Analogy-Making as a Complex Adaptive System

Melanie Mitchell*
Biophysics Group
Los Alamos National Laboratory

To appear in L. Segel and I. Cohen (editors), *Design Principles for the Immune System and Other Distributed Autonomous Systems*. New York: Oxford University Press

1 Introduction

This paper describes a computer program, called Copycat, that models how people make analogies. It might seem odd to include such a topic in a collection of papers mostly on the immune system. However, the immune system is one of many systems in nature in which a very large collection of relatively simple agents, operating with no central control and limited communication among themselves, collectively produce highly complex, coordinated, and adaptive behavior. Other such systems include the brain, colonies of social insects, economies, and ecologies. The general study of how such emergent adaptive behavior comes about has been called the study of "complex adaptive systems".

The Copycat program is meant to model human cognition, and its major contribution is to show how a central aspect of cognition can be modeled as the kind of decentralized, distributed complex system described above. In doing so it proposes principles that I believe are common to all complex adaptive systems, and that are particularly relevant to the study of immunology.

Copycat was developed by Douglas Hofstadter and myself, and has been described previously in [3, 10, 11, 16, 17]. This paper summarizes these earlier works, and makes explicit links to the immune system.

2 Analogy-Making and Cognition

Analogy-making can be defined as "the perception of two or more non-identical objects or situations as being the 'same' at some abstract level." We chose to focus on analogy-making

*Address: P-21, MS D454, LANL, Los Alamos, NM 87545. Email: mmitchell@lanl.gov

Figure 1: A page of 'A's in different typefaces from a recent Letraset catalog. Reprinted from [8], p. 243.

because of its centrality to every aspect of cognition.

For example, analogy-making is at the core of recognition and categorization. Children learn to recognize instances of categories such as "dog" or "cat". Even though different dogs look very different, children perceive some essential sameness at an abstract level and can differentiate a dog from a cat. Likewise, children learn to recognize cats and dogs in books as well as in real life, even though on the surface such images are very different from one another and from the corresponding real-life creatures. Hofstadter has pointed out that even the ability to recognize the letter 'A' in many different typefaces and handwriting styles requires a highly sophisticated analogy-making ability [6]. For example, the collection of 'A's given in Figure 1, taken from a typeface catalog, illustrates the ease with which people can recognize vastly different shapes as instances of 'A' because of some essential abstract similarity. Hofstadter points out that "no single feature, such as having a pointed top or a horizontal crossbar (or even a crossbar at all) is reliable" as an indicator of being an 'A'. ([8], p. 242.)

At a more abstract level, people can easily recognize styles of music—"That sounds like Mozart"—or familiar music transported to a less familiar idiom—"Hey, that's a muzak version of 'Hey Jude'." When you think about it, these are examples of sophisticated analogy-making as well. Any two pieces by Mozart are superficially very different, but at some level we perceive a common essence. Likewise, the Beatles rendition of *Hey Jude* and the version you might hear in the supermarket have little in common in terms of instrumentation, tempo, vocals, and other readily apparent musical features, but we can easily recognize it as "the same song".

People make analogies all the time, both consciously and unconsciously. You've probably had the experience of hearing a friend's story about how her flight from Boston to San Francisco was delayed for four hours, and how her four pieces of luggage were lost. You exclaim "The same thing happened to me", thinking of your flight from Adelaide to Perth and how it was delayed for two hours and how two of your three pieces of luggage were lost. Not exactly the same thing, but close. Or you might have read about a war waged by the Soviets in Afghanistan in which their superior military power was thwarted again and again by the determination of a small Western-supported army fighting on its own soil, and been instantly reminded of a war the Americans waged during the 60s and 70s in Asia against a small Soviet-supported army fighting on its own soil; you might have even thought, "This is another Vietnam". Again, it's not exactly the same, but basically. It is that "basically" where analogy lies. Anytime you recognize something, are reminded of something, or see a similar essence in two different situations, you are making an analogy. (For an excellent discussion of analogies, conscious and unconscious, in human thought, see [3].) Such abstract analogies come about by what might be called *high-level perception*, in which objects, pieces of music, memories, or complex situations are viewed in the mind's eye and found to be similar to one another.

It should be clear from all these examples that in making analogies, elements of one situation are fluidly mapped to another situation. A four-hour flight delay in Boston is easily mapped to a two hour flight delay in Adelaide or perhaps even a six hour train stopover in Providence. The parallel diagonal crossbar in the Tintoretto face (third row, second

column of Figure 1) is easily seen to correspond to the curved and striped crossbar/foot of the Stripes face (seventh row, sixth column of Figure 1). The lilt and clean lines of Mozart's *Eine Kleine Nachtmusik* is easily seen to be similar to the style of his *Divertimento in D*. The ability for concepts to "slip" from situation to situation in this fluid way is a hallmark of human thought and is one of the salient differences between human intelligence and the rigid literality of computers. Our goals in the Copycat project are to understand how human concepts attain this fluidity and how to impart such fluidity to computers.

3 Idealizing Analogy-Making

As a first step in modeling analogy-making in a computer, Hofstadter devised a "microworld" consisting of analogies between strings of letters [3]. This microworld captures many of the features of analogy-making described above, in an idealized fashion.

For example, consider the following problem: if **abc** changes to **abd**, what is the analogous change to **ijk**? Most people describe the change as something like "Replace the rightmost letter by its alphabetic successor", and answer **ijl**. But clearly there are many other possible answers, among them:

- **ijd** ("Replace the rightmost letter by a d"),
- **ijk** ("Replace all c's by d's; there are no c's in **ijk**"), and
- **abd** ("Replace any string by **abd**").

There are, of course, an infinity of other, even less plausible answers, such as **ijxx** ("Replace all c's by d's and all k's by two x's"), and so on, but almost everyone immediately views **ijl** as the best answer. This being an abstract domain with no practical consequences, I may not be able to convince you that **ijl** is a better answer than, say, **ijd** if you really believe the latter is better. However, it seems that humans have evolved in such a way as to make analogies in the real world that affect their survival and reproduction, and their analogy-making ability seems to carry over into abstract domains as well. This means that almost all of us will, at heart, agree that there is a certain level of abstraction that is "most appropriate", and here it yields the answer **ijl**. Those people who truly believe that **ijd** is a better answer would probably, if alive during the Pleistocene, have been eaten by tigers, which explains why there are not many such people around today.

The knowledge available to an analogy-maker in this microworld is fairly limited. The 26 letters are known, but only as members of a Platonic linear sequence; shapes of letters, capital versus lower case, sounds, words, and all other linguistic and graphic facts are unknown. The only relations explicitly known are predecessor and successor relations between immediate neighbors in the alphabet. Ordinal positions in the alphabet (e.g., the fact that *S* is the 19th letter) are not known. (In this paper I denote "Platonic" letters by italic capitals—"*S* is the 19th letter". I denote their instances by non-italic lower case–"the a in **abc** is the leftmost letter in its string". Strings of letters appearing in analogy problems are in boldface.)

A and *Z*, being alphabetic extremities, are salient landmarks of equal importance. The alphabet is not circular; that is, *A* has no predecessor and *Z* has no successor. The alphabet is known equally well backward and forward (i.e., the fact that *N* is the letter before *O* is as accessible as the fact that *P* is the letter after *O*). In addition, strings (such as **abc** or **kkjjii**) can be parsed equally well from left to right and from right to left. The analogy-maker can count, but is reluctant to count above 3 or so, and has a common-sense notion of grouping by sameness or by alphabetical adjacency (forward or backward with equal ease).

With these restrictions in mind, let's proceed to a second analogy problem: if **abc** changes to **abd**, what is the analogous change to **iijjkk**? The **abc** ⇒ **abd** change can again be described as "Replace the rightmost letter by its alphabetic successor", but if this rule is applied literally to **iijjkk** it yields answer **iijjkl**, which doesn't take into account the double-letter structure of **iijjkk**. Most people will answer **iijjll**, implicitly using the rule "Replace the rightmost *group of letters* by its alphabetic successor," letting the concept *letter* of **abc** slip into the concept *group of letters* for **iijjkk**.

Another kind of conceptual slippage can be seen in the problem

abc ⇒ **abd**
kji ⇒ ?

A literal application of the rule "Replace the rightmost letter by its alphabetic successor" yields answer **kjj**, but this ignores the reverse structure of **kji**, in which the increasing alphabetic sequence goes from right to left rather than from left to right. This puts pressure on the concept *rightmost* in **abc** to slip to *leftmost* in **kji**, which makes the new rule "Replace the *leftmost* letter by its alphabetic successor", yielding answer **lji**. This is the answer given by most people. Some people prefer the answer **kjh**, in which the sequence **kji** is seen as going from left to right but decreasing in the alphabet. This entails a slippage from "alphabetic successor" to "alphabetic predecessor", and the new rule is "Replace the rightmost letter by its alphabetic *predecessor*".

It should be clear by now that the key to analogy-making in this microworld (as well as in the real world) is what I am calling *conceptual slippage*. Finding appropriate conceptual slippages given the context at hand is the essence of finding a good analogy. The Copycat program is a model of how concepts slip in response to pressures brought about by ongoing perception of a situation. Copycat is the successor to two previous programs that modeled high-level perception and conceptual slippage: Jumbo, which produced English-like anagrams [4], and Seek Whence, which searched for patterns underlying numerical sequences [5, 15].

As a prelude to developing Copycat, we created thousands of letter-string analogy problems to explore what kinds of slippages come about in response to different kinds of perceptual pressures. Two more examples will be instructive.

Consider

abc ⇒ **abd**
mrrjjj ⇒ ?

.

You want to make use of the salient fact that **abc** is an alphabetically increasing sequence, but how? This internal "fabric" of **abc** is a very appealing and seemingly central aspect of the string, but at first glance no such fabric seems to weave **mrrjjj** together. So either (like most people) you settle for **mrrkkk** (or possibly **mrrjjk**), or you look more deeply. The interesting thing about this problem is that there happens to be an aspect of **mrrjjj** lurking beneath the surface that, once recognized, yields what many people feel is a more satisfying answer. If you ignore the *letters* in **mrrjjj** and look instead at *group lengths*, the desired successorship fabric is found: the lengths of groups increase as "1-2-3". Once this connection between **abc** and **mrrjjj** is discovered, the rule describing **abc** ⇒ **abd** can be adapted to **mrrjjj** as "Replace the rightmost *group of letters* by its *length* successor", which yields "1-2-4" at the abstract level, or, more concretely, **mrrjjjj**.

Finally, consider

abc	⇒	**abd**
xyz	⇒	?

At first glance this problem is essentially the same as the problem with target string **ijk** given above, but there is a snag: *Z* has no successor. Most people answer **xya**, but in Copycat's microworld the alphabet is not circular and thus the program could not come up with this answer. We intentionally excluded it because one of the goals of the project is to model the process by which people deal with impasses. This problem forces an impasse that requires analogy-makers to restructure their initial view, possibly making conceptual slippages that were not initially considered, and thus to discover a different way of understanding the situation.

People give a number of different responses to this problem, including **xy** ("Replace the z by nothing at all"), **xyd** ("Replace the rightmost letter by a d"; given the impasse, this answer seems less rigid and more reasonable than did **ijd** for the first problem above), **xyy** ("If you can't take the z's *successor*, then the next best thing is to take its *predecessor*"), and several other answers. However, there is one particular way of viewing problem 7 that, to many people, seems like a genuine insight, whether or not they come up with it themselves. The essential idea is that **abc** and **xyz** are "mirror images"—**xyz** is wedged against the end of the alphabet, and **abc** is similarly wedged against the beginning. Thus the z in **xyz** and the a in **abc** can be seen to correspond, and then one naturally feels that the x and the c correspond as well. Underlying these object correspondences is a set of slippages that are conceptually parallel: *alphabetic-first* ⇒ *alphabetic-last*, *rightmost* ⇒ *leftmost*, and *successor* ⇒ *predecessor*. Taken together, these slippages convert the original rule into a rule adapted to the target string **xyz**: "Replace the *leftmost* letter by its *predecessor*". This yields a surprising but strong answer: **wyz**.

It is important to emphasize once again that the goal of this project is not to model specifically how people solve these letter-string analogy problems (it is clear that the microworld involves only a very small fraction of what people know about letters and what knowledge they might use in solving these problems), but rather to propose and model mechanisms for

high-level perception and analogy-making in general. Analogy-making can be characterized very broadly as distilling the *essence* of one situation and *adapting* it (via conceptual slippage) to fit another situation. The letter-string analogy problems were designed to isolate and make very clear some of the mental abilities that are required for this process of understanding and perceiving similarity between situations. These abilities include the following (which, though listed separately, are of course strongly interrelated):

- Mentally constructing a coherently structured whole out of initially unattached parts.
- Describing objects, relations, and events at the appropriate level of abstraction.
- Grouping certain elements of a situation while viewing others individually.
- Focusing on relevant aspects and ignoring irrelevant or superficial aspects of situations.
- Taking certain descriptions literally and letting others slip when perceiving correspondences between aspects of two situations.
- Exploring many avenues of possible interpretations while avoiding a search through a combinatorial explosion of possibilities.

The rest of this paper describes how these abilities are modeled in Copycat.

4 Dynamics of Exploring Ways of Understanding Situations

When given a situation with many components and potential relations among components, be it a visual scene, a friend's story, or a scientific problem, how does a person (or how might a computer program) mentally explore the typically intractably huge number of possible ways of understanding what is going on and possible similarities to other situations?

The following are two opposite and equally implausible strategies, both to be rejected:

1. Some possibilities are *a priori* absolutely excluded from being explored. For example, after an initial scan of **mrrjjj**, make a list of candidate concepts to explore (e.g., *letter*, *letter group of letters*, *successor*, *predecessor*, *rightmost*) and rigidly stick to it. The problem with this strategy, of course, is that it gives up flexibility. One or more concepts not immediately apparent as relevant to the situation (e.g., *group length*) might emerge later as being central.
2. All possibilities are equally available and easy to explore, so one can do a "full-width" search through all concepts and possible relationships that would ever be relevant in any situation. The problem with this strategy is that in real life there are always too many possibilities, and it's not even clear ahead of time what might constitute a possible concept or relationship for a given situation. If you hear a funny clacking

noise in your engine and then your car won't start, you might give equal weight to the possibilities that (a) the timing belt has accidentally come off its bearings or (b) the timing belt is old and has broken. If for no special reason you give equal weight to the third possibility that your next-door neighbor has furtively cut your timing belt, you are a bit paranoid. If for no special reason you also give equal weight to the fourth possibility that the atoms making up your timing belt have quantum-tunneled into a parallel universe, you are a bit of a crackpot. If you continue and give equal weight to every other possibility... well, you just can't, not with a finite brain. But, on the other hand, there is some chance you might be right about the malicious neighbor, and the quantum tunneling possibility shouldn't be forever excluded from your cognitive capacities or you risk missing a Nobel Prize.

The moral is that all possibilities have to be potentially available, but they can't all be equally available. Counterintuitive possibilities (e.g., your malicious neighbor; quantum tunneling) must be potentially available but must require significant pressure to be considered (e.g., you've heard complaints about your neighbor; you've just installed a quantum tunneling device in your car; every other possibility that you have explored has turned out to be wrong).

The problem of finding an exploration strategy that achieves this goal has been solved many times in nature. One example is the way ant colonies forage for food. Many ants wander in random directions away from the nest. When one locates a food source (by sight or smell), it picks up some of the food and returns to the nest, leaving a pheremone trail. Other ants follow such trails when they encounter them, following the trails to the food sources and reinforcing the trails with more pheremone. In this way, the shortest trails leading to the best food sources attain the strongest scent, and increasing numbers of ants follow these trails. But at any given time, some ants are still following weaker, less plausible trails, and some ants are still foraging randomly, allowing for the possibility of new food sources to be found.

This is an example of what John Holland has called "the balance between exploration and exploitation" [12]. When promising possibilities are identified, they should be exploited at a rate and intensity related to their estimated promise, which is being continually updated. But at all times exploration for new possibilities should continue. The problem is how to allocate limited resources—be they ants or thoughts—to different possibilities in a dynamic way that takes new information into account as it is obtained. Ant colonies have solved this problem by having large numbers of ants follow a combination of two strategies: continual random foraging combined with a simple feedback mechanism of preferentially following trails scented with pheremone and laying down additional pheremone while doing so.

The immune system also seems to maintain a near optimal balance between exploration and exploitation. At any time large numbers of B lymphocytes with different receptors are available for matching potential antigens; these different receptor types are formed via random combinations of genetic material in B cell precursors. In this way the immune system uses randomness to attain the potential for responding to virtually any antigen it encounters. This potential is realized when an antigen activates a particular B cell and triggers the proliferation of that cell and the production of antibodies with increasing specificity for the antigen in question. Thus the immune system *exploits* the information it encounters in the

form of antigens by allocating much of its resources towards targeting those antigens that are actually found to be present. But it always continues to explore additional possibilities that it might encounter by maintaining its huge repertoire of different B cells. Like ant colonies, the immune system combines randomness with highly directed behavior based on feedback.

Holland formalized the exploitation versus exploration balance in terms of a "multi-armed bandit" and proved some theorems regarding the optimal allocation of resources in uncertain environments in which information is continually being obtained. He proposed these as candidate general principles for all adaptive systems [12]. Hofstadter proposed a similar, more specific scheme for exploring such environments: the "parallel terraced scan" [4]. In this scheme many possibilities are explored in parallel, each being allocated resources according to feedback about its current promise, whose estimation is updated continually as new information is obtained. Like in an ant colony or the immune system, all possibilities have the potential to be explored, but at any given time only some are being actively explored, and not with equal resources. When a person (or ant colony, or immune system) has little information about the situation facing it, the exploration of possibilities starts out being very random, highly parallel (many possibilities being considered at once) and "bottom-up": there is no pressure to explore any particular possibility more strongly than any other. As more and more information is obtained, exploration gradually becomes more focused (increasing resources are concentrated on a smaller number of possibilities), less random, and more "top-down": possibilities that have already been identified as promising are exploited. As in ant colonies and the immune system, in Copycat such an exploration strategy emerges from myriad interactions among simple, autonomous, and interacting components.

5 Overview of the Copycat Program

Copycat's task is to use the concepts it possesses to build perceptual structures—descriptions of objects, links between objects in the same string, groupings of objects in a string, and correspondences between objects in different strings—on top of the three "raw", unprocessed strings given to it in each problem. The structures the program builds represent its understanding of the problem and allow it to formulate a solution. Since for every problem the program starts out from exactly the same state with exactly the same set of concepts, its concepts have to be adaptable, in terms of their relevance and their associations with one another, to different situations. In a given problem, as the representation of a situation is constructed, associations arise and are considered in a probabilistic fashion according to a parallel terraced scan in which many routes toward understanding the situation are tested in parallel, each at a rate and to a depth reflecting ongoing evaluations of its promise.

Copycat's solution of letter-string analogy problems involves the interaction of the following components:

- The *Slipnet*: A network of concepts, each of which consists of a central node surrounded by potential associations and slippages. A picture of some of the concepts and relationships in the current version of the program is given in Figure 2. Each node in

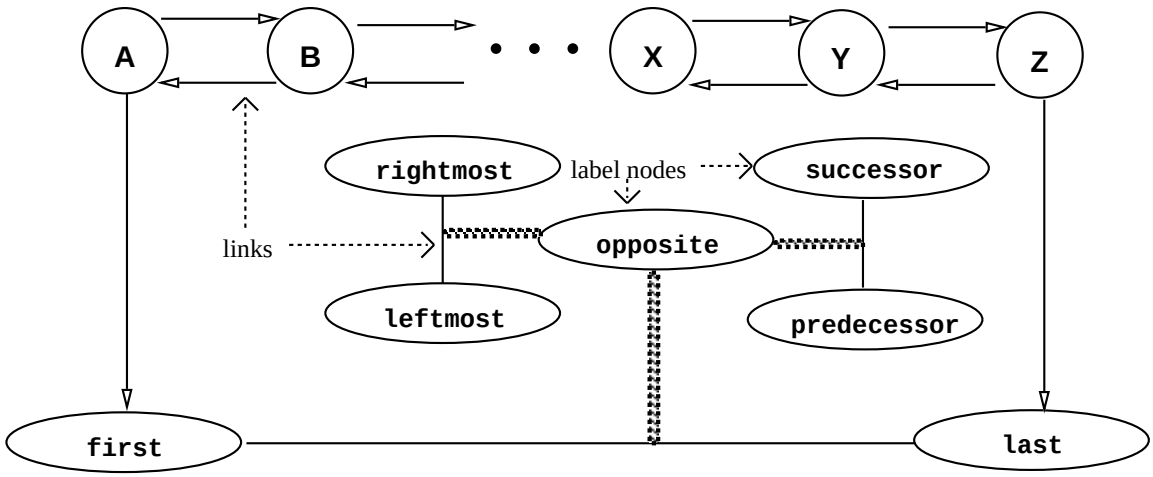


Figure 2: Part of Copycat's Slipnet. Each node is labeled with the concept it represents (e.g., *A–Z*, *rightmost*, *successor*, etc.). Some links between nodes (e.g., *rightmost–leftmost*) are connected to a label node giving the link's relationship (e.g., *opposite*). Each node has a dynamic activation value (not shown) and spreads activation to neighboring nodes. Activation decays if not reinforced. Each link has an intrinsic resistance to slippage, which decreases when the label node is activated.

the Slipnet has a dynamic *activation* value which gives its current perceived relevance to the analogy problem at hand, and which therefore changes as the program runs. Activation also spreads from a node to its conceptual neighbors, and decays if not reinforced. Each link has a dynamic *resistance* value which gives its current resistance to slippage. This also changes as the program runs. The resistance of a link is inversely proportional to the activation of the node naming the link. For example, when *opposite* is highly active, the resistance to slippage between nodes linked by opposite links (e.g., *successor* and *predecessor*) is lowered, and the probability of such slippages is increased.

- The *Workspace*: A working area in which the letters composing the analogy problem reside and in which perceptual structures are built on top of the letters.

- *Codelets*: Agents that continually explore possibilities for perceptual structures to build in the Workspace, and, based on their findings, attempt to instantiate such structures. (The term "codelet" is meant to evoke the notion of a "small piece of code", just as the later term "applet" in Java is meant to evoke the notion of a small application program.)

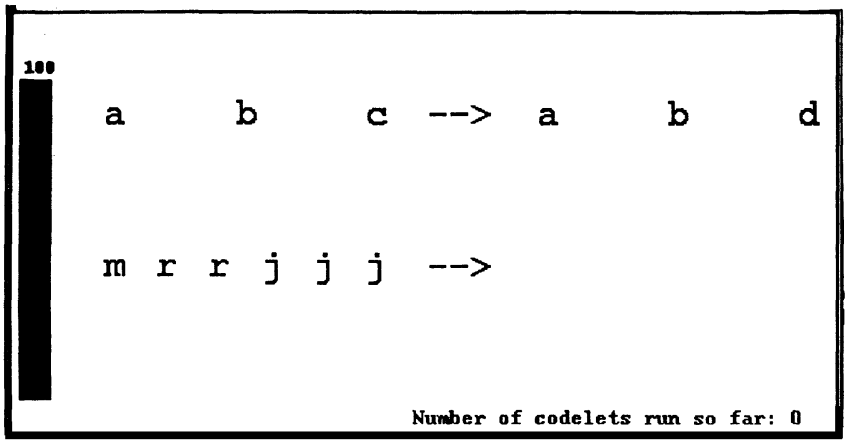
  Teams of codelets cooperate and compete to construct perceptual structures defining relationships between objects (e.g., "b is the successor of a in **abc**", or "the two i's in **iijjkk** form a *group*", or "the b in **abc** corresponds to the group of j's in **iijjkk**", or "the c in **abc** corresponds to the k in **kji**"). Each team considers a particular possibility for structuring part of the world, and the resources (codelet time) allocated to each team depends on the promise of the structure it is trying to build, as assessed dynamically

as exploration proceeds. In this way, a parallel terraced scan of possibilities emerges as the teams of codelets, via competition and cooperation, gradually build up a hierarchy of structures that defines the program's "understanding" of the situation with which it is faced.

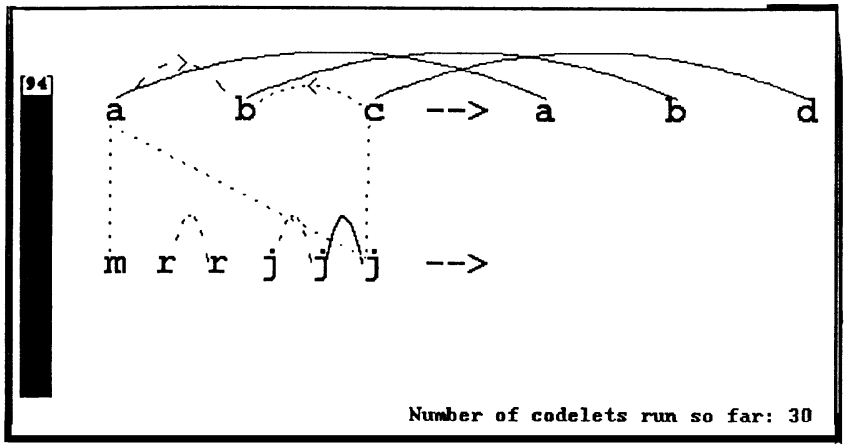
- *Temperature*, which measures the amount of perceptual organization in the system. As in the physical world, high temperature corresponds to disorganization, and low temperature corresponds to a high degree of organization). In Copycat, temperature both measures organization and feeds back to control the degree of randomness with which codelets make decisions. When the temperature is high, reflecting little perceptual organization and little information on which to base decisions, codelets make their decisions more randomly. As perceptual structures are built and more information is obtained about what concepts are relevant and how to structure the perception of objects and relationships in the world, the temperature decreases, reflecting the presence of more information to guide decisions, and codelets make their decisions more deterministically.

6 A Run of Copycat

The best way to describe how these different components interact in Copycat is to display graphics from an actual run of the program. These graphics are produced in real-time as the program runs. This section displays snapshots from a run of the program on **abc** ⇒ **abd**, **mrrjjj** ⇒ ? This is the same run that was described in [10]. For details about the implementation of the program, see [16]. The source code for Copycat, written in Common Lisp, is publicly available; see http://www.santafe.edu/~mm for instructions on how to get it.



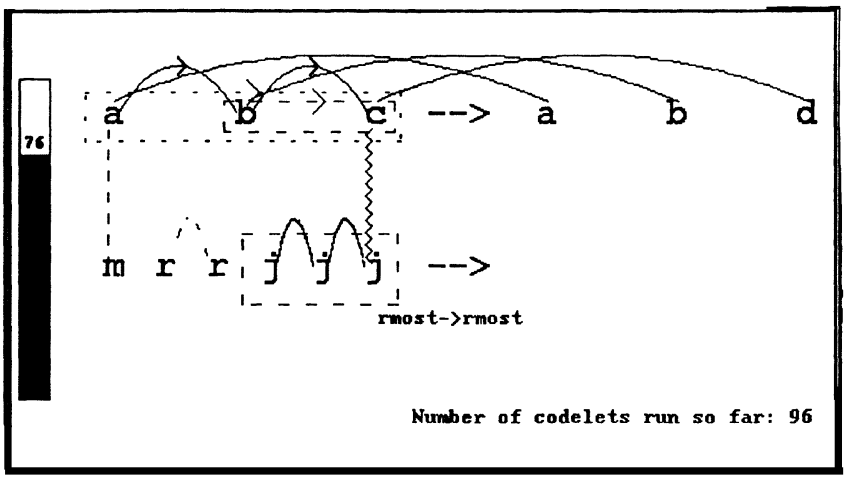
1. The problem is presented. The picture above displays the Workspace (here, the as-yet unstructured letters of the analogy problem); a "thermometer" on the left which gives the current temperature (initially set at 100, its maximum value, reflecting the lack of any perceptual structures); and the number of codelets that have run so far (zero).



2. Thirty codelets have run and have investigated a variety of possible structures. Conceptually, codelets can be thought of as ant-like agents, each one probabilistically following a path to explore but being guided by the paths laid down by other codelets. In this case the "paths" correspond to candidate perceptual structures. Candidate structures are proposed by codelets looking around at random for plausible descriptions, relationships, and groupings within strings, and correspondences between strings. A proposed structure becomes stronger as more and more codelets consider it and find it worthwhile. After a certain threshold of strength, the structure is considered to be "built" and can then influence subsequent structure building.

In the picture above, dotted lines and arcs represent structures in early stages of consideration; dashed lines and arcs represent structures in more serious stages of consideration; finally, solid lines and arcs represent structures that have been built. The speed at which proposed structures are considered depends on codelets' assessments of the promise of the structure. For example, the codelet that proposed the a–m correspondence rated it as highly promising because both objects are *leftmost* in their respective strings: identity relationships such as *leftmost* $\Rightarrow$ *leftmost* are always strong. The codelet that proposed the a–j correspondence rated it much more weakly, since the mapping it is based on, *leftmost* $\Rightarrow$ *rightmost*, is much weaker, especially given that *opposite* is not currently active. Thus the a–m correspondence will be investigated more quickly than the less plausible a–j correspondence.

The temperature has gone down from 100 to 94 in response to the single built structure, the "sameness" link between the rightmost two j's in **mrrjjj**. This sameness link activated the node *same* in the Slipnet (not shown), which creates top-down pressure in the form of specifically targeted codelets to look for instances of sameness elsewhere.



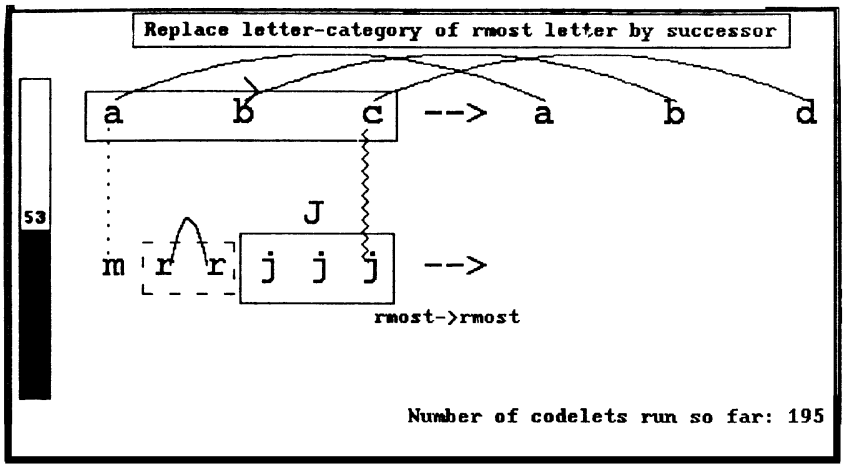
3. Ninety-six codelets have run. The successorship fabric of **abc** has been built. Note that the proposed c-to-b predecessor link of the previous picture has been out-competed by a successor link. The two successor links in **abc** support each other: each is viewed as stronger due to the presence of the other, making rival predecessor links much less likely to destroy the successor links.

Two rival groups based on successorship links between letters are being considered: bc and abc (a whole-string group). These are represented by dotted or dashed rectangles around the letters. Although bc got off to an early lead (it is dashed while the latter is only dotted), The group abc covers more objects in the string. This makes it stronger than **bc**—codelets will get around to testing it more quickly and will be more likely to build it than to build **bc**. A strong group, **jjj**, based on sameness is being considered in the bottom string.

Exploration of the crosswise a–j correspondence (dotted line in the previous picture) has been aborted, since codelets that further investigated it found it (probabilistically) too weak to be built. A c–j correspondence has been built (jagged vertical line); the mapping on which it is based (namely, both letters are *rightmost* in their respective strings) is given beneath it.

Since successor and sameness links have been built, along with an identity mapping (*rightmost* $\Rightarrow$ *rightmost*), these nodes are highly active in the Slipnet, and are creating top-down pressure in the form of codelets to search explicitly for other instances of these concepts. For example, an identity mapping between the two leftmost letters is being considered.

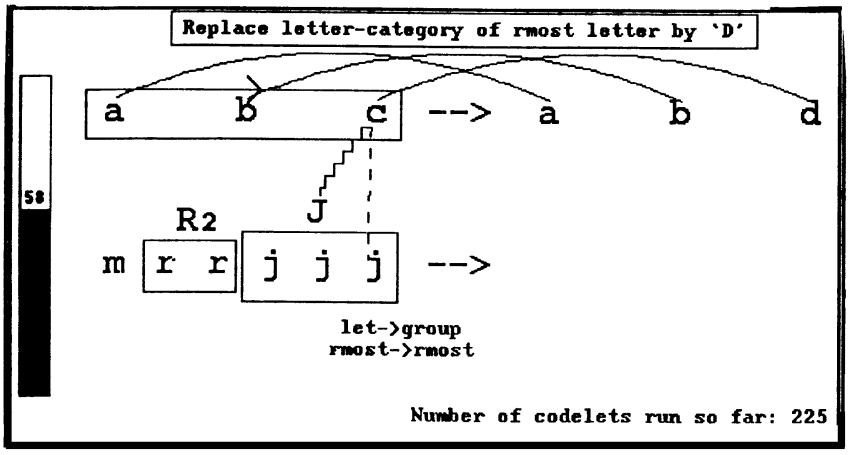
In response to the structures that have been built, the temperature has decreased to 76. The lower the temperature, the less random are the decisions made by codelets, so unlikely structures such as the **bc** group are even more unlikely to be built.



4. The **abc** and **jjj** groups have been built, represented by solid rectangles around the letters. For graphical clarity, the links between letters in a group are not displayed. The existence of these groups creates additional pressure to find new successorship and sameness groups, such as the **rr** sameness group that is being strongly considered. Groups, such as the **jjj** sameness group, become new objects in the string, and can have their own descriptions as well as links and correspondences to other objects. The capital J represents the object consisting of the **jjj** group; the **abc** group likewise is a new object but for clarity a single letter representing it is not displayed. Note that the length of a group is not automatically noticed by the program; it has to be noticed by codelets, just like other attributes of an object. Every time a group node (e.g., *successor group*, *sameness group*) is activated in the Slipnet it spreads some activation to the node *length*. Thus length is now weakly activated and creating codelets to notice lengths, but these codelets are not urgent compared with others and none so far have run and noticed the lengths of groups.

A rule describing the **abc** $\Rightarrow$ **abd** change has been built: "Replace letter-category of rightmost letter by successor". The current version of Copycat assumes that the example change consists of the replacement of exactly one letter, so rule-building codelets fill in the template "Replace _____ by _____", choosing probabilistically from descriptions that the program has attached to the changed letter and its replacement, with a probabilistic bias towards choosing more abstract descriptions (e.g., usually preferring *rightmost letter* to *C*).

The temperature has fallen to 53, resulting from the increasing perceptual organization reflected in the structures that have been built.



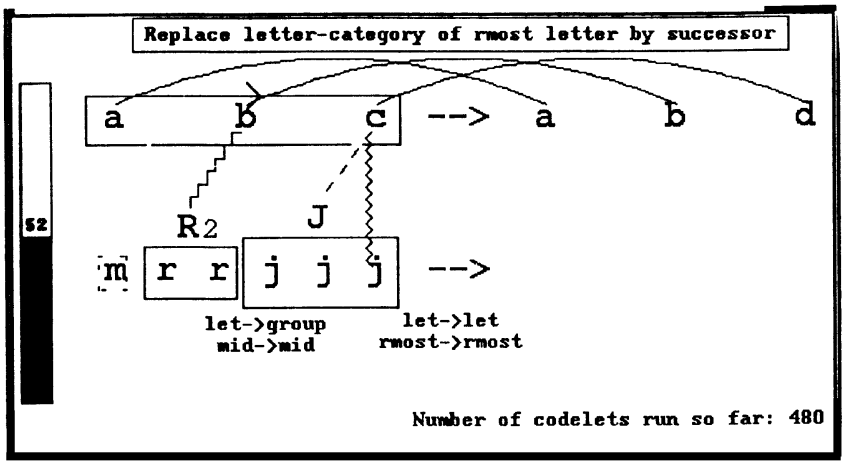
5. Two-hundred and twenty five codelets have run. The letter-to-letter c–j correspondence has been defeated by the letter-to-group c–J correspondence. Reflecting this, the *rightmost* ⇒ *rightmost* mapping has been joined by a *letter* ⇒ *group* mapping underlying the correspondence. The c–J correspondence is stronger than the c–j correspondence because the former covers more objects and because the concept *group* is highly active and thus seen as highly relevant to the problem. However, in spite of its relative weakness, the c–j correspondence is again being considered by a new team of codelets.

Meanwhile, the **rr** group has been built. In addition, its length (represented by the 2 next to the R) has been noticed by a codelet (a probabilistic event). This event activated the node *length*, creating pressure to notice other groups' lengths.

A new rule, "Replace the letter category of the rightmost letter by 'D"', has replaced the old one at the top of the screen. Although this rule is weaker than the previous one, competitions between rival structures (including rules) are decided probabilistically, and this one simply happened to win. However, its weakness has caused the temperature to increase to 58.

If the program were to stop now (which is quite unlikely, since a key factor in the program's probabilistic decision when to stop is the temperature, which is still relatively high), the rule would be adapted for application to the string **mrrjjj** as "Replace the letter category of the rightmost group by 'D"', obeying the slippage *letter* ⇒ *group* spelled out under the c-J correspondence. This yields answer **mrrddd**, an answer that Copycat does indeed produce, though on very rare occasions.

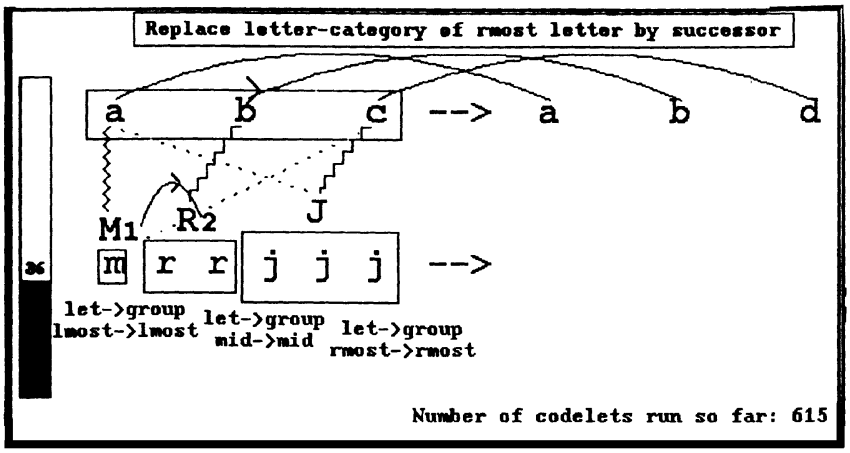
Codelets that attempt to create an answer run frequently throughout the program (their attempts are not displayed here) but are not likely to succeed unless the temperature is low.



6. Four hundred and eighty codelets into the run, the rule "Replace letter-category of rightmost letter by successor" has been restored after it out-competed the previous weaker rule (a probabilistic event). However, the strong c–J correspondence was broken and replaced by its weaker rival, the c–j correspondence (also a probabilistic event). As a consequence, if the program were to stop at this point, its answer would be **mrrjjk**, since the c in **abc** is mapped to a letter, not to a group. Thus the answer-building codelet would ignore the fact that b has been mapped to a group, putting the slippage *letter* $\Rightarrow$ *group* in the workspace. However, the (now) candidate correspondence between the c and the group J is again being strongly considered. It will fight again with the c–j correspondence, but will likely be seen as even stronger than before because of the parallel correspondence between the b and the group R.

In the Slipnet the activation of *length* has decayed since the length description given to the R group hasn't so far been found to be useful (i.e., it hasn't yet been connected up with any other structures). In the Workspace, the diminished salience of the group R's length description "2" is represented by the fact that the "2" is no longer in boldface.

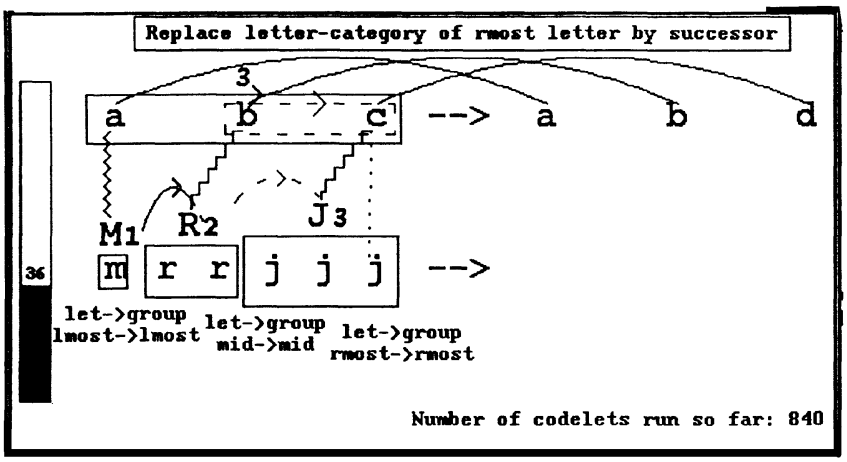
The temperature is still fairly high, since the program is having a hard time making a single, coherent structure out of **mrrjjj**, something that it did easily with **abc**. That continuing difficulty, combined with strong top-down pressure from the two sameness groups that have been built inside **mrrjjj**, caused the system to consider the *a priori* very unlikely idea of making a single-letter sameness group. This is represented by the dashed rectangle around the letter m.



7. As a result of these combined pressures, the M sameness group was built, to parallel the R and J groups in the same string. Its length of 1 has been attached as a description, activating *length*, which makes the program again consider the possibility that group length is relevant for this problem. This activation now more strongly attracts codelets to the objects representing group lengths. Some codelets have already been exploring relations between these objects and, likely due to top-down pressures from **abc** to see successorship relationships, have built a successorship link between the 1 and the 2.

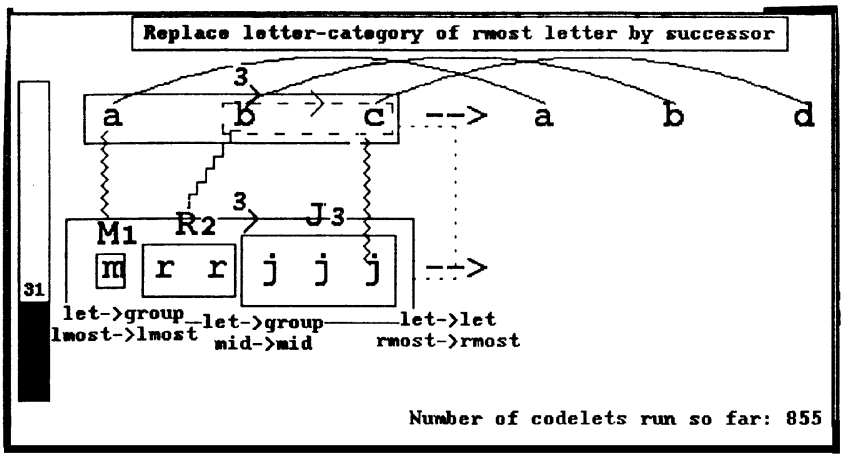
A consistent trio of *letter* $\Rightarrow$ *group* correspondences have been made, and as a result of these promising new structures, the temperature has fallen to the relatively low value of 36, which in turn helps to lock in this emerging view.

If the program were to halt at this point, it would produce the answer **mrrkkk**, which is its most frequent answer (see Figure 3 below).



8. As a result of *length*'s continued activity, length descriptions have been attached to the remaining two groups in the problem, **jjj** and **abc**, and a successorship link between the 2 and the 3 (for which there is much top-down pressure coming from both **abc** and the emerging view of **mrrjjj**) is being considered. Other less likely candidate structures (a

bc group and a c–j correspondence) continue to be considered, though at considerably less urgency than earlier, now that a coherent perception of the problem is emerging and the temperature is relatively low.



9. The link between the 2 and the 3 was built, which, in conjunction with top-down pressure from the **abc** successor group, allowed codelets to propose and build a whole-string group based on successorship links, here between numbers rather than between letters. This group is represented by a large solid rectangle surrounding the three sameness groups. Also, a correspondence (dotted vertical line to the right of the two strings) is being considered between the two whole-string groups **abc** and **mrrjjj**.

Ironically, just as these sophisticated ideas seem to be converging, a small renegade codelet, totally unaware of the global movement, has had some good luck: its bid to knock down the c–J correspondence and replace it with a c–j correspondence was successful. Of course, this is a setback on the global level; while the temperature would have gone down significantly because of the strong **mrrjjj** group that was built, its decrease was offset by the now non-parallel set of correspondences linking together the two strings. If the program were forced to stop at this point, it would answer **mrrjjk**, since at this point, as in pictures 4 and 6, the object that changed, the c, is seen as corresponding to the letter j rather than the group J. However, the two other correspondences will continue to put much pressure on the program (in the form of codelets) to go back to the c–J correspondence.

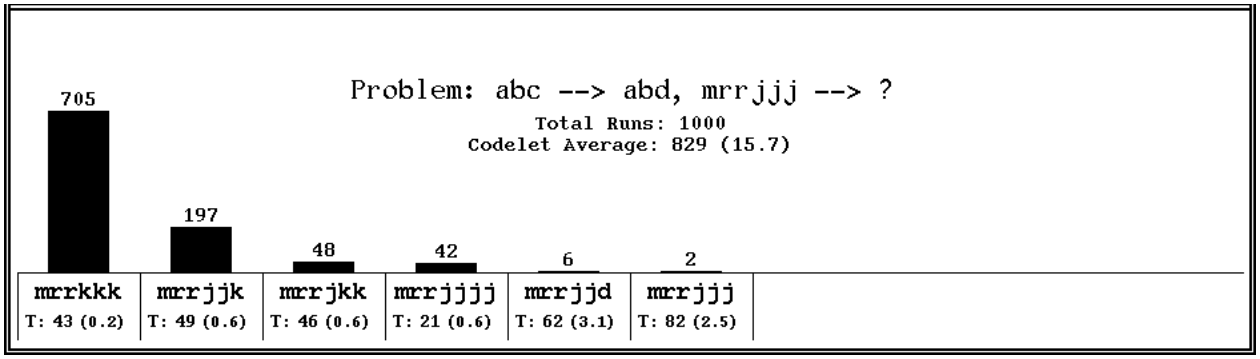
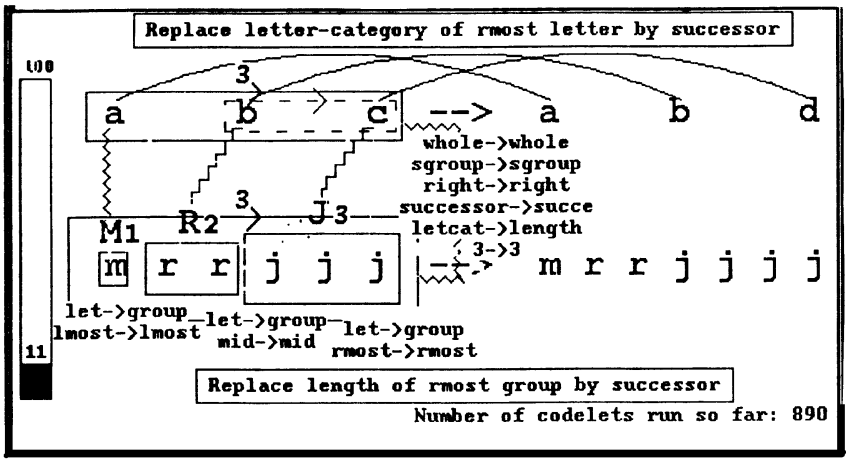


Figure 3: A histogram of the different answers Copycat gave over 1000 runs, each starting from a different random number seed.



10. Indeed, not much later in the run this happens: the c–j correspondence has been broken and the c–J correspondence has been restored. In addition, the proposed whole-string correspondence between **abc** and **mrrjjj** has been built; underlying it are the mappings *whole* ⇒ *whole*, *successor-group* ⇒ *successor-group*, *right* ⇒ *right* (direction of the links underlying both groups), *successor* ⇒ *successor* (type of links underlying both groups), *letter-category* ⇒ *length*, and *3* ⇒ *3* (size of both groups).

The now very coherent set of perceptual structures built by the program resulted in a very low temperature (11), and (probabilistically) due to this low temperature, a codelet has succeeded in translating the rule according to the slippages present in the Workspace: *letter* ⇒ *group* and *letter-category* ⇒ *length* (all other mappings are identity mappings). The translated rule is "Replace the length of the rightmost group by its successor", and the answer is thus **mrrjjjj**.

It should be clear from the description above that because each run of Copycat is permeated with probabilistic decisions, different answers appear on different runs. Figure 3 displays a histogram of the different answers Copycat gave over 1000 runs, each starting from a different random number seed. Each bar's height gives the relative frequency of the

answer it corresponds to, and printed above each bar is the actual number of runs producing that answer. The average final temperature for each answer is also given below each bar's label, with the standard error in parentheses.

The frequency of an answer roughly corresponds to how obvious or immediate it is, given the biases of the program. For example, **mrrkkk**, produced 705 times, is much more immediate to the program than **mrrjjjj**, which was produced only 42 times. However, the average final temperature on runs producing **mrrjjjj** is much lower than on runs producing **mrrkkk** (21 versus 43), indicating that even though the latter is a more immediate answer, the program judges the former to be a better answer, in terms of the strength and coherence of the structures it built to produce each answer.

7 Summary

To summarize, Copycat makes sense of and perceives analogies between situations in a fluid and cognitively plausible way via interaction among three main mechanisms:

- Codelets continually investigating possible structurings in the Workspace and making probabilistic decisions concerning
  - what to look at next;
  - whether to build a structure there (possibly destroying an existing structure);
  - how fast to build it.

  Probabilities are used to insure that no possibilities are ruled out in principle, but that not all possibilities have to be considered.
- The Slipnet, in which concepts
  - become active when instances of them are noticed in the Workspace;
  - feed back to the Workspace by creating top-down pressure, via codelets, to look for further instances of themselves; and
  - spread activation to their neighbors.

  Objects in the Workspace can be mapped onto one another, often requiring slippages between their associated concepts. The Slipnet defines intrinsic resistance to such slippages, but slippages become easier when the concept defining the slippage (e.g., *opposite* for the slippage *successor* $\Rightarrow$ *predecessor*) becomes active.
- Temperature, which starts off high and drops as perceptual structures are built (and rises when they are destroyed). Temperature in turn feeds back to codelets by making their decisions more random when temperature is high and more deterministic when temperature is low.

Via these mechanisms, Copycat avoids the Catch-22 of perception: you can't explore everything, but you don't know which possibilities are worth exploring without first exploring them. You have to be open-minded, but the territory is too vast to explore everything; you need to use probabilities in order that exploration be fair. In Copycat's strategy, early on there is little information, resulting in high temperature and high degree of randomness, with lots of parallel explorations. As more and more information is obtained and fitting concepts are found, the temperature falls, and exploration becomes more deterministic and more serial as certain concepts come to dominate. The overall result is that the system gradually changes from a mostly random, parallel, bottom-up mode of processing to a deterministic, serial, top-down mode in which a coherent perception of the situation at hand gradually discovered and gradually "frozen in". Our claim is that this gradual transition between different modes of processing is a feature common to cognitive systems and to adaptive systems in general.

While Copycat's mechanisms have been shown to be successful to a high degree in its letter-string microworld [16], it remains to be demonstrated that such a system will work well on more realistic situations requiring a much larger repertoire of concepts (e.g., visual images). We believe that it will, and this belief is supported by the success of two projects using architectures similar to Copycat's: Tabletop, which makes analogies between objects and relationships on an idealized cafe table [2], and Letter Spirit, which recognizes and creates letters in different typeface styles on an idealized grid [13]. Copycat has recently been extended to incorporate "self-watching", in which the program monitors at a high level its own actions [14], an essential component for general high-level perception that was missing in Copycat. Current work in Hofstadter's group includes extending these ideas to the task of solving Bongard problems, a beautiful and open-ended class of visual analogy problems [1, 7]. If successful, this project will go a long way towards the development of a general cognitive architecture for high-level perception and analogy-making.

8 Epilogue: Copycat and the Immune System

Copycat is one of the "other distributed autonomous systems" referred to in the title of this book. Copycat was not designed with the immune system in mind; ant colonies and cell metabolism (which I didn't discuss here) were the direct biological inspirations. However there are some interesting parallels between Copycat and the immune system that are worth pointing out. At the most general level, both systems produce global behavior that emerges from the interactions among many simple components with local interactions. At a more specific level, both systems are faced with complex recognition problems that require exploration of many possibilities. They both rely on search controlled by both bottom-up and top-down forces. Like Copycat, the immune system must have the potential to deal with any possibility (i.e., pathogen); it cannot *a priori* exclude possibilities ahead of time. But the number of possible pathogens is huge and cannot be prepared for ahead of time. Like Copycat, the immune system uses randomness to avoid this Catch-22. Random combinations are used to construct B-cell receptors from gene libraries, and random mutations allow the system to find increasingly better matches to antigens during affinity maturation. Also like Copycat, the immune system's search is controlled by both bottom-up and top-down

aspects. One example of bottom-up search is the continual patrol of B-cells with different receptors, collectively prepared to approximately match any antigen. Top-down search consists of focused B-cells, which when activated by a match, create similar B-cells to zero in on the particular antigen that has been detected. As in all adaptive systems, maintaining the right balance between these two search modes is essential.

More detailed analogies between Copycat and the immune system are possible. For example, a Slipnet node's activation could be said to correspond to a concentration level of particular B-cells or cytokines: both reflect a particular possibility that the system has deemed worth exploring. Codelets that explore structures might correspond to lymphocytes of various kinds that patrol the body seeking signs of pathogen invasion and other kinds of disequilibrium. To quote from Hofmeyer's chapter in this volume: "we can abstractly view lymphocytes as mobile, independent *detectors*. There are trillions of these lymphocytes, forming a system of distributed detection, where there is no centralized control, and little, if any, hierarchical control. Detection and elimination of pathogens is a consequence of trillions of cells—detectors—interacting through simple, localized rules." This sounds very much like the way codelets work. Going further, high temperature in Copycat might correspond to fever in the body—the former a signal to Copycat that it must explore more broadly and intensely; the latter a signal to the body that it must increase the intensity of the immune response. (Several of these correspondences were proposed by L. Segel, personal communication.)

Analogies such as these force us to think more broadly about the systems one is building or trying to understand. If one notices, say, that the role of cytokines in immune signalling is similar to that of codelets that call attention to particular sites in an analogy problem, one is thinking at a general *information-processing* level about the function of a biological entity. Or if one sees that temperature-like phenomena in the immune system—fever, inflammation—emerge from the joint actions of many agents, one might get some ideas on how to better model temperature in a system like Copycat. The main purposes of this chapter have been to draw the reader's attention to mechanisms of recognition and search that we have proposed as general properties of complex systems, to provoke some thought on how these properties might be implemented in specific complex systems in nature, and to open discussion on how artificial intelligence and cognitive science might additionally benefit from what is being learned about such natural systems.

Acknowledgments

Many thanks to Lee Segel for discussions at the Santa Fe Institute on links between distributed artificial intelligence and the immune system, for inviting me to a very stimulating workshop, and for helpful comments on an earlier version of this manuscript.

References

[1] M. Bongard. *Pattern Recognition.* Hayden Book Co., Spartan Books, Rochell Park, NJ, 1970.

[2] R. M. French. *The Subtlety of Sameness: A Theory and Computer Model of Analogy-Making.* MIT Press, Cambridge, MA, 1995.

[3] D. R. Hofstadter. Analogies and roles in human and machine thinking. Chapter 24 in [8].

[4] D. R. Hofstadter. The architecture of Jumbo. Chapter 2 in [9].

[5] D. R. Hofstadter. To seek whence cometh a sequence. Chapter 1 in [9].

[6] D. R. Hofstadter. Variations on a theme as the crux of creativity. Chapter 12 in [8].

[7] D. R. Hofstadter. *Gödel, Escher, Bach: an Eternal Golden Braid.* Basic Books, New York, 1979.

[8] D. R. Hofstadter. *Metamagical Themas.* Basic Books, New York, 1985.

[9] D. R. Hofstadter. *Fluid Concepts and Creative Analogies.* Basic Books, New York, 1995.

[10] D. R. Hofstadter and M. Mitchell. The Copycat project a model of mental fluidity and analogy-making. Chapter 5 in [9].

[11] D. R. Hofstadter and M. Mitchell. The Copycat project: A model of mental fluidity and analogy-making. In K. Holyoak and J. Barnden, editors, *Advances in Connectionist and Neural Computation Theory, Volume 2: Analogical Connections*, Norwood, NJ, 1984. Ablex.

[12] J. H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, MA, 1992. Second edition (First edition, 1975).

[13] G. E. McGraw Jr. *Letter Spirit: Emergent High-Level Peception of Letters Using Fluid Concepts.* PhD thesis, Indiana Unversity, Bloomington, IN, 1995.

[14] J. B. Marshall. *Metacat: A Self-Watching Cognitive Architecture for Analogy-Making and High-Level Perception.* PhD thesis, Indiana Unversity, Bloomington, IN, 1999.

[15] M. J. Meredith. *Seek-Whence: A Model of Pattern Perception.* PhD thesis, Indiana Unversity, Bloomington, IN, 1986.

[16] M. Mitchell. *Analogy-Making as Perception: A Computer Model.* MIT Press, Cambridge, MA, 1993.

[17] M. Mitchell and D. R. Hofstadter. Perspectives on Copycat: Comparisons with recent work. Chapter 6 in [9].