

How to Calm Hyperactive Agents

H. Van Dyke Parunak

Sven A. Brueckner

Robert Matthews

John Sauter

Altarum

3520 Green Ct. Suite 300
Ann Arbor, MI 48105 USA

+1 734 302 4684

+1 734 302 4683

+1 734 302 4657

+1 734 302 4682

{van.parunak, sven.brueckner, robert.matthews, john.sauter}@altarum.org

ABSTRACT

System performance in multi-agent resource allocation systems can often *improve* if individual agents *reduce* their activity. Agents in such systems need a way to modulate their individual behavior in the light of the system's state, preferably in a way that does not require centralized control. We illustrate the problem of hyperactive agents in two domains related to resource allocation. We describe a simple, decentralized scheme, inspired by insect pheromones, that enables individual agents to adjust their level of activity as the system operates, and discuss a general approach to dealing with approaching deadlines. Then we demonstrate the effectiveness of these mechanisms in the two example domains.

Categories and Subject Descriptors

I.2.11 [Distributed Artificial Intelligence]: Coherence and coordination; Multiagent systems.

General Terms

Algorithms, Performance, Experimentation.

Keywords

Digital pheromones, machine learning, adaptation, resource allocation, complexity, dynamics

1. INTRODUCTION

Recent research into the dynamics of multi-agent systems shows that system performance can often *improve* if individual agents *reduce* their activity. This phenomenon is particularly evident in multi-agent resource allocation systems, where the activity in question is competing for access to shared resources. When contention for a resource is high, the average level of supply experienced by each of the competing agents may be so low that none of the agents can achieve a high level of performance, and overall system output suffers.

Agents in such systems need a way to modulate their individual behavior in the light of the system's state. Centralized mechanisms such as markets are one approach. We describe more

decentralized techniques, which rely only on knowledge locally available in a single agent. Our basic technique is inspired by insect pheromone mechanisms and can be viewed as a specialized form of temporal difference learning. The key concept is that an agent's decision to compete for resources is stochastic, and the probability that the agent will compete on a given turn depends on the success it has enjoyed on recent attempts. We extend this technique with methods that deal with approaching deadlines by shifting the focus of the agent population from exploratory to exploitative behavior.

Section 2 illustrates the problem of hyperactive agents in two examples of resource allocation. Section 3 describes our mechanisms. Sections 4 and 5 report on results of the mechanisms in the two application domains outlined in Section 2. Section 6 concludes.

2. HYPERACTIVE AGENTS

Two application domains illustrate the problem of hyperactive agents. Dynamic graph coloring is an abstract model of resource allocation, while mission scheduling is a concrete application in this domain.

2.1 Dynamic Graph Coloring

Graph coloring is a useful abstraction of many resource allocation problems. Each node in the graph represents a task, each color represents a resource, and an edge between two nodes indicates that a single resource cannot service them simultaneously. Under this model, resource allocation becomes a matter of coloring the graph to minimize the proportion of adjacent nodes that have the same color.

Consider a random graph of N nodes, each connected to K neighbors via undirected edges. To construct such a graph, we distribute the nodes randomly on a $2D$ 1 by 1 square. Then, cycling through the nodes in a fixed order, we connect each node to its nearest neighbor on the square until it has K neighbors. Once constructed, this structure is static. The resulting graph may not necessarily be planar, but violations will be local. These graphs are characterized by a short characteristic path length (approx. 1.3) and medium clustering coefficient (approx. 0.68).

We have been studying the dynamics of a soft real-time graph coloring algorithm developed by Kestrel Institute in addressing sensor allocation problems [5, 10]. The graph's dynamics are based on synchronous execution in discrete time steps. At any point in time, each node is assigned one of G colors. The assignment of colors may change over time. A node can perceive the current color of its neighbors. A change in a neighbor's color is perceived after a delay of CL ("communication latency") time units. All nodes share a global probability value AL ("activation level"), which determines the individual node's probability to activate its local reasoning mechanism at a given time step. If acti-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS'03, July 14-18, Melbourne, Australia.

Copyright 2003 ACM 1-58113-000-0/00/0000...\$5.00.

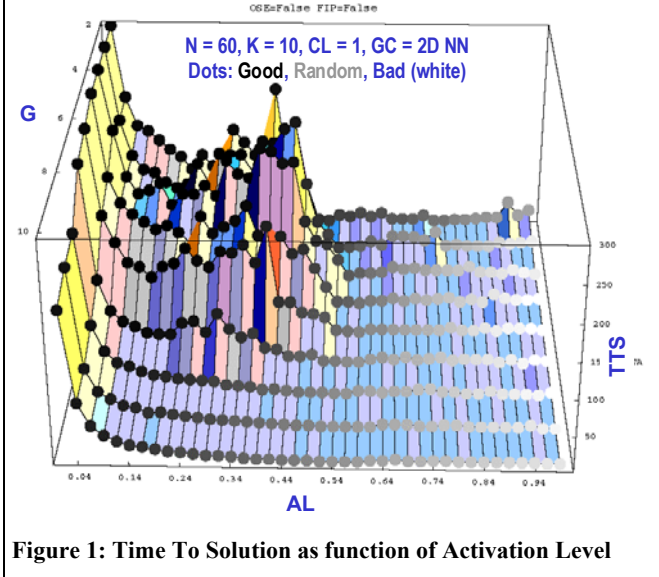


Figure 1: Time To Solution as function of Activation Level

ated, a node re-evaluates its color assignment based on the local metric Degree of Conflict (DoC), computed by dividing the number of neighbors that share the node’s color by the overall number of neighbors (K). The node calculates the DoC for each of the G possible colors, using the perceived color of its neighbors. The node compares the resulting DoC values with the DoC of its current color. Any color whose DoC fulfills a *movement direction* constraint MD is placed into a set of available colors. Two options for MD are “any” (any color is eligible) and “only up” (only colors that would improve the DoC are eligible). The new color of the node is selected from this set of available colors based on a *color selection* rule CS (in this case, pick the color giving the “best,” that is, the lowest, DoC).

Two metrics over this system are useful to us. Global degree of conflict (GDOC) is the number of edges in the graph that connect two nodes with the same color, times G , divided by the overall number of edges. GDOC is 0 for a perfectly colored graph, 1 if all agents make random choices, and greater than 1 if conflicts are worse than under random choice. Time to Solution (TTS) estimates how long it takes the system to asymptote to a solution. We execute multiple runs at each configuration, and run each long enough to asymptote safely, capturing the time series of GDOC for each run. Then we construct the Mean Time Series (MTS) over the runs for each configuration, compute the Delta between the first point in each MTS and the Stable Point (the mean over the last 50 points), and define the Threshold as 2% of the maximum Delta over all configurations. In the MTS for each configuration, we measure the distance of each point from the Stable Point, and declare that the solution has been reached if this distance is less than the Threshold. Both GDOC and the convergence used to estimate TTS are global metrics, not locally available to individual agents.

Consider first the dependency of TTS on AL. Figure 1 plots TTS as a function of AL and G . Dots superimposed on the TTS surface show GDOC, ranging from black (GDOC =

Table 1: Basic Structure of a CAMERA Problem (from [4])

Missions [value]	Resources								
	A	B	C	D	E	F	G	H	I
Mission 1 [300]	R1	X	X						
	R2				X		X		
	R3							X	X
Mission 2 [600]	R1	X		X					
	R2					X	X		
	R3							X	X
Mission 3 [200]	R1		X	X					
	R2						X		
	R3								X
Mission 4 [400]	R1	X	X	X					
	R2					X	X		
	R3							X	X

0) to white (GDOC = 1.8) Gray dots indicate effectively random choice (GDOC = 1).

For very low AL, TTS is very high, since the system can take only very small steps toward solution. However, eventually the system reaches a good solution. Up to a point, TTS decreases with increasing AL, as the system takes larger steps toward the solution. Then the system begins to thrash, leading to a peak in TTS, which then drops to low time to solution with poor results. (Although TTS is low, an individual agents cannot perceive that the system has converged, and so continue executing in this region, wasting resources on a futile search for improvement.)

If AL is too slow, nodes cannot keep up with dynamic changes. But if AL is too fast with respect to CL, nodes make decisions with obsolete information and thrash. Very high AL produces essentially random behavior. This observation is critical in the light of contemporary tendencies to reducing the cycle time in information systems (essentially, increasing AL). Such increases are useful only up to a point, beyond which they introduce instability. Our result here is comparable to the behavior observed in [7] in a related domain.

Operationally, we want individual agents to reduce their effective AL to stay in high-performing region of Figure 1, but without requiring centralized computation and global distribution of metrics such as GDOC and system convergence.

2.2 Mission Scheduling

The CAMERA system at ISI [11] allocates resources to aviator training missions. Each mission has a number of requirements, each of which can be satisfied by a number of resources, includ-

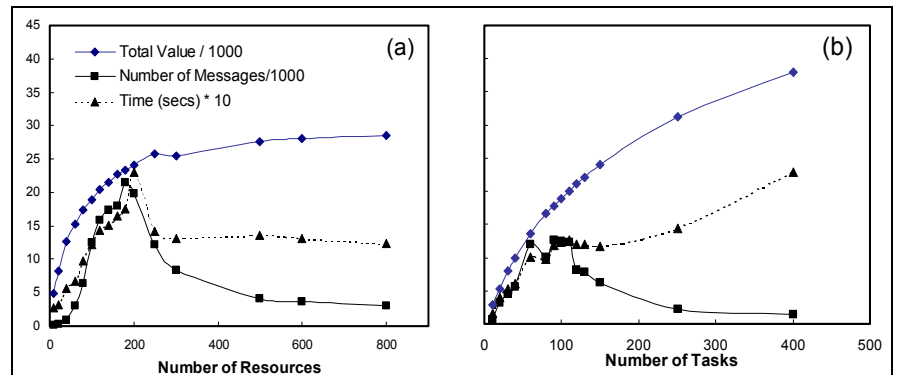


Figure 2: Execution profiles for Marbles. (a) 100 tasks, (b) 100 resources (from [6])

ing bombing ranges, pilots, instructors, and aircraft.

Table 1 shows the basic structure of a CAMERA problem. A set of resources (R) is available to satisfy a set of requirements (Q), but can actually be assigned only to one. Each requirement is part of one (and only one) task from a set of tasks (T). There also exists a set of eligibility markings (L) (the X's in Table 1) that denote if a given resource is eligible to satisfy a specific requirement. We call a cell with an X, an "engagement."

Every task is assigned a positive value that is "recovered" only if every requirement of the task is assigned a resource. In other words, if a task has all its requirements satisfied, then the total recovered value of the overall problem solution will include the full value of the task, otherwise 0 value is included from the task.

We focus here on a family of bidding mechanisms known as Marbles [6]. These schemes are a subset of single-resource auctions, with two interesting characteristics. First, they resemble Edgeworth barter rather than Walrasian auctions [1], in that resources move among consumers as the protocol executes, rather than waiting for all bidders to submit bids and then clearing once for all. Second, when an individual task senses that it will be unable to get the resources it needs, it stops trying, thereby dynamically reducing the constraints on the system. This behavior already exemplifies the value of damping "hyperactive agents" that is the theme of this paper. Figure 2 shows how execution time and message traffic increase until tasks start dropping out.

2.3 Characterizing Hyperactivity

On the basis of these examples, we can identify two main drivers of agent hyperactivity.

Hyperactivity may reflect an agent's specialization. In CAMERA, a task is hyperactive if its requirements are inconsistent with the resources available. In this case, any negotiation load it places on the system is excessive, because there is no chance that it will succeed in obtaining its resources and delivering its value. This case is an example of an agent's functional specialization being inappropriate for the state of the problem. In graph coloring, different regions of the graph may experience different rates of change. For instance, the nodes may represent surveillance regions in a sensor network, and only those nodes in the vicinity of an intruder need to update frequently. A node is hyperactive if its activation level is higher than the rate of change of the environment in its immediate vicinity. In cases like this, the agent's geographical specialization is inappropriate.

Hyperactivity may also reflect the system's *stability*, which can change the relative priority of exploring vs. exploiting the environment [8]. When first started, or when the environment changes, individual agents do not know whether their specializations are appropriate or not. All agents need to execute in order to *explore* the problem state and determine whether their specializations are useful. If the system stabilizes, preference should be given to those agents whose specializations *exploit* the state of the system.

A common underlying feature of these drivers is the tension between global system requirements and the local nature of the information available to each agent. Whether an agent's specialization is suited to the overall requirements of the system, or whether the system is stable enough to favor exploitation over exploration, are global characteristics to which an individual agent generally does not have access. Our mechanisms must en-

able individual agents to estimate its hyperactivity on the basis of locally available information.

3. CALMING MECHANISMS

We calm with hyperactive agents with two mechanisms. The first, inspired by insect pheromones, supports the ongoing operation of a system (a "going concern," related to maintenance goals [12]). The second adjusts agent activity in the face of approaching deadlines (related to achievement goals).

3.1 Pheromone Damping for Ongoing Operation

In ongoing operations, hyperactive agents can congest the system and reduce performance below what might otherwise be achieved. An appropriate damping mechanism should have several characteristics. It should enable individual agents to detect when their activity is inappropriately high and slow down, but it should also allow agents to reenter the system in the event that the environment changes and their specialization becomes needed. Because agents must estimate the appropriateness of their action on the basis of local information, the decision mechanism should integrate multiple observations, rather than acting precipitously.

One model of learning that exhibits these features of fusing information from multiple observations and purging obsolete information is the pheromone mechanism used by insects to guide their collective decision processes. This mechanism has four components:

Aggregation.—An entity (insect or simulation agent) marks an event by adding to an existing base of pheromone. This component effectively fuses multiple observations into a single variable.

Evaporation.—Over time, pheromones gradually fade (unless new deposits reinforce them). This component is a novel mechanism for truth maintenance. Instead of remembering everything that it has learned unless there is reason to forget it (the traditional AI approach, entangled in NP-completeness), an ant colony immediately begins to forget everything it learns as soon as it learns it, unless it is reinforced.

Propagation.—Spatially, pheromones disperse usually with the maximum concentration of a deposit remaining at the original point of deposit.

Sensing.—Other insects/agents make decisions or take actions based upon the pheromone levels they sense in their environment.

To avoid hyperactivity in the steady state, we assign each agent a No-Action Pheromone (NAP) variable for each action that we wish to modulate.

- NAP decays exponentially over time (multiplying by a constant $E < 1$ at each time step)
- The agent makes a deposit into its NAP when its individual experience suggests it may be hyperactive with respect to the action in question. This deposit rule Dep is application-dependent, but must be based on locally-available information.
- Each time the agent is eligible to take action, it generates a random number in $[0,1]$ and compares it with the ratio of action's NAP to a threshold θ . If the random number exceeds this ratio, the agent acts, otherwise it does not. This rule selectively discourages agents that have not been successful in recent trials from imposing on the system at the current cycle. The consequences of NAP reaching the threshold depend on the applica-

tion. In some cases, it is useful to use such an overflow as a signal to retire the agent permanently.

The pheromone dynamics impart two important characteristics to this mechanism.

1. Because NAP aggregates over individual deposits, a decision to abstain can be the result of repeated experience rather than a one-time disappointment. This averaging behavior is important because the deposit is based on locally available information, which may only approximately estimate the global situation. (Thus the deposit rate and threshold should reflect the designer’s confidence that local metrics track global state.)
2. Because NAP evaporates, an unsuccessful agent can reenter the activity later, when circumstances may have changed so that its approach is again worth trying. (Thus the evaporation rate E should reflect expectations of the rate of change of the problem.)

The combination of an exponential decay with a episodic deposit is formally reminiscent of temporal difference learning with eligibility traces $TD(\lambda)$ [14]. In both cases, an overall result (total pheromone strength in our case; estimated return in $TD(\lambda)$) is computed as a discounted sum of inputs over time.

The major difference between the two is the condition and size of the deposit event. In $TD(\lambda)$, a uniform deposit occurs each time the learning agent visits a state, independent of the return it receives in that state. In our system, the amount of the deposit depends on the outcome of the associated action. $TD(\lambda)$ distributes responsibility for an erroneous prediction of the return to all recently visited states, weighting their responsibility only by how long ago they were visited. Pheromone damping assigns that responsibility only to actions that actually experience a prediction error.

A potential extension of our method offers another contrast with $TD(\lambda)$. The general pheromone model suggests the possibility of propagating NAP among agents based on a topology in which they are embedded. If this topology reflects similar constraints on the appropriate level of activation of the various agents, it would be worthwhile for them to share information about the right level of activity. This sort of information sharing makes sense in both of our example problems. The graph coloring problem directly offers a topology in which nearby agents should have similar activation rates, while those more distant from one another might well differ in the appropriate frequency of activity (for example, due to the local nature of an intrusion into the sensor network). In the mission scheduling problem, one might instantiate requirements rather than tasks as the agents, and connect all requirements in a single task together, so that if one learns it should slow down, the others would also. This topological sharing of information has no parallel in $TD(\lambda)$. We have not explored it in our experiments so far.

3.2 Exploration-Exploitation Shift for Deadline Management

Deadline management poses a special case of the exploration/exploitation tension. The closer the deadline, the more preference should be given to exploitation, since there is less and less time to

take advantage of lessons learned from further exploration. Thus it makes sense to vary an agent’s activities based on deadline proximity. Our two examples illustrate two possible approaches.

In some systems, the difference between exploitation and exploration can be identified with the specializations of individual agents. For example, in mission scheduling, a task whose requirements are consistent with the presumed configuration of the system is exploitative, while one that has been found inconsistent in the past may still be useful in exploring the system for possible changes. In this case, as a deadline approaches, we want to discourage an agent that has been unsuccessful from competing for resources. This capability can be provided elegantly within the NAP mechanism by decreasing the threshold θ as the deadline approaches, thus increasing the ratio of NAP to threshold for any given NAP level and correspondingly reducing the probability that the agent will activate. We apply this approach to deadline management in the mission scheduling problem in Section 4.

In other systems, each agent may be capable of both exploratory and exploitative actions. In this case, the agents choose between these actions stochastically, and the probability of choosing an exploratory action decreases as the deadline grows nearer. We demonstrate this approach in the graph coloring problem in Section 5.

4. APPLICATION: MISSION SCHEDULING

In the mission scheduling application, the evaporation factor $E = 0.5$, and a task agent makes a deposit each time it loses a bid for a resource. The amount of the deposit is

$$deposit = \frac{1}{1 + e^{\frac{1 - bidave / VPR}{pressure}}}$$

where

- *pressure* represents the amount of contention for a particular resource, and in these experiments is simply the number of bids on the resource;
- *bidave* = (sum of all current bids on resource) / (number of current bids on resource)
- *VPR* is the “value-per-resource needed,” equal to the total task value divided by the number of requirements (and thus the number of needed resources).

VPR is global information, but static for a given configuration of the problem. *Pressure* and *bidave* are information that can reasonably be returned by a resource in its response to a task’s bid.

Our experimental matrix consists of 14 configurations differ-

ing in the number of tasks: $T \in \{10, 20, 30, 40, 60, 80, 90, 100, 110, 120, 130, 150, 250, 400\}$. The number of requirements R is fixed at 100, the number of requirements $Q = 4 * T$, the number of engagements $L = 9 * Q$, so that the density (the proportion of cells in a table such as Table 1 that are occupied with X’s) is constant at .09. We exercise seven algorithms, summarized in Table 2, across these configurations, and report average results for 11 replica-

Table 2: Enhancements of MarbleSize

Version	Bid Skip-ping	Enhanced Dropout	Deadline Re-sponse
MarbleSize	no	no	no
MarbleSize w. hstop	no	no	hardstop
MarbleSize w. roff	no	no	rolloff
Siggy	yes	yes	no
Siggy w. roff	yes	yes	rolloff
Quicksiggy	no	yes	no
Quicksiggy w. roff	no	yes	rolloff

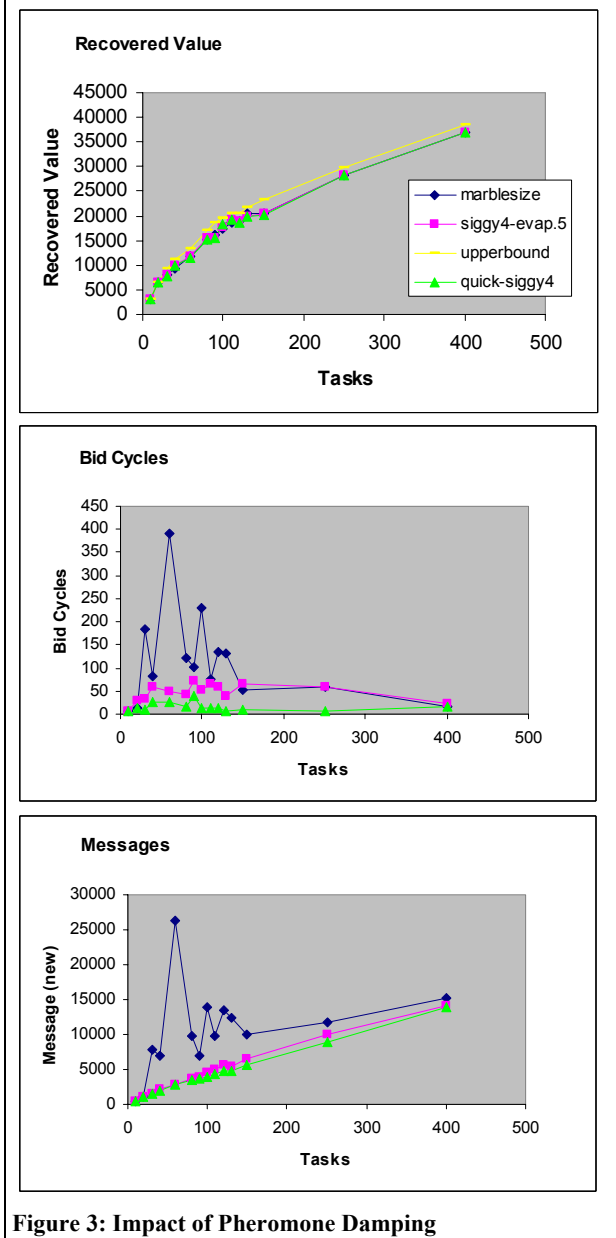


Figure 3: Impact of Phormone Damping

tions of each configuration. The algorithms differ along three dimensions:

Bid-skipping indicates whether a task agent skips bidding cycles probabilistically based on its NAP.

Enhanced dropout indicates whether a task agent dies when its NAP exceeds its threshold.

Deadline response indicates whether the system has a deadline, and if so, whether the deadline is managed by gradually lowering the threshold (“rolloff”) or simply by stopping the run at the threshold (“hardstop”).

“MarbleSize” is the algorithm reported in [6], and serves as a baseline for the other mechanisms. We discuss first the mechanisms without a deadline, then those with one.

4.1.1 Non-Deadline Experiments

Figure 3 summarizes the results of the experiments with no deadline. Each chart compares the performance of the original Mar-

bleSize solver with two enhanced versions. Siggy uses the NAP for both bid skipping and task dropout. Quicksiggy uses just the task dropout. We observe:

- All solvers performed at near ideal levels in terms of total recovered value. The maximum possible recovered value, the UpperBound, is drawn as a reference.
- Siggy show significant improvement in reducing the number of bid cycles required to solution. Quicksiggy did even better.
- Both modified solvers showed significant improvement in message reduction to the point that the number of messages required is approximately linear with the number of tasks (and requirements) in the problem.

Use of the NAP results in quickly identifying less desirable tasks and removing them from the bidding processing. By doing so, both messaging and cycle time are significantly reduced with little or no loss in quality of solution (i.e., total recovered value).

These results confirm and extend our previous results [13] showing that the effort profile in a constraint optimization problem depends on the algorithm used, and not just on the nature of the problem. In that paper, we cited MarbleSize as an example of a constraint optimization algorithm with an easy-hard-easy effort profile. The application of phormone learning greatly lowers the “hard” part of this profile in terms of bidding cycles, and replaces it with a steady linear increase in terms of messages exchanged.

4.1.2 Deadline Experiments

A second set of experiments (Figure 4) explores the introduction of a deadline into the solver process, requiring that a solution be delivered in a fixed number of cycles. The graphs compare quality of solution for each of solver when a deadline is imposed. The metric of quality, “% change in recovered value w.r.t. no deadline,” indicates the change in recovered value measured as a percentage of that recovered in the absence of a deadline. For example, in the first plot (“Marblesize w/ Hardstop”), the asterisk at 30 tasks and -17 indicates that with a deadline of 10 cycles, this algorithm recovered only 83% (100% - 17%) of the value recovered in the absence of a deadline. (Recall from the first graph in Figure 3 that with unlimited deadlines, all algorithms recover the same amount at each level of tasking.)

The algorithms other than “hardstop” versions use the NAP with monotonically decreasing $\theta(t)$:

$$\theta(t) = \text{Min}\left(\sqrt{\text{Deadline} - t/2}, 1\right)$$

As the solver approaches the deadline, the threshold for task dropout decreases, making dropout more likely. At the deadline, the threshold is zero, making dropout a certainty for all unsatisfied tasks. Solvers that make task dropout decisions early, such as Siggy and Quicksiggy, clearly outperform a hardstop.

5. APPLICATION: DYNAMIC GRAPH COLORING

Figure 1 shows that performance is worst in the high AL region, so our basic control mechanism will be to reduce AL. Even when AL says that a node should evaluate its color assignment, it will not evaluate with probability NAP/θ . We have explored two locally accessible metrics to define the size of the deposit. In both cases $E = 0.9$. θ is defined as the asymptotic level that the NAP would reach if the maximum deposit were made at each step. Since each mechanism has a maximum deposit of 1 per unit time,

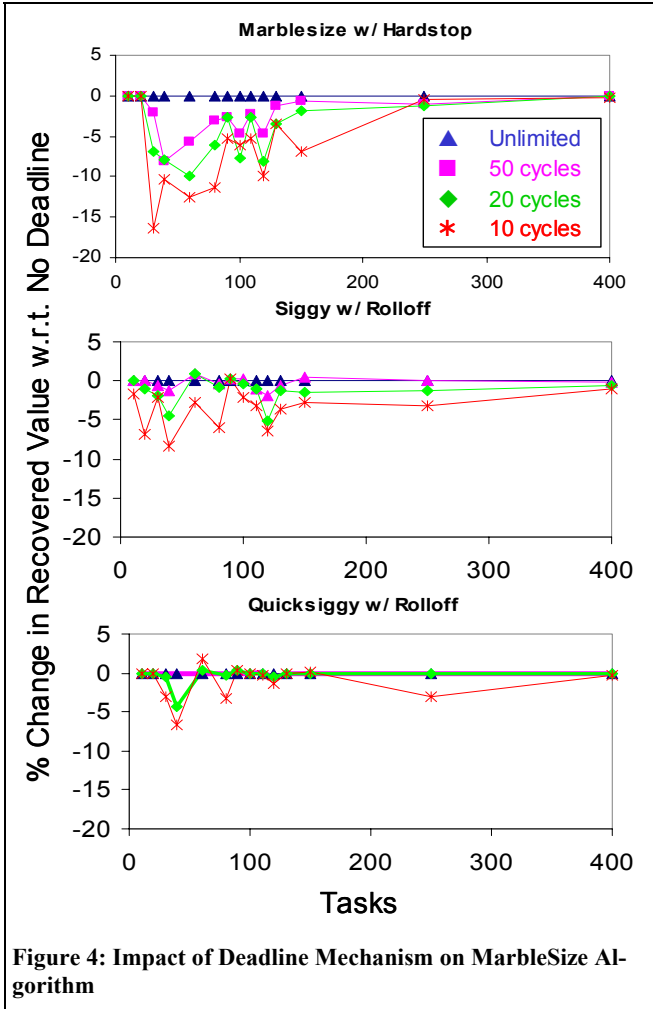


Figure 4: Impact of Deadline Mechanism on MarbleSize Algorithm

the analytic results in [2] predict $\theta = 1/(1 - .9) = 10$ for a single mechanism or 20 for the joint operation of the two.

A node's **option set entropy** (OSE) is the entropy over the probability that a node chooses a particular color in a cycle, normalized by the maximum entropy ($\ln(G)$, defined by a random choice among all G colors). Given the MD and CS rules in this set of experiments, the selection probability is zero for all colors that do not result in the lowest local degree of conflict in a cycle. All remaining colors share the same non-zero probability, which add up to one. Thus if n colors offer the lowest DoC for a node at a given moment, the probability that the node will choose any one of them is $1/n$, the entropy is $-n*(1/n)*\ln(1/n) = \ln(n)$, and the OSE is $\ln(n)/\ln(G)$. The only information needed by the node to compute the OSE is G (which is set when the system is configured) and its local degree of conflict (which is local information). The node's OSE

reflects the level of guidance that it has in its local decision process. OSE = 0 indicates that there is only one option available, while OSE = 1 indicates a random choice across all colors. Experiments indicate that OSE tends to decrease as the system nears convergence, and continued agent activity yields no further improvements, making it a useful local estimator of system-wide behavior. To discourage agents from continuing to activate when no further improvement is possible, we deposit $1 - \text{OSE}$ each time the agent activates.

A node's **false information percentage** (FIP) is the proportion of neighbors for which the current color assumed in the node's decision process is not the neighbor's actual color. This difference results from communication latency, since a change in a neighbor's color requires some time (here always one cycle) to become known to a node. The metric reflects the impact of the physical reality of restricted information exchange on the decision processes of the nodes. As with OSE, FIP is based on locally available information. When FIP is zero, a node has perfect knowledge of the state of the environment, while a value of one indicates that all input to the local optimization function is misleading and the system is trying to change faster than information can be updated. To discourage action on inadequate information, we deposit FIP.

Figure 5 shows the impact of these mechanisms on both quality of solution (GDOC) and TTS. The ordinate in each plot is the difference between the metric without pheromone learning and that with pheromone learning. Both mechanisms increase TTS (lower row of plots), since the NAP effectively reduces AL and thus the system requires more cycles to converge. OSE learning provides slightly greater increase in solution quality than does FIP learning, but greatly increases TTS in the high-AL and high- G regime. Combining the two mechanisms (right-hand plots) yields an increase in quality as good as that achieved with OSE alone, but restricts the increase in TTS to the low- G region where the increase in quality is actually achieved. A companion paper [3] explores the dynamical characteristics of this system in more detail.

To achieve deadline control in the dynamic graph coloring

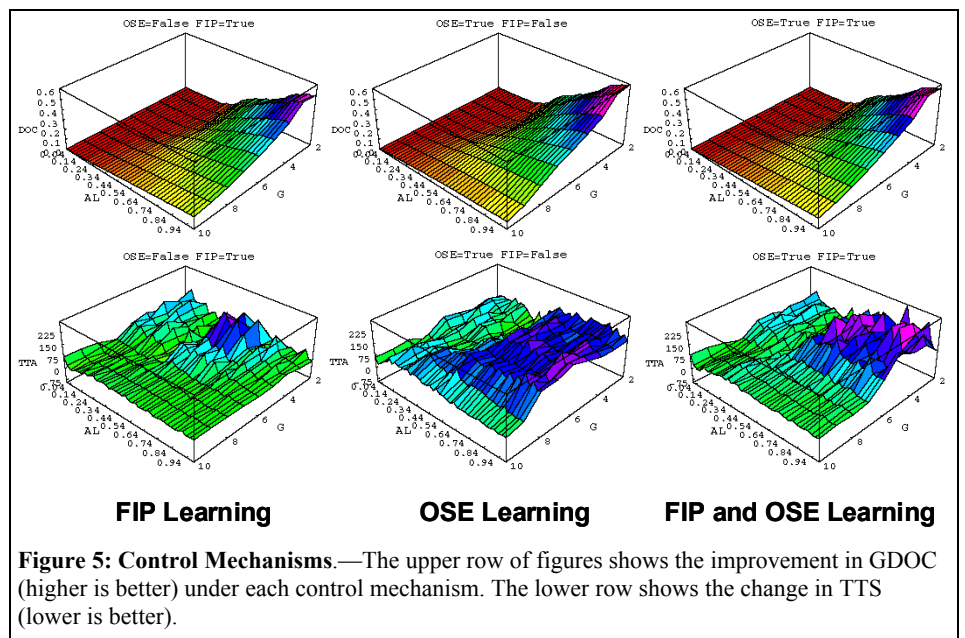


Figure 5: Control Mechanisms.—The upper row of figures shows the improvement in GDOC (higher is better) under each control mechanism. The lower row shows the change in TTS (lower is better).

problem, we let individual agents choose between exploratory and exploitative behavior. Recall the two-step nature of the color choice algorithm: first select a movement direction MD, then invoke a color selection rule CS. The color selection rule has available only the set of colors identified in the movement direction step.

Most of our experiments use MD = Any (allow a node to choose any of the available colors, even those that would increase DoC) and CS = Best (pick randomly among the colors that would yield the lowest DoC). An alternative MD rule is OnlyUp, which permits a node to choose only from colors that would reduce its DoC. The difference is subtle but important. MD = Any with CS = Best means that if there are other colors that would yield the same DoC as the current color, the node can select them, and thus move laterally in search space, *exploring* new configurations. MD = OnlyUp means that the set of colors on which CS operates includes the node’s current color and any colors that would yield a *lower* DoC, but no colors that would yield the *same* DoC as the current color. Thus the more frequently a node chooses MD = OnlyUp, the more rigorously it hill-climbs, *exploiting* the gradient it sees.

We apply deadline control by modifying the MD decision. With Only-Up Probability

$$OUP = (e^{a^*/\text{deadline}} - 1) / (e^a - 1)$$

where a is a tuning constant, we apply MD = OnlyUp instead of MD = Any. Figure 6 shows the result of this mechanism applied to a system with deadline = 1000. The two top plots show the evolution of DoC and OUP as the system runs. Without control (a), OUP remains at 0 and DoC quickly stabilizes around 0.39. With control (b), as OUP grows, DoC drops. The bottom plot (c) shows that runs with deadline control cluster around the level of the best DoC achievable with perfect knowledge, while uncontrolled results have much higher DoC.

Another way to look at Figure 6 is to observe that the behavior seen in the upper plot would also result if we had no deadline at all. That is, the use of OUP control does more than enable us to cope with a deadline. It actually improves the behavior of the system over the case where there is no deadline, by forcing the system to shift from exploration to exploitation and thus to zero in on a desirable solution. It thus plays a role analogous to that of temperature in simulated annealing [9].

6. CONCLUSIONS

One of the major benefits of an agent architecture is the localization of decision-making in cases where global information is difficult to obtain or distribute. This very characteristic can lead to agent actions that appear desirable from the agent’s perspective but that actually compromise system-level performance. The domain of resource allocation is particularly susceptible to such dynamic pathologies. Agents that persistently compete for resources even in configurations in which they cannot be satisfied make it more difficult for agents with feasible demands to be supported. Agents that update their local configuration more frequently than the system’s communication latency permits can initiate thrashing. Some degree of agent exploration is desirable when a system is changing, but when it is stable, and as firm deadlines approach, the system should shift to a focus on exploitation. These pathologies can be understood as agent hyperactivity that is related either to agent specialization or system stability.

We have developed two sets of mechanisms to calm hyperactive agents. One, inspired by insect pheromones, is appropriate for managing ongoing systems. It is based on the

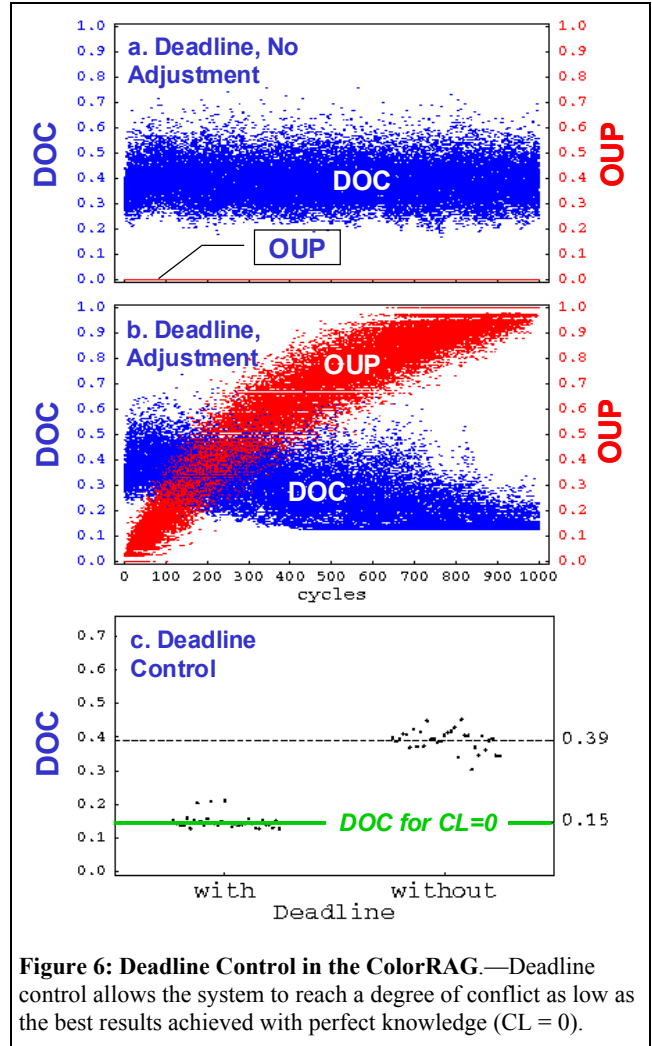


Figure 6: Deadline Control in the ColorRAG.—Deadline control allows the system to reach a degree of conflict as low as the best results achieved with perfect knowledge (CL = 0).

for managing ongoing systems. It is based on the accumulation and evaporation of “no-action pheromone” that is compared with a threshold to determine whether an agent should act on a given cycle. This mechanism can be viewed as a variety of temporal difference learning that is sensitive to the reward experience in a state, not just the state itself. The other mechanism enables the management of approaching deadlines, by shifting agent activity (either collectively or individually) from exploration to exploitation. In the collective case, the second mechanism can be implemented as a natural extension of no-action pheromones.

We have demonstrated these mechanisms in two problems: dynamic graph coloring and air mission scheduling. In both cases, the mechanisms enable these systems to achieve dramatic improvements in efficiency without compromising overall performance.

ACKNOWLEDGMENTS

This work is supported in part by the DARPA ANTS program, contract F30602-99-C-0202 to Altarum, under DARPA PM Vijay Raghavan. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government. We gratefully acknowledge the cooperation of Bob Neches, Alejandro

Bugacov, and Pedro Szekely at ISI for access to their mission scheduling package, and to Stephen Fitzpatrick at Kestrel Institute for consultations on their graph coloring model for sensor allocation.

7. REFERENCES

- [1] R. Axtell and J. Epstein. Distributed Computation of Economic Equilibria via Bilateral Exchange. Brookings Institution, Washington, DC, 1997.
- [2] S. Brueckner. *Return from the Ant: Synthetic Ecosystems for Manufacturing Control*. Dr.rer.nat. Thesis at Humboldt University Berlin, Department of Computer Science, 2000.
- [3] S. Brueckner and H. V. D. Parunak. Information-Driven Phase Changes in Multi-Agent Coordination. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 2003)*, pages (submitted), 2003.
- [4] A. Bugacov. SAT encoding of the single time-slot resource allocation problem. 2001. PDF file, http://www.isi.edu/attend/frames/Sat_encoding/sat_encoding_short_doc.pdf.
- [5] S. Fitzpatrick and L. Meertens. Soft, Real-Time, Distributed Graph Coloring using Decentralized, Synchronous, Stochastic, Iterative-Repair, Anytime Algorithms: A Framework. Technical Report KES.U.01.5., Kestrel Institute, 2001.
- [6] M. Frank, A. Bugacov, J. Chen, G. Dakin, P. Szekely, and B. Neches. The Marbles Manifesto: A Definition and Comparison of Cooperative Negotiation Schemes for Distributed Resource Allocation. In *Proceedings of AAAI Symposium on Negotiation Methods for Autonomous Cooperative Systems*, pages 36-45, AAAI, 2001.
- [7] T. Hogg and B. A. Huberman. Controlling Chaos in Distributed Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6 (November/December)):1325-1332, 1991.
- [8] J. H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, University of Michigan, 1975.
- [9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671-80, 1983.
- [10] L. Meertens and S. Fitzpatrick. Peer-to-Peer Coordination of Autonomous Sensors in High-Latency Networks using Distributed Scheduling and Data Fusion. Technical Report KES.U.01.09, Kestrel Institute, 2001.
- [11] R. Neches and P. Szekely. CAMERA Project: Coordination and Management Environments for Responsive Agents. 1999. Web Page, <http://www.isi.edu/camera/>.
- [12] H. V. D. Parunak. From Chaos to Commerce: Practical Issues and Research Opportunities in the Nonlinear Dynamics of Decentralized Manufacturing Systems. In *Proceedings of Second International Workshop on Intelligent Manufacturing Systems*, pages k15-k25, K.U.Leuven, 1999.
- [13] H. V. D. Parunak, S. Brueckner, J. Sauter, and R. Savit. Effort Profiles in Multi-Agent Resource Allocation. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS02)*, pages 248-255, 2002.
- [14] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, MIT Press, 1998.