PROPERTIES OF THE BUCKET BRIGADE ALGORITHM

John H. Holland

The University of Michigan

The bucket brigade algorithm is designed to solve the apportionment of credit problem for massively parallel, message-passing, rule-based systems. The apportionment of credit problem was recognized and explored in one of the earliest significant works in machine learning (Samuel [1959]). In the context of rule-based systems it is the problem of deciding which of a set of early acting rules should receive credit for "setting the stage" for later, overtly successful actions. In the systems of interest here, in which rules conform to the standard condition/action paradigm, a rule's overall usefulness to the system is indicated by a parameter called its *strength*. Each time a rule is active, the bucket brigade algorithm modifies the strength so that it provides a better estimate of the rule's usefulness in the contexts in which it is activated.

The bucket brigade algorithm functions by introducing an element of competition into the process of deciding which rules are activated. Normally, for a parallel message-passing system, all rules having condition parts satisfied by some of the messages posted at a given time are automatically activated at that time. However, under the bucket brigade algorithm only some of the satisfied rules are activated. Each satisfied rule makes a *bid*, based in part on its strength, and only the highest bidders become active (thereby posting the messages specified by their action parts). The size of the bid depends upon both the rule's strength *and* the specificity of the rule's conditions. (The rule's specificity is used on the broad assumption that, other things being equal, the more information required by a rule's conditions, the more likely it is to be "relevant" to the particular situation confronting it). In a specific version of the algorithm used for classifier systems, the bid of classifier C at time t is given by

$$b(C,t) = cr(C)s(C,t),$$

where r(C) is the specificity of rule C (equal, for classifier systems, to the difference between the total number of defining positions in the condition and the number of "don't cares" in the condition), s(C,t) is the strength of the rule at time t, and c is a constant considerably less than 1

(e.g., 1/4 or 1/8).

The essence of the bucket brigade algorithm is its treatment of each rule as a kind of mid-level entrepreneur (a "middleman") in a complex enconomy. When a rule C wins the competition at time t, it must decrease its strength by the amount of the bid. Thus its strength on time-step t+1, after winning the competition, is given by

$$s(C, t+1) = S(C, t) - b(C, t) = (1 - cr(C))S(C, t).$$

In effect C has paid for the privilege of posting its message. Moreover this amount is actually paid to the classifers that sent messages satisfying C's conditions -- in the simplest formulation the bid is split equally amongst them. These message senders are C's *suppliers*, and each receives its share of the payment from the *consumer* C. Thus, if C_1 has posted a message that satisfies one of C's conditions, C_1 has its strength increased so that

$$s(C_1, t+1) = S(C_1, t) + b(C, t)/n(C, t) = (1 - cr(C)/n(C,t))S(C,t),$$

where n(C, t) is the number of classifiers sending messages that satisfy C at time t.

In terms of the economic metaphor, the suppliers $\{C_1\}$ are paid for setting up a situation usable by consumer C. C, on the next time step, changes from consumer to supplier because it has posted its message. If other classifiers then bid because they are satisfied by C's message, and if they win the bidding competition, then C in turn will receive some fraction of those bids. C's survival in the system depends upon its turning a profit as an intermediary in these local transactions. In other words, when C is activated, the bid it pays to its suppliers must be less (or, at least, no more) than the average of the sum of the payments it receives from its consumers.

It is important that this process involves no complicated "bookkeeping" or memory over long sequences of action. When activated, C simply pays out its bid on one time-step, and is immediately paid by its consumers (if any) on the next time-step. The only variation on this transaction occurs on time-steps when there is payoff from the environment. Then, all classifiers active on that time-step receive equal fractions of the payoff in addition to any payments from classifiers active on the next time-step. In effect, the environment is the system's ultimate consumer. From a global point of view, a given classifier C is likely to be

profitable only if its usual consumers are profitable. The profitability of any chain of consumers thus depends upon their relevance to the ultimate consumer. Stated more directly, the profitability of a classifier depends upon its being coupled into sequences leading to payoff.

As a way of illustrating the bucket brigade algorithm, consider a set of 2-condition classifiers where, for each classifier, condition 1 attends to messages from the environment and condition 2 attends to messages from other classifiers in the set. As above, let a given classifier C have a bid fraction b(C) and strength s(C,t) at time t. Note that condition 1 of C defines an equivalence class E in the environment consisting of those environmental states producing messages satisfying the condition.

Consider now the special case where the activation of C produces a response r that transforms states in E to states in another equivalence class E' having an (expected) payoff u. Under the bucket brigade algorithm, when C wins the competition under these circumstances its strength will change from s(C,t) to

$$s(C,t+1) = s(C,t) - b(C)s(C,t) + u + (\text{any bids C receives from classifiers active on the next time-step}).$$

Assuming the strength of C is small enough that its bid b(C)s(C,t) is considerably less than u, the usual case for a new rule or for a rule that has only been activated a few times, the effect of the payoff is a considerable strengthening of rule C.

This strengthening of C has two effects. First, C becomes more likely to win future competitions when its conditions are satisfied. Second, rules that send messages satisfying one (or more) of C's conditions will receive higher bids under the bucket brigade, because $b(C)s(C,t+1) > b(C)s(C,t)$.

Both of these effects strongly influence the development of the system. The increased strength of C means that response r will be made more often to states in E when C competes with other classifiers that produce different responses. If states in E' are the only payoff states accessible from E, and r is the only response that will produce the required transformation from states in E to states in E', then the higher probability of a win for C translates into a higher payoff rate to the classifier system.

Of equal importance, C's higher bids mean that rules sending messages satisfying C's second condition will be additionally strengthened because of C's higher bids. Consider, for example, a classifier C_0 that transforms environmental states in some class E_0 to states in class E by evoking response r_0. That is, C_0 acts upon a causal relation in the environment to "set the stage" for C. If C_0 also sends a message that satisfies C's second condition, then C_0 will benefit from the "stage setting" because C's higher bid is passed to it via the bucket brigade.

It is instructive to contrast the "stage setting" case with the case where some classifier, say C_1, sends a message that satisfies C but *does not* transform states in E_1 (the environmental equivalence class defined by its first condition) to states in E. That is, C_1 attempts to "parasitize" C, extracting bids from C via the bucket brigade without modifying the environment in ways suitable for C's action. Because C_1 is not instrumental in transforming states in E_1 to states in E, it will often happen that activation of C_1 is not followed by activation of C on the subsequent time-step because C's first (environmental) condition is not satisfied. Every time C_1 is activated without a subsequent activation of C it suffers a loss because it has paid out its bid $b(C_1)s(C_1,t)$, without receiving any income from C. Eventually C_1's strength will decrease to the point that it is no longer a competitor. (There is a more interesting case where C_0 and C_1 manage to become active simultaneously, but that goes beyond the confines of the present illustration).

One of the most important consequences of the bidding process is the automatic emergence of default hierarchies in response to complex environments. For rule-based systems a "default" rule has two basic properties:

1) It is a general rule with relatively few specified properties and many "don't cares" in its condition part, and
2) when it wins a competition it is often in error, but it still manages to profit often enough to survive.

It is clear that a default rule is preferable to no rule at all, but, because it is often in error, it can be improved. One of the simplest improvements is the addition of an "exception" rule that responds to situations that cause

the default rule to be in error. Note that, in attempting to identify the error-causing situations, the condition of the exception rule specifies a *subset* of the set of messages that satisfy the default rule. That is, the condition part of the exception rule *refines* the condition part of the default rule by using *additional* identifying bits (properties). Because rule discovery algorithms readily generate and test refinements of existing strong rules, useful exception rules are soon added to the system.

As a direct result of the bidding competition, an exception rule, once in place, actually aids the survival of its parent default rule. Consider the case where the default rule and the exception rule attempt to set a given effector to a different values. In the typical classifier system this conflict is resolved by letting the highest bidding rule set the effector. Because the exception rule is more specific than the default rule, and hence makes a higher bid, it usually wins this competition. In winning, the exception rule actually prevents the default rule from paying its bid. This outcome saves the the default rule from a loss, because the usual effect of an error, under the bucket brigade, is activation of consumers that do not bid enough to return a profit to the default rule. In effect the exception protects the default from some errors. Similar arguments apply, under the bucket brigade algorithm, when the default and the exception only influence the setting of effectors indirectly through intervening, coupled classifiers.

Of course the exception rules may be imperfect themselves, selecting some error-causing cases, but making errors in other cases. Under such circumstances, the exception rules become default rules relative to more detailed exceptions. Iteration of the above process yields an ever more refined, and efficient, default hierarchy. The process improves both overall performance and the profitability of each of the rules in the hierarchy. It also uses fewer rules than would be required if all the rules were developed at the most detailed level of the hierarchy (see Holland, Holyoak, Nisbett, and Thagard [1986]). The bucket brigade algorithm strongly encourages the top-down discovery and development of such hierarchies (cf. Goldberg [1983] for a concrete example).

At first sight, consideration of long sequences of coupled rules would seem to uncover an important limitation of the bucket brigade algorithm. Because of its local nature, the bucket brigade algorithm can only propagate strength back along a chain of suppliers through repeated activations of the whole sequence. That is, on the first repetition of a

sequence leading to payoff, the increment in strength is propagated to the immediate precursors of the payoff rule(s). On the second repetition it is propagated to the precursors of the precursors, etc. Accordingly, it takes on the order of n repetitions of the sequence to propagate the increments back to rules that "set the stage" n steps before the final payoff. However, this observation is misleading because certain kinds of rule can serve to "bridge" long sequences.

The simplest "bridging action" occurs when a given rule remains active over, say, T successive time-steps. Such a rule passes increments back over an interval of T time-steps on the *next* repetition of the sequence. This qualification takes on importance when we think of a rule that shows persistent activity over an *epoch* -- an interval of time characterized by a broad plan or activity that the system is attempting to execute. For the activity to be persistent, the condition of the epoch-marking rule must be general enough to be satisfied by just those properties or cues that characterize the epoch. Such a rule, if strong, marks the epoch by remaining active for its duration.

To extract the consequences of this persistent activation, consider a concrete plan involving a sequence of activities, such as a "going home" plan. The sequence of coupled rules used to execute this plan on a given day will depend upon variable requirements such as "where the car is parked", "what errands have to be run", etc. These detailed variations will call upon various combinations of rules in the system's repertoire, but the epoch-marking "going home" rule D will be active throughout the execution of each variant. In particular, it will be active both at the beginning of the epoch and at the time of payoff at the end of the plan ("arrival home"). As such it "bridges" the whole epoch.

Consider now a rule I that initiates the plan and is coupled to (sends a message satisfying) the general epoch-marking rule D. The *first repetition* of the sequence initiated by I will result in the strength of I being incremented. This comes about because D is strengthened by being active at the time of payoff and, because it is a consumer of I's message, it passes this increment on to I the very next time I is activated. D "supports" I as an element of the "going home" plan. The result is a kind of one-shot learning in which the earliest elements in a plan are rewarded on the very next use. This occurs despite the local nature of the bucket brigade algorithm. It requires only the presence of a general rule -- a kind of default -- that is activated when some general kind of activity or goal

is to be attained. An appropriate rule discovery algorithm, such as a genetic algorithm, will soon couple more detailed rules to the epoch-marking rule. And, much as in the generation of a default hierarchy, these detailed rules can give rise to further refined offspring. The result is an emergent plan hierarchy going from a high-level sketch through progressive refinements yielding ways of combining progressively more detailed components (rule clusters) to meet the particular constraints posed by the current state of the environment. In this way a limited repertoire of rules can be combined in a variety of ways, and in parallel, to meet the perpetual novelty of the environment.

References.

Goldberg, D. E. *Computer-aided Gas Pipeline Operation Using Genetic Algorithms and Machine Learning*. Ph. D. Dissertation (Civil Engineering). The University of Michigan. 1983.

Holland, J. H., Holyoak, K. J,, Nisbett, R. E., and Thagard, P. R. *Induction: Learning, Discovery, and the Growth of Knowledge*. [forthcoming, MIT Press].

Samuel, A. L. "Some studies in machine learning using the game of checkers." *IBM Journal of Research and Development, 3*. 211-232, 1959.