The Parallel Terraced Scan: An Optimization for an Agent-Oriented Architecture

John Rehling and Douglas Hofstadter
Center for Research on Concepts and Cognition
Indiana University
510 North Fess Street
Bloomington, Indiana 47408-3822
USA
rehling@cogsci.indiana.edu dughof@cogsci.indiana.edu
http://www.cogsci.indiana.edu

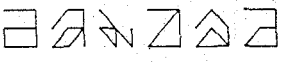

Figure 1: Six 'a's Rendered on the Grid.

Abstract — A relatively new area of research concerns computer programs which operate by the action of many small agents, rather than the serial execution of an algorithm. Because they differ drastically from conventional programs, new approaches are required for developing and maintaining them. An optimization technique, the Parallel Terraced Scan, has been applied to one such program: a letter-recognition system called the Examiner [7]. The Parallel Terraced Scan is the exploration of many possible paths, but with more computation devoted to paths which are identified as being more promising. Thus, it resembles pruning of search trees in some respects, but it does not completely abandon paths which are tentatively identified as less fruitful.

I. INTRODUCTION

The work described here concerns modifications made to a program called the Examiner, intended to be one module for use in a more complex program, Letter Spirit, which is currently under development [4]. The Letter Spirit project is concerned with the creation of novel, stylistically-consistent typefaces as rendered on a medium-resolution (3x7) grid of 56 unit-length quanta, with great diversity allowed in the particular stylistic properties of letterforms in the data set [3]. Six examples of the letter 'a' rendered on this grid appear in Figure 1.

The Examiner's architecture has much in common with the Copycat [9] and Tabletop [2] programs. All three are intended to capture certain key aspects of human cognition within their given domains, and, among other distinctive features they have in common, carry out nearly all processing via the action and interaction of many very small procedures, called Codelets. The Hearsay-II program [1], the first blackboard system, provided inspiration for this aspect of Copycat, Tabletop and the Examiner. These programs provide greater flexibility than many traditional problem-solving programs, and — beyond their interest as cognitive models — may be the best way to approach certain practical problems. Creating systems of this kind is a considerably different task from traditional programming, and a new repertoire of techniques for developing them is needed. This work describes an optimization for such architectures which can increase the speed, accuracy and quality of output of such programs, and is, in addition, suggestive of certain qualities of human cognition.

II. THE EXAMINER: AN AGENT-ORIENTED ARCHITECTURE

A task related to Optical Character Recognition is the categorization of letters rendered on the Grid. Many letterforms of each category may exist, and the task of interest here is the categorization of novel letterforms on the Grid, henceforth called gridletters. Creating a system that performs this task well is not straightforward. Several approaches are documented in [7]. One of these, the Examiner, is the starting point for the optimization technique described herein. The Examiner's architecture operates very roughly as follows:

- The Workspace is the data structure where a structured parsing of the letter may be built up. The quanta are collected into subsets, which are contiguous parts of the letterform. Each part has potentially several semantically meaningful labels, such as "short", or "ascender". A part may be bound to a role, which is an abstraction of a portion of a letter, such as "left-post", "dot", or "crossbar".
- The Conceptual Memory is a localist network containing nodes for each possible role, and each possible role-set, or "whole". Links connect role-sets to their constituent roles. For example, the node representing "f" has links to the nodes for "crossbar" and "f-post". Each node has an activation value which can vary between -100 and +100, and each link has a fixed, permanent strength indicating the propensity of activation to spread from one node to another. When a part is bound to a role, the concept node for that role will receive a large amount of positive activation. This process is called "sparking".
- The Coderack is a list of Codelets, essentially procedure calls which are placed on the Coderack by other Codelets, for possible selection and execution later, rather than being run immediately. The order of execution is probabilistic, each Codelet's chances being weighted by an urgency fixed when the Codelet is posted. Codelets are removed from the Coderack and run one at a time. This allows many paths of activity to operate in parallel, each having some

0-7803-4253-4/97/$10.00 © 1997 IEEE

Codelets on the Coderack. Codelets fall into many types, such as those responsible for creating and altering parts, attaching semantic labels to parts, binding parts to roles thereby positively activating those roles, spreading activation in the Conceptual Memory, and activating role-sets directly based upon global properties of the letterform.

- The Temperature is a number from 0 to 100. This is an inverse "goodness" rating for the quality of the work done thus far in a run, with high Temperature corresponding to situations where the system has not yet built up much useful structure. Temperature focuses the behavior of Letter Spirit by determining how much weight to give each codelet when the virtual roulette wheel is spun which picks the next codelets to be run. This makes behavior more directed when an answer seems nearer, and more likely to consider a wide range of options when it seems that little useful progress has been made. This phenomenon can be seen in many complex systems, from computer models using simulated annealing to the choices made by politicians.

In an ideal run, the Examiner recognizes a letter as follows. First, the gridletter is parsed into non-overlapping parts. A 'd', for example, could be parsed into two parts, one corresponding to something tall and thin on the right, and an open bowl similar to a lone letter 'c'. The exact shapes of these parts will vary depending upon the gridletter. Given a parsing, the Examiner tries to label the parts. Labels are properties that can apply to a shape. These labels, (for instance, "tall"), are attached probabilistically, with a better probability of a label being applied to more appropriate parts. When a part has some labels, sparking can occur. Sparking is a process in which a part is used to activate the concept nodes for roles. The definition of a role includes a collection of labels, and the extent to which a part sparks a role is determined by how well the sets of labels of the gridbound part and the abstract role correspond. Activation is then spread, so that a whole receives activation in proportion to the sum of the activation of its constituent roles. Any wholes with positive activation are subject to R-Role checking. R-Roles, or Relational Roles, are norms for where, if at all, role-fillers should touch, how tips at the ends of parts should be positioned in relation to one another, and so on. If the Temperature is low enough, then the program may halt. Temperature is lowest when exactly one whole has high activation. The lower the Temperature, the greater the probability is that the system will halt, returning the most-highly activated whole as its answer. At any point in processing, the system may also decide to change its parsing of the letterform into a different set of component parts. This will be likely to occur if a part has received many labels but fails to spark any roles. If the problematic part is small, then it will be absorbed into a neighboring part, and if it is large, it will be broken into multiple smaller parts. All of these decisions are made probabilistically, so that, if the same situation recurs, different choices may be made in successive runs. At fixed deadlines (defined in terms of number of Codelets run), the standards the program has for sparking are made less stringent. Therefore, an unusual part may not be able to spark any roles early in a run, but able to, using the looser standards, later on. This approach, called role loosening, is favored, rather than having roles very loose initially, so that strange answers do not supercede correct ones in the early stages of a run. In later stages, when it seems unlikely that the gridletter is a typical example of any category, role loosening allows the system to explore more unusual possibilities.

Small steps in the Examiner's processing (the labelling of a part with one label, the sparking of roles with a part, the spread of activation, the re-parsing of the letterform, etc.) are carried out by individual Codelets. Because the selection of the next Codelet to be run is made probabilistically, the Examiner may explore many different parsings of a letterform, and the set of labels a part has acquired will also have many possibilities. This is a great strength of the system. If the method of parsing and labelling were deterministic, then the system would only be able to recognize letterforms whose style was not too strange for the program's rules and definitions. The probabilistic nature of the system allows it to explore various possibilities, but, because it is not totally random, it is unlikely to produce very strange answers (such as identifying a typical 'o' as an 'x'). The exact course of a run depends upon many factors. Strange letters will usually need to be re-parsed many times, and the runs may take thousands of Codelets. Plain letters are usually recognized in a few dozen Codelets, and are sometimes recognized correctly on the first parsing.

The Examiner is a robust letter-recognizer which achieves a high rate of correct identification over stylistically varied letterforms. Its performance is favorable in comparison with other letter-recognition architectures and resembles the behavior of humans when given the same task.

III. THE PARALLEL TERRACED SCAN

The idea behind the Parallel Terraced Scan is to explore many possibilities at a time, but to devote more computational power to the directions which are more promising. Thus, progress is made towards the eventual answer, regardless of which among the many possibilities it may be. However, the possibilities are not treated equally, and the average running time is greatly reduced from exhaustive search. This is reminiscent to work on the k-armed bandit problem [5].

The original Examiner searched multiple possibilities in parallel. For example, if a gridletter is parsed into three parts, then any of the 97,290 ways in which they could be bound to three of the forty-seven roles could follow. Different parsings could allow even more possibilities. And the binding of parts to roles is only part of a process meant to lead to the activation of one whole. Numerous potential outcomes (perhaps numbering in the millions) will be active possibilities during the middle of a run. If each possibility were investigated to an equal extent, then the program would have exponential running time. At the other extreme, if only one path to an answer were possible, then the system would suffer the rigidity that

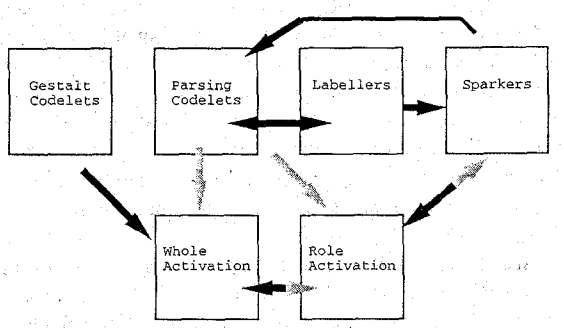

Figure 2: Flow of Influence in the Examiner.

is common in symbolic AI programs. The compromise is to be able to explore many possibilities, and at any given point in time to be actively considering several, with an emphasis on those which *a priori* seem most likely to be correct. The focus on the better possibilities is accomplished by many means. Parts are more likely to receive certain labels than others. Codelets tend to post new Codelets that pursue the same directions. The use of Temperature to weigh the impact of urgencies on Codelet selection focuses the system on better possibilities. Thus, by performing a weighted consideration of many possibilities at once, the original Examiner already provides an example of the Parallel Terraced Scan.

The optimization was not based, therefore, on introducing the Parallel Terraced Scan to the Examiner, but on making the system have an even greater focus on the best possibilities, enhancing the extent to which the Parallel Terraced Scan prunes the vast tree of possibilities. The primary goal was to increase the speed of the program, but maximizing the accuracy of its answers was a secondary goal. The major idea was to use the activations of concepts in the Conceptual Memory as the means for an increased focus on more appropriate roles. The optimization, thus, had two sub-goals. The first was to make the activations of a role at all times as consistent as possible with the likelihood that the role is represented in the gridletter being recognized. The second was to involve the activation of roles with sparking, so that a role with high activation will have a better chance of being involved in sparking, and one with low activation little or no chance. The correspondence between a role's definition and a part's labels remains the primary factor in sparking, but can be influenced significantly by the role's activation.

To make activations more meaningful in terms of expressing the importance of each role at each point in the processing, several changes had to be made to the system. Originally, activation of a given role or role-set could change drastically, rising and falling in large jumps. In the optimization, it was attempted to insure that activations would change in small steps, and always reflect the likelihood of that concept being involved in the current letterform. The optimizations are as follows:

- Re-parsings immediately clear all activations in the Conceptual Memory, so that leftover activations for parts that no longer exist do not muddle further processing.
- Each time the parsing of the letterform changes, a Gestalt Codelet sends activation to each role-set, based on the shape of the entire gridletter. This is posted with high urgency so that it will run soon and be able to influence future processing beneficially. When the Gestalt codelet has run, only wholes which are likely to be legitimate possibilities for the letterform have positive activation. This may reduce the number of candidate answers on the level of letter category from 26 to a much smaller pool, rarely more than 4. Extensive experimentation was carried out to find a good, fast Gestalt function the output of which would be a good heuristic for the actual answers. In the original Examiner, Gestalt Codelets were liable to run at any stage of the processing, abruptly changing the activations of wholes.
- The R-Role Checker Codelet was eliminated. This Codelet was originally posted for every whole with positive activation. The delay between posting and running an R-Role Checker allowed poor wholes to retain high activation for arbitrarily long spans of time before the necessary decrease in the wholes' activation would occur. All the activity of the system was potentially misdirected until R-Role Checking occurred. This checking was added directly to the Activation-Spreading Codelet, only wholes with activations greater than +20 are checked. This change reduces the number of Codelets run for the same amount of work, but reduces the total amount of processing because fewer wholes need to be checked. Although this may slightly distort the statistics given in the RESULTS section on the number of Codelets run, the number of R-Role Checker Codelets in a run was typically a small fraction of the whole, and more harm was done by the delay between their posting and execution than in the time spent running them.
- Originally, the activation of each role and whole's node was set to the previous activation plus the sum of the weighted inputs from each connected node, minus a decay factor. Activation could only pass from a whole down to a role, however, if the activation of the whole was above +75. A large jolt of activation could stay with a node for a long time, and would only eventually decay. In the optimization, at the time that activation spreads, a node is allowed to retain only a small portion of its previous activation, in addition to what activation is spread to it via its inputs. For a whole, the sum of the weighted inputs from each connected role node is added in. For a role, the maximum of the weighted inputs from the connected whole nodes is added. The discrepancy is easily explained. A whole receives activation from roles only to the extent that each of its component roles is active. A role, however, receives activation from wholes to the extent that any of the wholes it may be a member of is active. Logically, a whole is present only if all of its associated roles are. A role is present if any

one of its associated wholes is.

- The onset of phases of loosening was made probabilistic. Before, distinct phases of loosening were inforced on a fixed schedule. Now, a short first phase maintains the tightest roles, and most letterforms are recognized during this phase. Subsequent phases randomly toggle between higher and lower looseness settings. Thus, letterforms that are only identified with loose roles may be recognized relatively quickly, while a letterform that requires tight roles will have many chances later, if it is not recognized in the first phase.
- The sparking of roles with parts was made to be influenced by the activations of the roles. Roles with higher activations are given higher priority in the decision of which roles to spark with a part (however, this is not the case for roles which already have a part bound to them). This is perhaps the most important optimization, and many of the other modifications were necessary so that the activation of a role is, at all times, a good indicator that the role should be considered as relevant to the gridletter being recognized. Roles with negative activation are not considered at all for sparking, so it is very important that a role receive negative activation only if it is exceedingly improbable that it is a component of the correct answer's whole.

The primary strategy behind the optimizations can be seen in Figure 2. Here, the primary components of the model (in terms of Codelets and nodes in the Conceptual Memory), as already described above, are shown schematically. An arrow indicates that one component of the system may influence the one the arrow points to. Influence may take the form of updating activations, posting Codelets, or influencing the behavior of Codelets. Black arrows indicate aspects of the original Examiner; each of these continued to exist (although perhaps in a different form) after optimization. Gray arrows indicate those aspects of the system added in the optimization. No black arrow was drawn from whole activations to role activations, because activation spreading of this sort, while possible in the original Examiner, was extremely rare.

Considering only the black arrows, we can see that there is a loop between the Codelets involved with parsing, labelling parts, and sparking roles, but from there, all activity is feedforward. That subsystem influences role activations, which in turn influence whole activations. In addition, Gestalt Codelets may directly alter the activations of wholes.

In the optimized system, considering both the black and gray arrows, we see that there are loops allowing each of the components to influence any other (except Gestalt), even if only indirectly. The interaction indicated by these loops constitute the most important aspects of the optimization. Of particular note is the bottom-up/top-down interaction between the two levels of the Conceptual Memory. By allowing the low and high levels of a conceptual hierarchy to influence each other, this is an application of top-down pressure of the kind described in the interactive activation model of [6]. A whole can receive activation via Gestalt, or from one or more

Figure 3: Six 'b's from EASY.

of its constituent roles being sparked. Activation (positive or negative) will then spread back down to roles, including those which have not yet been involved in sparking. Subsequently, when sparking occurs, the roles that have received higher activation from their wholes will be favored. Thus, when the system suspects the presence of a certain whole, it will try to complete the figure (if possible) by looking for parts for the remaining roles involved in that whole.

The basic principles that emerge are that it is beneficial to use information which is easy to calculate, or has already been calculated for another purpose, to save in subsequent computational effort. This principle could be carried out further in the Examiner. Whole and role activations could influence parsing so that quanta are grouped into parts that are likely to be involved in the gridletter. While this optimization is not planned, it could improve performance, particularly for gridletters which are difficult to parse correctly.

IV. RESULTS

The primary measure of performance was the average number of codelets per run in a test set. For purposes of debugging and parameter tuning, a small set of 52 gridletters, henceforth called TINY, representing each category twice, was used. A larger test set of 544 gridletters, called TEST, was used to demonstrate the real progress of the system. If a small test set were used for all debugging and testing, then it would be possible that the system had been tuned to work well only for those gridletters. Meanwhile, a large test set involved in the debugging stage would make testing excessively slow. TINY was used to implement all major features of the optimization.

The only modifications made using TEST as a test set were to the deadlines used to initiate the various stages of role loosening and the inclusion of an additional R-Role checker to help discriminate 'g' and 'q'. The effect of modifying deadlines was to find a good balance in the tradeoff between speed and accuracy. TEST was further broken into two subsets, EASY and HARD. The 388 gridletters in EASY were meant to be more typical of their intended categories, while the 156 members of HARD were intuitively stranger and less typical of their categories, and therefore harder to recognize quickly and correctly. The intrinsic difficulties of the subsets of TEST were shown in experiments with human subjects (with the subsets going by the names NORMALS and FONTS, respectively) as shown in [8]. Figures 3 and 4 show examples of the letter 'b' to demonstrate the contrast between EASY and HARD.

The results of the optimization can be seen in Table 1. The raw data for the pre-optimization Examiner are given under OLD, and the optimized version under NEW. The factor of

	CODELETS/RUN			CORRECT%		
SET	OLD	NEW	Δ	OLD	NEW	Δ
TEST	1381.3	624.8	2.2	82.6	85.3	2.7
PLAIN	770.2	390.9	2.0	95.7	96.8	1.1
HARD	2901.1	1206.7	2.4	49.9	56.8	6.9
TINY	1427.8	430.2	3.3	89.0	90.8	1.8

Table 1: Performance of the Examiner.

Figure 4: The six 'b's from HARD.

increase in speed and percent increase in accuracy are given in the Δ columns. Speed is indicated with the number of Codelets per run. In all instances, the performance following the optimizations was improved.

V. CONCLUSIONS

As shown above, the optimization roughly doubled the speed of the Examiner on the TEST data set, while also increasing the rate of correct identification. The optimized system, throughout a run, takes advantage of prior computation by using values computed in earlier stages of processing to wisely direct future processing. In particular, the use of Gestalt illustrates the value of this principle. The Gestalt Codelet is quick and easy to compute, and it usually does its work soon after a new parsing occurs, so that the small investment in computation involved in running the Gestalt Codelet can lead to large savings in total run time by eliminating most of the possible wholes from consideration immediately. In many ways, this is like the use of a heuristic to direct search in a symbolic AI program, but the Parallel Terraced Scan simply prioritizes the many possibilities it explores in parallel, rather than selecting or ordering possibilities searched serially.

The improvement in performance is more pronounced for the HARD set than for PLAIN, probably because there is more room for improvement. In extreme cases, the average speed-up for a particular gridletter can be by a factor of 20 or more. The even greater speed-up on the TINY set is probably due to the fact that many changes ended up being particularly well-suited for those particular gridletters.

The tasks which can be solved by systems using the Parallel Terraced Scan differ from many traditional programming situations. There is a very familiar paradigm in which a problem is guaranteed to have one correct answer, and it is the programmer's task to find an algorithm that will always find that answer. The domains of Copycat, Tabletop and the Examiner represent a different type of task, where multiple answers may exist, although some are better than others. (For the test sets used with the Examiner above, we assumed that each gridletter was a member of exactly one letter category, but ambiguous gridletters, which suggest two or more categories, certainly exist.) These domains may present a dilemma for programming with traditional algorithms, where a deterministic approach may lead to brittleness and exhaustive search is impractical. The architecture used in stochastic, agent-oriented systems such as the Examiner avoids both of these hazards. The approach may work in many areas where traditional AI has failed to meet with success. As research with agent-oriented systems continues, it will be important for techniques such as that described in this paper to be part of the implementations. Faster programs are easier to develop, because shorter times for testing and evaluating modifications can lead to much faster programming.

This work demonstrates the power of the Parallel Terraced Scan as a computational optimization; it is encouraging that a mechanism hypothesized to have a role in human cognition should also prove to be computationally beneficial.

VI. ACKNOWLEDGEMENTS

This research was supported by a Sun Microsystems Computer Company Academic Equipment Grant EDUD-NAFO 960418. Valuable proofreading was performed by Helga Keller and Janet Rehling.

VII. REFERENCES

[1] Erman, L. D., Hayes-Roth, F., Lesser, V. R. and Raj Reddy, D. (1980). The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213-253.

[2] French, R. M. (1992). *Tabletop: An emergent stochastic computer model of analogy-making*. Doctoral dissertation, University of Michigan, Ann Arbor.

[3] Hofstadter, D. R. (1985). *Metamagical themas*. New York: Basic Books.

[4] Hofstadter, D. R. and the members of FARG (1995). *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. New York: Basic Books.

[5] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

[6] McClelland, J. L. and Rumelhart, D. E. (1981). An interactive activation model of context effects in letter perception: Part 1. An account of basic findings. *Psychological Review*, 88:375-407.

[7] McGraw, G. (1995). *Letter Spirit (part one): Emergent high-level perception of letters using fluid concepts*. Doctoral dissertation, Indiana University, Bloomington, IN.

[8] McGraw, G., Rehling, J., and Goldstone, R. (1994). Letter perception: Toward a conceptual approach. *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*.

[9] Mitchell, M. (1990). *Copycat: A computer model of high-level perception and conceptual slippage in analogy making*. Doctoral dissertation, University of Michigan, Ann Arbor.