported from only four of the thirty subjects. All other cases involved conscious effort of remembering, showing that incubation (unconscious processing) and illumination are not necessary for insight to take place.

Weisberg's claim is that creativity is merely a part of normal solving, and there is no special creative process. It is intelligence combined with a large amount of domain knowledge that enables a person to have creative insights, but the mechanisms are just the same as those used in normal everyday problem solving.

The main weakness of Weisberg's view is that it fails to account for the peculiar experience of illumination which seems to occur on some occasions (those associated with creativity), but not on others. This observed effect is explicitly accounted for in the cortical arousal theory as a return from low to higher levels of arousal.

The crucial element of difference between the two psychological theories is the arousal mechanism proposed by Martindale. The alternative view is that there is no such special mechanism for creative thinking. To model the process of regression, the *terraced scan* search method is used (Hofstadter 1983). Cortical arousal is represented by the *temperature* parameter. This parameter varies according to the state of the problem solving process. When there is little order discovered, the temperature is high. This corresponds to a *low* level of arousal. This is because a high temperature increases the amount of randomness in the task selection process. In other words, it flattens the gradient of activation across agents. This is precisely what low cortical arousal is hypothesised as doing. At times when little progress has been made on a problem, arousal is low and attention defocused. This means that weak cognitive units may receive some attention. A low temperature arises as the system converges on a solution. At low temperatures, only the strongest bidding agent is likely to run. This corresponds to attention being highly focused on one or two units under high levels of cortical arousal. If a representation is found to be incorrect it is abandoned. Thus temperature increases after such an event. This corresponds to a regression from secondary processing (focused attention) to primary processing (unfocused attention), which, Martindale suggests, is a key mechanism for creative thought.

Weisberg claims, however, there is no such mechanism. Rather, creative thinking is just a part of normal thinking. Normal problem solving involves trying out the most likely ideas first and correcting them if they prove inadequate. If an idea is completely wrong, it may be replaced by a competing one. In the computational theory, this corresponds to task selection being a *best-first* process. The highest bidding agent is the one that runs. If a representation is wrong then it is abandoned. The bucket-brigade credit-assignment scheme means that agents on the chain that led to this representation are all eventually weakened (if the same mistake is repeated). This means that an alternative move will be stronger, leading to the building of a different representation. The difference between the two theories can thus be seen as a difference in task selection strategy.

9.2. *Computational behaviour*

As with many aspects of intelligence, models of creativity present researchers with the difficult task of showing that the claimed phenomenon (in this case creative problem solving) is indeed modelled by their programs. The card game *Eleusis* was chosen as the application domain within which the GENESIS models operate. Gardner selected this game as one which models the classic creative act: theory discovery in science (Gardner 1977). In this game the dealer thinks up a card-sequence rule (e.g., red odd follows black even, etc.) — output creativity — and then the individual players must discover the rule — input creativity — as a result of evidence collected by playing cards which the dealer declares to be correctly following the rule or not. In addition, the scoring is such that overly complex rules are counterproductive. The best score for the dealer is obtained from a rule that is quickly discovered by just one of the players.

In a number of experiments based around a simplification of this sequence prediction game, called *micro-eleusis*, the two k-line programs (one for each selection strategy) were compared with each other and also with some other programs. GENESIS with the best-first strategy (GENESIS-BF) models Weisberg's "normal problem solving" theory and the terraced scan version (GENESIS-TS) models the cortical-arousal theory. Both programs took turns as dealers and players.

The GENESIS programs construct theories about sequences which can be re-used. The theories are stored in k-lines which are "freeze-dried" abstractions. However, they are constructed in a bottom-up fashion creating a society of inter-related meaningful chunks. The abstraction process leads to increased applicability through (sometimes partial) analogies. The reconstructive nature of the k-line theory means that the representations are flexible: there is no *a priori* commitment to certain models. The threshold activation of k-lines enables them to be activated in partially matching situations. The competition between theories, governed by the bucket-brigade algorithm, provides a controlled relaxation of constraints. The terraced-scan version goes further in allowing only partially relevant theories a chance to be activated. Both versions allow multiple representations of the notational and structural kinds. The best-first selection method favours staying with one theory until it is sufficiently weakened for another to take over. The terraced-scan method encourages the creation of multiple representations and allows them opportunities to be used without the risk of their weights being penalised because of short-term problems. However, it also allows many useless or over-complex theories to be produced. The terraced scan tries to control the randomness, but does not direct the combination process (beyond the requirements of the external situation, that is). The best-first method, on the other hand, always has a reason for its combinations. There is still room for further self-criticism, however. It has a method of elaboration in the bucket-brigade algorithm which can be harsh, but prevents the build up of useless k-lines. The terraced-scan version of the bucket-brigade algorithm tends to upset the k-line organisation.

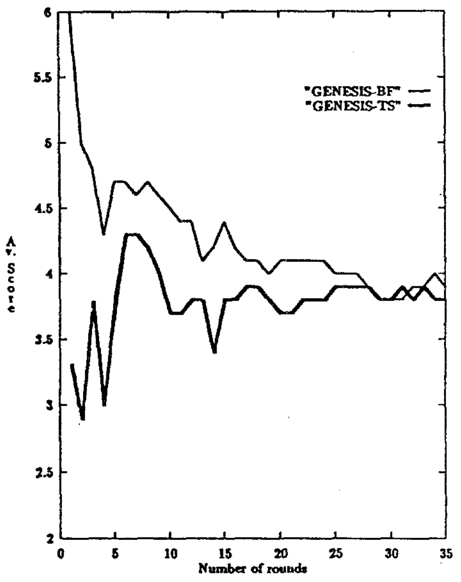
As can be seen from the two graphs of average scores obtained by the two programs, they both settled down to play reasonable games as both dealers and players. The most negative observation from this data is that GENESIS-TS is getting progressively worse as a dealer. This was because it produced rules that look complex but which are, in fact, so full of non-determinism that they are trivial. One such rule is illustrated in Figure 4 as a finite-state automaton in which the arcs are labelled with the cards that are playable to effect that transition — the labels are self-explanatory given that "b" is black, "r" is red, "o" is odd, and "e" is even. The GENESIS-BF version, by way of contrast, does seem to learn an appropriate output creativity, whereas GENESIS-TS suffers from too much randomness.

In addition, the two programs played a number of games against two human players, and although the humans beat the programs, the overall performance of the programs was quite creditable. GENESIS-TS scored just over half of the human scores (the two human players scored almost exactly the same totals), and GENESIS-BF was less impressive at just one quarter of the average human score. In terms of scores as dealers, both programs and one of the humans were equal; the other human scored just over twice as highly. From these trials we conclude that the programs can be said to exhibit significant creative behaviour, both output creativity (when dealing) and input creativity (when acting as players).

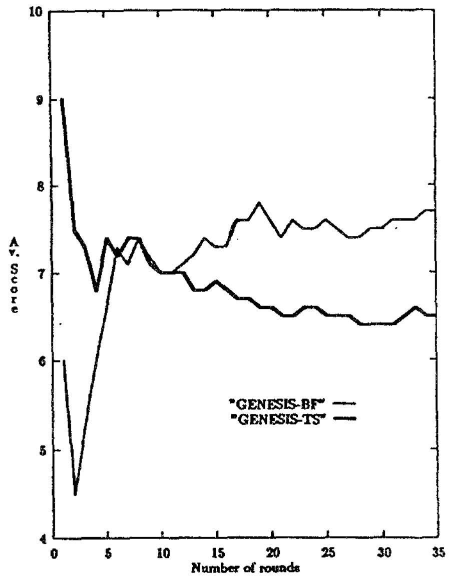
One of the most important results is the behaviour of GENESIS-BF in its analysis of micro-eleusis sequences. As commitment to one theory is weakened by the bucket-brigade algorithm, another (possibly completely different) has the chance to take over. The effect of this is an "aha!" experience as an unfruitful path is suddenly replaced by a more profitable one, possibly of a quite different sort. This is important as it seemed to be a weakness of the best-first model that it couldn't account for this phenomenon that is explicitly modelled by temperature change in the terraced scan model. The terraced scan can also help GENESIS to get out of ruts, but at a higher cost in speed and complexity.

9.3. *Conclusions for GENESIS study*

The GENESIS study has looked at both psychological factors and artificial intelligence mechanisms. Using non-classical AI techniques, based on the models of Artificial Life, an *emergent memory* model was developed with which



Average play per hand over 35 rounds



Average deal score per round of each program over 35 rounds

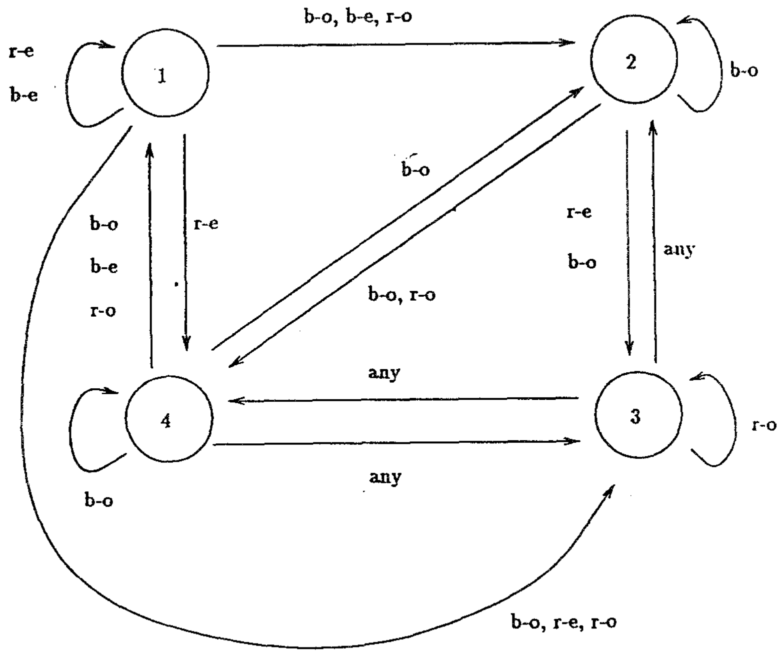


Fig. 4. GENESIS-TS produced a rule that looks complex, but is in fact trivial.

the two opposing psychological theories could be compared. The theoretical background to this led to two hypotheses concerning the action of memory. A behavioural result was that *analogy* can be seen as an emergent property of memory, rather than being an explicitly programmed mechanism.

The most significant psychological implication arises from the "aha" behaviour of the best-first search. This repairs the chief weakness of Weisberg's theory which argues that there is no special mechanism for creativity, yet a distinctive brain-wave pattern is sometimes observed. The fact that this special effect can be objectively observed (e.g., on an EEG) may indicate that it is an effect, rather than a cause, of the emergence of an insight. The "no-special-mechanism" viewpoint now seems to be stronger than the cortical arousal theory.

There are two main implications for artificial intelligence. Firstly, that Minsky's k-line theory can be made to work reasonably well. There is still room for improvement, though, particularly the needs for feature selection and self-criticism. The second result is that the terraced scan (devised by Hofstadter) is a useful search algorithm in some cases. It is useful for avoiding local optima. However, it may need work to make it more stable and efficient. When there is some kind of learning strategy, the terraced scan interferes with its action. It may be possible to devise a learning algorithm which co-operates with the terraced scan.

The central result, on which these others rest, is that a multi-agent system with an emergent memory mechanism, displays, to an extent, the required characteristics of creativity. That is, it develops flexible knowledge representations, it can tolerate ambiguity, it makes use of multiple representations, it is guided by semantic as well as syntactic changes and it assesses the consequences of its creations.

10. SUMMARY

Creativity is an important, but elusive, phenomenon intimately bound up, it seems, with the notion of intelligence. We have surveyed a wide range of AI programs which have, in their various ways, explored some aspect or interpretation of this phenomenon. We have reviewed these programs using a set of five characteristics that we believe must be central to any 'mechanism' for creative behaviour.

Finally, from the shortcomings of the earlier programs we developed an argument for the mechanism underlying creative behaviour to be a co-operating collection of autonomous primitive agents whose activities will give rise to flexible, high-level representations. This theory has been formally defined, implemented in the GENESIS system, and several implementational alternatives have been explored (see (Rowe 1991) for full details).

REFERENCES

I. D. Craig. Blackboard systems. *Artificial Intelligence Review*, 2: 79—118, 1988.

R. Davis and D. Lenat. *Knowledge Based Systems in Artificial Intelligence*. McGraw-Hill, 1982.

Thomas G. Dietterich and Ryszard S. Michalski. Learning to predict sequences. In Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, chapter 4, pages 63—106. Morgan Kauffmann Publishers Inc., 1986.

K. Duncker. On problem solving. *Psychological Monographs*, 58(270), 1945.

J. L. Elman. Finding structure in time. Technical Report CRL Technical Report 8801, Center for Research in Language, University of California, San Diego, April 1988.

S. L. Epstein. Learning and discovery: One system's search for mathematical knowledge. *Computational Intelligence*, 4(1): 42—53, 1988.

M. Gardner. On playing the new eleusis, the game that simulates the search for truth. *Scientific American*, 237: 18—25, 1977.

D. Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7: 155—170, 1983.

Kenneth W. Haase Jr. Discovery systems. In *Proceedings of ECAI-86*, volume 1, pages 546—555, 1986.

Douglas R. Hofstadter. The architecture of jumbo. In R. S. Michalski, editor, *Proceedings of the International Machine Learning Workshop*, pages 161—170, 1983.

Douglas R. Hofstadter. On the seeming paradox of mechanizing creativity. In *Metamagical Themas*, chapter 23, pages 526—546. Penguin, 1986a.

Douglas R. Hofstadter. Waking up from the Boolean dream. In *Metamagical Themas*, chapter 26, pages 631—665. Penguin, 1986b.

John H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, chapter 20, pages 593—623. Morgan Kauffmann Publishers Inc., 1986.

P. N. Johnson-Laird. Reasoning, imagining and creating. *Bulletin of the British Psychological Society*, 40: 121—129, 1987.

M. Keane. Analogical mechanisms. *Artificial Intelligence Review*, 2: 229—251, 1988.

P. Langley, H. A. Simon, G. L. Bradshaw, and J. M. Zytkow. *Scientific Discovery*. MIT Press, Camb. Mass., 1987.

Christopher Langton, editor. *Artificial Life*. Addison-Wesley, Redwood City, California, 1989.

D. Lenat. The nature of heuristics. *Artificial Intelligence*, 19, 21, 1982.

D. Lenat. Why AM and eurisko appear to work. In *Proceedings of the American Association of Artificial Intelligence*, pages 236—240, 1983.

C. Martindale and D. Hines. Creativity and cortical activation during creative, intellectual and EEG feedback tasks. *Biological Psychology*, 3: 71—80, 1975.

C. Martindale. *Cognition and Consciousness*. Dorsey Press, 1981.

James. R. Meehan. Tale-spin, an interactive program that writes stories. In *5th International Joint Conference on Artificial Intelligence*, pages 91—98, 1977.

Marsha Jean Ekstrom Meredith. *Seek-Whence: a Model of Pattern Perception*. PhD thesis, Indiana University, 1986.

Marvin Minsky. *The Society of Mind*. Picador, 1985.

Melanie Mitchell and Douglas R. Hofstadter. The emergence of understanding in a computer model of concepts and analogy-making. *Physica D*, 42: 322—334, 1990.

Erik Thomas Mueller. *Day-dreaming and Computation: A Computer Model of Everyday Creativity, Learning and Emotions in the Human Stream of Thought*. PhD thesis, University of California, Los Angeles, 1987.

Ajit Narayanan. What is it like to be a machine? In S. Torrance, editor, *Mind and the Machine*. Ellis Horwood, 1983.

R. M. Olton. Experimental studies of incubation: Searching for the elusive. *Journal of Creative Behaviour*, 13: 9—22, 1979.

D. Pastre. MUSCADET: An automatic theorem proving system using knowledge and metaknowledge in mathematics. *Artificial Intelligence*, 38(3): 257—318, 1989.

Jordan Pollack. Connectionism: Past, present, and future. *Artificial Intelligence Review*, 3(1): 3—22, 1989.

Racter. *The Policeman's Beard is Half-constructed.* Warner Books, New York, NY, 1984. Racter is a program written by William Chamberlain.

J. D. Read and D. Bruce. Longitudinal tracking of difficult memory retrievals. *Cognitive Psychology*, 14: 280—300, 1982.

G. D. Ritchie and F. K. Hanna. AM: A case study in AI methodology. In Partridge and Wilks, editors, *The Foundations of AI: A sourcebook.* Cambridge University Press, 1990.

Jonathan Rowe. *Emergent Creativity: A Computational Study.* PhD thesis, Computer Science Department, University of Exeter, UK, 1991.

David E. Rumelhart. Notes on a schema for stories. In Collins, editor, *Representation and Understanding: Studies in Cognitive Science.* Academic Press, 1975.

Roger C. Schank and Robert P. Abelson. *Scripts, Plans, Goals and Understanding.* Erlbaum, Hillsdale NJ, 1977.

Roger C. Schank. *Explanation Patterns.* Lawrence Erlbaum Associates, 1986.

Mark J. Steedman. A generative grammar for jazz chord sequences. *Music Perception*, 2: 52—77, 1984.

Robert Weisberg. *Creativity: Genius and Other Myths.* W. H. Freeman, 1986.

M. Yazdani. A computational model of creativity. In Richard Forsyth, editor, *Machine learning: principles and techniques.*, chapter 9, pages 171—183. Chapman and Hall, 1989.