Resource Allocation in a Fully Decentralized Market of Agents: the Mini-Mart Approach

Steve Burbeck
IBM, Research, RTP
SBURBECK@IBMUS
Internet: sburbeck@us.ibm.com

and

Sam S. Adams
IBM, Research, RTP
SSADAMS@IBMUS
Internet: ssadams@us.ibm.com

Revision History:
First Draft - 12/2/97
Second Draft - 1/x/98

TABLE OF CONTENTS

1 Introduction

1.1 Decentralized open systems

More than a billion computing systems are at work on the planet today (not to mention more than a thousand at work off the planet). Those explicitly identifiable as "computers", e.g., the palm-tops, notebooks, PC's, mid-range servers, and mainframes, are just the tip of the iceberg. The vast majority of computing systems labor anonymously in devices such as wrist watches, cameras, pagers, microwave ovens, cell-phones, automobiles, TVs, retail cash registers, and vending machines. Today most of these systems work in isolation. But webs of interconnections are forming within this collection of previously isolated computing systems. The diligent surfer of the World Wide Web can find examples of connected vending machines, remotely controlled video cameras, and a variety of other digital sensors and actuators. This trend may well continue until isolated digital systems are the exception rather than the rule. Moreover, within the growing webs of computing systems lives a growing population of mobile software agents, e.g., Java applets [cf., VT97]. As these software elements operate, they often communicate with software running on other machines. In short, the classical assumption that computing is about what happens inside individual computers is giving way to the notion that computing is about the interaction of many processors and processes in decentralized networks [Hub88, Weg97].

The Internet is the most visible and most important present day example of a decentralized network. The physical network of machines in the Internet changes more rapidly than any entity – human or cybernetic – can comprehend. Its topology and extent varies second-by-second as dial-up connections are made and broken. The connections themselves vary in latency, bandwidth, and reliability. Within the physical web, the web of software agents varies even more rapidly in response to external events such as human input.

Agents operating in this net sometimes need services or resources from other agents in the net. Their efforts to obtain the needed resources must surmount problems such as asynchronous operation, imperfect knowledge of possible collaborators and imperfect communication with other agents. This paper investigates some of the issues that arise from the need to allocate resources in such a chaotic environment. Because this environment is not unlike the economic environment of human commerce, the decentralized allocation mechanisms we explore here are based on an economic metaphor.

1.2 Decentralized resource allocation

Many authors have approached the problem of distributed resource allocation with analogs of economic trading [Sut68, FL72, FFH73, MD88, GS91, WHH92, CH93, FNS95, YWI96]. For example Kurose and Simha [KS89] point out that, "In the broadest sense, a distributed computer system can be thought of simply as a set of interconnected computing agents which require the use of certain system resources in order to perform their assigned tasks." They consider this resource allocation problem from the perspective of mathematical economics. Wellman [Wel96] has coined the term "market-oriented programming" to describe the process of building a computational market to solve allocation problems. He defines the economic approach to be inherently about 1) resource allocation, 2) rational self interest, and 3) decentralized decision making [Wel95a].

Researchers investigating market-oriented systems typically assume that individual economic agents make decentralized decisions in the context of a central market. The decentralized agents make independent decisions about the prices at which they will trade. Yet the trading decisions that actually allocate resources, i.e., those that match proposals to buy with proposals to sell, occur in a central market. This requirement for a centralized market does not scale well in decentralized computing systems. As the network grows in size and diversity, the communication and decision load on the central market becomes insupportable. The work reported in this paper uses distributed market mechanisms that trade effectively in a fully decentralized system, i.e., one without a central market.

Some prior work has been done on fully decentralized resource allocation. In the challenge ring protocol [GFS92], agents that provide resources are organized in a ring topology. Tasks that need resources circle the ring, challenging for use of those resource until every task is done. Waldspurger, et al. [WHH92] have done considerable work on a system called *Spawn* that implements a distributed computational economy. In their system, each agent operates its own auction to allocate its excess computational resources to competing bidders. It is not clear, however, how the design of either of these systems might solve more general resource allocation problems in larger and more chaotic networks.

1.3 A decentralized web of mini-marts

Resource allocation in an environment as swiftly changing as the Internet differs in at least four important ways from allocation in a fixed, tightly coupled system. First, as discussed above, a decentralized network of agents may not have a natural central clearing house or market in which a global allocation algorithm can decide which resources should be assigned to which agents. Second, the supply of and the demand for a given resource in the network does not necessarily remain constant while decentralized resource allocation takes place. Third, agents cannot expect to know the identity of or communicate with more than a small fraction of the other agents in the system. And fourth, agents cannot assume reliable communication with all potential collaborating agents. Therefore, in the present work, we assume that the agents exist within a random net topology so that agents cannot exploit any specific knowledge of the topology. We also assume that agents can communicate directly with only a small fraction of the other agents. All other communication is by forwarding which is fallible; forwarded messages may be lost in the net – just as in the Internet – so agents must take that possibility into account.

Within this network, each agent behaves both as a supplier, a consumer, and as a mini-mart. That is, *we bring the essence of a market to each agent*. This is an example of a general decentralization strategy that can be summarized as: replace a centralized X with a decentralized and loosely coupled collection of X's. An example of this strategy in distributed AI is to replace a central problem solver with a collection of loosely coupled problem solvers [e.g., DS83].

A decentralization strategy presents three challenges for the system designer. First, individual agents must somehow communicate with each other about what is needed to accomplish the global task. Second, individual agents (as well as outside observers of the system) often need a way to learn, at least in summary, what other agents are doing or have done. And third, the strategy of replacing a centralized algorithm with a decentralized collection of agents that implement the algorithm is overly simplistic. In general, decentralized agents need some but not all of the behavior of the central algorithm. And they need additional behavior to compensate for the lack of a tightly coupled central decision algorithm. A good decentralized design differs from a poor one primarily in the selection of behaviors given to the individual agents. This paper examines the effectiveness of different choices of trading behavior in individual mini-mart agents.

1.3.1 Communication and trading behavior:

In all of the simulation work presented here, agents use the following protocol for communicating with one another and for initiating and consummating trades.

- Desires to trade are specified by proposals. Proposals, are of two kinds: *bids* that offer to buy a given amount of a commodity at a given price, and *asks* that offer to sell a given amount of a commodity at a given price. Proposals are commitments in the sense that an agent may have at most one bid or one ask for a given commodity active in the system at any time.
- Each agent is connected to a small number of others (typically 3 to 6) by a random network of asymmetrical (i.e., one way) connections. Agents communicate with each other by sending trade proposals and/or trade announcements over these connecting arcs.
- Trades are consummated by direct communication between the agent with commodity to sell and the agent that wishes to buy. Each proposal contains a reference to the proposing agent to allow this direct communication.
- Messages from a given agent propagate beyond the set of agents directly connected to it by message forwarding. After each agent processes incoming messages and sends any necessary proposals of its own, it forwards remaining proposals to randomly chosen connected agents. Forwarding is subject to the restriction that at most one bid and one ask can be sent via each outgoing connection. Thus agents forward proposals only if they have extra capacity after sending their own proposals. Messages not forwarded are permanently forgotten, i.e., they are not saved to be forwarded at a later time.
- Proposals have lifetimes, chosen by the proposing agent, after which they expire and are invalid. Once an agent sends a proposal to a neighbor, it cannot know the fate of the proposal unless and until another agent contacts it directly to consummate a trade. If a proposal is not consummated before expiring, the agent reissues a new proposal. Since forwarding is random, the new proposal will traverse a different path through the net and therefore encounter a different set of possible trading partners.
- Prices are chosen to facilitate trades rather than to maximize net worth. Bids are priced at the highest price an agent can afford (given its total goals) and asks are priced at zero. Thus a bid price for a given commodity always satisfies the price of any ask for the same commodity. The actual trade price is chosen to be halfway between the bid price and the ask price. Therefore agents prioritize on a first come first served basis without regard for maximizing their net worth.
- Announcements are generated and sent to neighbors to inform them about events such as trades, proposal expirations, and dropped proposals that may be of general interest. These announcements allow agents and the experimenter to estimate system effectiveness.

In summary, agents live in a randomly connected network through which proposals and announcements are forwarded via random paths. If a bid encounters an agent that can satisfy the bid or an ask encounters an agent that needs the offered resource, a trade occurs. When events occur that might interest other agents, announcements are created and sent to neighbors.

Some of the above choices have appeared elsewhere. The notion of sending out bids and asks that expire after a certain time appears at least as early as Reid Smith's Contract Net Protocol [Smi80]. Our decision to connect each agent to a small number of others is similar to the Spawn system which uses a variable (although not random) topology in which, "… each machine is only connected to a small number of other machines, demanding highly decentralized decision-making with no global state or controls" [WHH92]. That system also is an example in which agents can be either a buyer or a seller as the situation demands.

We depart significantly from prior work in the use of random fallible message forwarding for most inter-agent communication. We use random forwarding for three reasons, 1) it avoids the extremes of either hardwiring the communication topology or using broadcast -- neither of which are practical in large chaotic networks; 2) it guarantees that the system cannot take advantage of special topologies except insofar as a random topology is itself special, and 3) it is not unlike the realities of the Internet. The fallibility of forwarding is intended in part to deal with the possibility that agents may choose not to forward messages for reasons of self interest and in part because large scale networks in the real world will "lose" messages in any case.

Another significant departure is our treatment of price. Most of the work in economic modeling attempts to understand how autonomous agents acting in a market arrive at prices that cause their goals to be met (often at a market clearing price). The usual assumption is that, in addition to the goal of acquiring necessary resources, agents have a subsidiary goal of maximizing their net worth. Price mechanisms provide a means for prioritizing competing demands, especially when demand exceeds supply [FYN88, MD88, WHH92]. However price is only one way to prioritize. The casual assertion that it is superior to other approaches has not been formulated precisely in the context of allocating computing resources, nor has it been investigated in any systematic way. In fact the use of price for prioritizing may be detrimental to resource allocation if an agent's goal of maximizing its net worth conflicts with possible reallocation of resources. Spawn [WHH92] solves this problem by trading at zero price if there is no competition for a resource at a given node. We solve it by ensuring that asks are priced at zero (even though the actual sale takes place midway between the bid and ask price). Thus, in effect, we prioritize on a first come, first served basis.

Announcements address another issue that doesn't seem to be explored in the literature: that of providing generally available information about how the market is performing. For example, in the case of stock markets, one has available a constant flow of price and volume information. Some stock markets also provide more esoteric information such as estimates of short interest and details about insider trading. Announcements provide for the decentralized collection and diffusion of this sort of information.

1.3.2 Distributed market behavior

The above basic behaviors are sufficient prerequisites for distributed trading to take place. We implement a system (hereafter called a *BidAndAsk* system) that uses nothing more than these behaviors. It demonstrates that a completely unorganized collection of agents can trade with one another by nothing more than random forwarding of bids and asks. However the purpose of this paper is to investigate agents that incorporate some selected market behaviors in addition to the basic BidAndAsk behavior. We judge the effectiveness of three kinds of distributed market behaviors in comparison to the baseline provided by the BidAndAsk system.

- *Crossing* – Before forwarding proposals, a crossing agent checks those proposals for cases where a bid and an ask are for the same commodity. If any such matches are found, the agent immediately initiates a trade between the proposing agents (since references to the agents are available in the proposals). This kind of behavior is common in financial brokerages where it is often called trade crossing.
- *History* – The above crossing mechanism is limited to initiating trades between proposals that happen to reach a crossing agent at the same time. Crossing becomes more effective if we add a history mechanism to each agent that maintains a collection of unexpired proposals that have reached it in the past. By including this history, the agent is likely to find a larger number of crossing trades among the proposals of which it is aware.
- *Brokering* – In human economies, especially where central markets are lacking, there are often intermediaries between the producers and the consumers that buy a commodity expressly for later resale. Here we call such an intermediary a broker. They exist because direct trade between producers and consumers may be hindered by factors such as distance, poor information flow, or mismatches in scale (e.g., the producer sells by the ton whereas the consumer buys by the pound). In our work, information flow is the major barrier. The brokering mechanism we investigate works in the following way. When an agent receives a bid for a commodity that it neither has in excess nor requires for itself, it not only forwards the bid but also remembers the identity of the bidder and begins attempts to buy the same commodity for resale. If its attempts meet with success, either because a matching ask arrives or its bid finds a match, it immediately buys the commodity and offers to resell to the remembered bidder.

A crossing agent accepts responsibility for initiating trades other than those that satisfy its own direct needs, and in doing so plays the role of a mini-mart. As proposals are forwarded from one agent to another, they participate in one mini-mart after another. When we add history, each agent acts as a larger and therefore more effective mini-mart. Moreover, as a proposal wends it way via forwarding through the random net, it leaves behind a trail of agents that remember it and will match it to compatible proposals that arrive later. So not only does each agent become a larger and more effective market, but also each proposal participates in many of these markets at the same time. Out of this combination of forwarding, crossing, and distributed memory, a pseudo-global market emerges.

Brokering is a fundamentally different mechanism from crossing. It does not facilitate the matching of bids and asks. In fact, to some extent, broker's bids compete with bids from agents attempting to satisfy their own goals. Rather, brokering causes resources to change hands for reasons other than the direct satisfaction of goals. Purchases by a broker are on behalf of another agent. The broker is, in a sense, closer to the source of demand. Hence resources move "toward" demand within the net of agents.

In the following simulation work we investigate the effectiveness of these mini-mart mechanisms for solving two trading tasks. In the first task a set of agents must achieve an externally prescribed allocation of resources starting from an initial random distribution of resources. The effectiveness of each trading strategy is measured by the number of trading iterations needed to meet the prescribed goals. In the second task, goals are changed at a constant rate as the trading progresses so that trading is never finished. We compare strategies after the system has run long enough to reach a stochastic equilibrium by measuring how long it typically takes for an agent to satisfy a newly set target for a single commodity and measuring the proportion of agents with satisfied goals. For both tasks we investigate general issues such as how the effectiveness of the system is affected by scale (i.e., the number of agents in the system) and by the relationship between total supply of and total demand for a given resource.

2 Simulation Experiments and Results

2.1 The trading simulation framework

The market simulator is a specialization of a general object-oriented discrete-time simulation system developed by the authors in Smalltalk. This general system was created to investigate the global effects of message mediated collaborations between various kinds of agents in various kinds of networks. It has been used previously for investigating consensus formation in systems of randomly connected voting agents and for investigating avalanche behavior in systems of threshold crossing agents in self organizing criticality experiments [for a general introduction to self organizing criticality see BC91].

The general simulation system creates appropriate sets of agents, installs connections between them, steps the system through the discrete time steps until a stopping condition is met, displays the system as it operates, and gathers results. Starting with this general framework we implemented a network of **N** market agent nodes connected to one another by a random net. Each node is the origin of a fixed number, **C**, of unidirectional, i.e., asymmetrical, connections. The target of each connection is randomly chosen from the remaining nodes. The distribution of incoming connections is Binomial with a mean of **C** and a variance of
C(N - 2)/(N - 1). For small values of **C** an agent may well have no incoming connections (e.g., for **N** = 100 and **C** = 3, the probability that any given agent has no incoming connections is .049). But even so, outgoing bids and asks still provide a sufficient, although less efficient, means for trading. In the rare case where a trade is required between two agents that both lack incoming connections, the BidAndAsk strategy cannot make the trade. Crossing removes that barrier to trading because outgoing proposals from the two agents can be crossed by the many other agents that have an incoming path from both agents.

In recognition of the fact that real world agents often require more than one resource to carry out their function, the market agents we investigate are given the task of trading three commodities. Thus each agent owns some amount of each commodity together with some amount of “currency” with which to purchase more. Since these commodities are abstract resources that do not affect the operation of the trading system, we chose the three primary colors (red, green, and blue) as the commodities to be traded. This choice means that we can map the amounts of the three primaries owned by a given agent to a single color and directly observe the progress of trading via a graphical user interface to the simulation.

In the simulation, currency is a number that represents some abstract store of value that can be traded for any other commodity. Each agent begins with the same amount (three units) of currency that it uses to buy commodity from other agents. Given our approach to pricing, the movement of currency and the prices of trades is of no further interest here.

The purpose of trading by market agents is to satisfy goals set by the system. The simulation results reported below were all done with goals assigned to half of the agents. Thus in general half of the agents are buyers and half are sellers (however, if they are acting as brokers, all may be either buyers or sellers). An agent’s goal is expressed as a target amount (between zero and one) of each of the three commodities. If an agent has no goal, it is willing to sell all of its resources. If an agent has a goal, it buys to meet unmet portions of the goal or sells whatever it has in excess of the goal. Since goals are simply bundles of the three commodities, they too are colors that can be directly observed.

Market agents delegate all significant trading decisions to a *policy* object (for more on the general use of policies see the Strategy pattern in [GHJV94]). We investigate four classes of trading policies: 1) the BidAndAsk policy which implements all the basic trading behavior discussed above, 2) the Crossing policy, a subclass of the BidAndAsk policy, also implements crossing behavior, 3) the CrossHistory policy, a subclass of the Crossing policy, also maintains and uses a history of all valid proposals previously seen, and 4) the CrossHistoryBroker policy, a subclass of CrossHistory policy, also does brokering.

Crossing behavior is implemented in the agent itself, but is used only if the policy permits. The BidAndAsk policy does not permit crossing. The Crossing policy and its subclasses do. History is implemented directly by the policy, i.e., the CrossHistory policy takes responsibility for remembering all untraded proposals that have been received. This historical record supports historical crossing and brokering.

Policies provide a wide range of other behavior for their agents. The policy determines the amount of a given commodity needed by the agent. This amount determines whether an outgoing bid should be generated or an incoming ask should be accepted. The BidAndAsk class simply answers zero if the agent has no goal or if the goal is met. If it has an unfulfilled goal it answers the difference between the amount held by the agent and the amount required by the goal. The CrossHistoryBroker policy expands this behavior to allow bids for the purpose of brokering to be generated if the goal is met or the agent has no goal.

Policies also compute the bid prices and the ask prices and set the expiration times for proposals. In the following work, expiration times are fixed for a given simulation run and shared by all agents in that run. All policies use the same rule for generating prices. In this work we do not want the fundamental dynamics of the trading process to be obscured by pricing issues. So policies always set the asking price to zero to guarantee that trades are never blocked by price mismatches. Bid prices are set at the highest price consistent with available currency, with the needs for all three commodities, and with outstanding bids (i.e., bids are considered to be commitments for which currency must be reserved).

Agents send at most one bid and one ask to each directly connected agent. Any proposals in excess of that capacity are simply forgotten. The choice of which proposals are to be sent is made by the policy. The rule used here is that all proposals generated by the policy's agent are given priority over proposals from other agents. Within these two categories, proposals for larger amounts of a commodity are given priority over those for smaller amounts.

The trading process is a discrete time stepwise process. On each time step, all agents receive messages, then all agents do their trading, then all agents send messages to be dealt with on the next step. The order in which agents are allowed to trade is arbitrary and has no systematic effect on the outcome of trading. The process each agent goes through at each time step of the simulation is as follows:

1. Examine incoming bids, discard any that have expired, and sell any available excess if it is bid for. Similarly, examine incoming asks, discard any that have expired, and buy if appropriate. The order of processing of bids and of asks is by order of arrival.
2. Invalidate, i.e., mark as expired any existing self generated proposals that have been rendered unnecessary by trades executed in step one.
3. Do policy directed crossing and update history records if appropriate.
4. Save any incoming proposals that have not been satisfied for forwarding.

5. Process incoming announcements (see below),
6. Generate new bids to satisfy unmet goals for which there are no valid outstanding bids. If the policy specifies brokering, generate bids for brokering purchases.
7. Prioritize outgoing proposals, placing self generated proposals ahead of forwarded proposals. Within those two categories, prioritize by the amount of commodity involved.
8. Send as many outgoing bids and outgoing asks as there are connections – one of each per connection – assigning proposals to connections randomly.
9. Forget any bids that cannot be sent (i.e., they are dropped rather than saved for later forwarding).
10. Update the consolidated announcement (see below) and send a copy to each directly connected agent.

This process continues step by step until all goals are met.

The collection and consolidation of information about the trading is necessarily decentralized as well. Each agent maintains an object, called an *announcement*, which records what the agent knows about its own trading plus what it has learned from other agents about general market conditions. At each time step, these announcements are passed on to other directly connected agents in the network. An announcement is a time ordered collection of objects, one per time step, that summarize what is known about trading at that time. Announcements serve somewhat the same purpose as a stock market ticker tape, so the record for each time step is called a *tick*. In the present work, ticks record nothing more than the number of expiring proposals and forgotten (i.e., dropped) proposals. Since price is deliberately made unimportant in these experiments, ticks do not record trade prices.

Agents consolidate the information from the corresponding ticks in all incoming announcements into a new set of ticks in a new announcement. The incoming announcements are discarded after consolidation. After the agent is done trading, ticks older than a preset limit (currently 3 steps) are discarded and a new tick containing the information from the current agent on the current step is appended to the announcement's collection of ticks. Then the updated announcement is sent to all connected agents. Approximately **NC** announcements flow through the system in each time step. With a three step record, the consolidated information received by each agent includes information from as many as $\mathbf{C}^3$ other agents (assuming $\mathbf{C}^3 << \mathbf{N}$). Thus iterated consolidation and local broadcast of announcements provide each agent with a relatively wide summary view of trading.

2.2 Trading with static goals

We first investigate the task of trading to satisfy a static set of needs. In this task, each agent's goals remain constant during the trading process which continues until all goals are satisfied. The assumption that goals remain static while resources are allocated is a common one in the literature [e.g., GFS92, Wel95b, SHC96]. This scenario mimics the real-world situation in which resources must be allocated so that a set of agents can work in concert. Therefore all must obtain needed resources (i.e., satisfy their goals) before their task can be performed.

We evaluate the effectiveness of a given policy and given set of parameters by the mean number of steps required for a solution (i.e., satisfaction of all goals) in 20 simulation runs. Each set of runs begins by creating the agents, creating the policy for each, then creating the random net of connections. In the following simulations, we use systems with 100 agents (except when investigating scaling where the number of agents is varied systematically) with either three or six outgoing connections per agent. In any given run, all agents use the same class of policy. Each of the 20 runs begins with a different random allocation of the three color commodities to all agents. Then goals are assigned to half of the agents. Half must buy to fulfill their goals, and half are willing to sell all of their randomly assigned resources. The amount of each commodity specified by each agent's goal is the same. That amount is chosen so that the total demand specified by the goals is precisely related to the total available supply of each commodity (e.g., total supply exceeds total demand by some predetermined amount).

In addition to studying the effectiveness of the four policies, we also investigate the following issues:

- Supply versus demand – In the static goal task, total supply must at least equal total demand for each commodity or no solution is possible. If they are equal, the last trade simultaneously exhausts available supply and satisfies the last goal. If total supply exceeds total demand, the allocation task is less difficult. We investigate the sensitivity of the different policies to this degree of difficulty.
- Expiration – The number of steps before proposals expire determines the maximum number of other agents a proposal can reach. Different policies have different sensitivity to this parameter.
- Number of connections per node – This parameter affects the number of paths between any given pair of nodes. It also affects the probability that a proposal will be dropped before it expires since proposals are dropped when there are more to be forwarded than there are connections on which they can be sent.
- Scale – The number of agents in the system affects the probability of a proposal reaching any given agent.

2.2.1 Effect of excess supply and expiration

Observation of trading shows that these systems consummate many trades per step in the early stages of the process. As trading progresses, the number of possible trades decreases and the difficulty of matching bids and asks increases. The time expended making the last few trades tends to be a substantial proportion of the total time. Thus the mean number of steps to solution with static goals depends strongly on the effectiveness of the policies in finding trades when most goals are met. The difficulty in making the last few trades is, in turn, strongly affected by the degree to which total system supply exceeds total system demand. We therefore study the effect of varying excess supply over demand from zero to four percent in steps of one percent. To avoid the possibility that rounding errors during trading might inadvertently cause supply to fall slightly below demand, we use 0.001 percent excess supply in the nominal zero excess condition.

Expiration has a substantial effect as well. Figures (1 - 4) show both effects together.

The performance of the BidAndAsk policy is rather poor (see Figure 1), which should be expected from its simplicity. It requires an average of more than one hundred steps to complete in all conditions except the most favorable (four percent excess, expiration of seven) which takes an average of 75.4 steps. However, the notion of poor performance is relative. Given the difficulty of encountering a matching agent by nothing but a random walk, perhaps one should be amazed that it works as well as it does.

With the Crossing policy (Figure 2), the addition of one percent excess supply cuts the number of steps to half that required with supply equal to demand. The addition of another one percent supply cuts the number of steps in half again, but adding more supply past two percent excess has relatively little effect. We see something like a plateau, at roughly 50 - 70 steps to completion, for all conditions with three percent or more of excess supply and expiration more than three. Within the plateau, improvements from adding more excess supply or longer expirations are relatively minor.

We artificially limited the simulations to no more than 1000 steps per run. For the condition with zero excess and expiration of two, both the BidAndAsk policy and the Crossing policy exceeded that limit.

The general trend for the CrossHistory policy (Figure 3) with expiration greater than two shows a 40 percent reduction in the number of steps to completion when increasing supply from zero to one percent excess. Adding another percent of excess supply reduces steps to solution by another 20 percent. But further excess brings smaller improvements. The trend for expiration of two is more exaggerated. With three percent or more of excess supply and expiration of five or more, the number of steps to completion is roughly 24 - 34.

In difficult conditions, the CrossHistoryBroker policy (Figure 4) is considerably more effective than the CrossHistory policy at solving a static set of goals. For example, at one percent excess with expiration of three, the CrossHistory policy requires an average of 133 steps whereas the CrossHistoryBroker policy takes an average of 45 steps. In the region where they both have a relatively easy task, (i.e., excess greater than two percent and expiration greater than three), there is little to choose between them. At four percent excess and expiration of seven, the CrossHistory policy takes an average of 24.4 steps and the CrossHistoryBroker policy takes an average of 23.7 steps.

It should be noted that systems using the BidAndAsk policy with 100 agents and six connections need a minimum expiration of three to function effectively at all. Because the BidAndAsk policy trades only when a bid or ask reaches an agent that can trade directly with it, the policy must not expire before it can reach the target agent. With six connections and expiration of two, a proposal from one agent can reach at most 36 of the 99 other agents.

Expiration has a much bigger effect with supply equal to demand in both Crossing and the CrossHistory policies than it does with the BidAndAsk policy because expiration magnifies crossing opportunities, i.e., at each step a proposal has a new opportunity to cross with other proposals. However, expiration makes little difference in easier conditions (e.g., with one percent excess or more and expiration of three or more).

Only the CrossHistoryBroker policy of the policies studied here can trade with expiration of one. This is noteworthy because without forwarding no historical crossing can take place. Brokering combined with crossing can substitute for forwarding and history when necessary. A chain of directly connected neighbors, acting as brokers, buys a commodity from an agent with excess then successively resells it until it reaches the agent with the goal.

In general terms, as measured by mean steps to solution, the Crossing policy is about twice as effective (i.e., requires half as many steps) as the BidAndAsk policy in both the zero excess condition and the three percent excess condition (See Tables 1-2). The CrossHistory policy is roughly twice again as effective as the Crossing policy. The CrossHistoryBroker policy improves further on the CrossHistory policy but the improvement is, in most conditions, relatively small.

	expiration =	**expiration =**	**expiration = 7**

	3	5	
BidAndAsk	612	590	431
Crossing	399	271	198
CrossHistory	202	94.8	65.4
CrossHistoryBroker	119	83.5	65.9

Table 1: Mean Number of Steps with 0.001 percent excess supply

	expiration = 3	**expiration = 5**	**expiration = 7**
BidAndAsk	**233**	**137**	**136**
Crossing	**105**	**62.1**	**68.5**
CrossHistory	**46.1**	**34.0**	**27.8**
CrossHistoryBroker	**30.7**	**30.7**	**25.5**

Table 2: Mean Number of Steps with 3 percent excess supply

2.2.2 Effect of scale

The above described simulations all used systems with 100 agents. However the number of steps to find a solution can be affected by the number of agents in the system.

With supply essentially equal to demand, the difficulty in arranging a trade between the last agent to have an unfilled goal and the single agent that can provide the commodity to fill that need depends on the number of nodes in the network. One measure of the relative effectiveness of the various trading policies is their sensitivity to the size of the system. We obtain mean steps to solution from runs (with 0.001 percent excess, six connections per agent, and expiration of seven) in systems varying in size from 25 to as much as 2500 agents and fit functions of the form $\mathbf{aN^b}$ to these data (Table 3).

Policy	**Scaling Function**
BidAndAsk	$1.2\ N^{1.3}$
Crossing	2.4 N
CrossHistory	$0.87\ N^{0.96}$
CrossHistoryBroker	$2.9\ N^{0.7}$

Table 3: Scaling function , 0.001 percent excess, 6 connections, expiration of 7.

The effectiveness of policies as indicated by the exponents of the scaling functions shows the same order found with prior measures. Each successive addition of more "intelligent" market behavior lowers the exponent. The addition of crossing, which is the first behavior that fosters an emergent pseudo-global market by adding trading on behalf of other agents, makes the biggest reduction in the exponent (0.3). History helps scaling much less (0.04) although it reduces the linear coefficient (**a**) by almost a factor of three. The addition of brokering improves the exponent almost as much as does the initial addition of crossing. The time to solution required by agents using the best policy, the CrossHistoryBroker policy, grows a little faster than square root of **N** (see Figure 5). Note that the difference between an exponent of 1.3 and one of 0.7 is dramatic in large systems. For example, $1000^{1.3}$ is about 8000, whereas $1000^{0.7}$ is about 125.

In the case where there is excess supply, the situation is rather different. No longer is it a matter of the last agent with a goal trading with the last agent that has the needed commodity. The last agent with a goal can trade with several other agents (the number depends upon how much excess is available). Moreover, in random networks with random assignment of resources, any excess is randomly sprinkled about. Therefore its "nearness" to a given agent should not depend greatly on system size. And so it is. With three percent excess, six connections, and expiration of seven, all policies asymptote to a constant number of steps independent of the number of agents in the network. Figure 6 shows the effect of scale with three percent excess supply for the CrossHistoryBroker policy. All the policies behave similarly. What differs from one policy to the next is the asymptotic number of steps to solution (Table 4).

Policy	Steps at Asymptote
BidAndAsk	220
Crossing	110
CrossHistory	43
CrossHistoryBroker	35

Table 4: Asymptotic effectiveness (3 percent excess, 6 connections, expiration of 7)

This measure of the effectiveness of the various policies shows a pattern very similar to that found when examining the effect of excess supply and expiration: Crossing doubles the effectiveness of the simple BidAndAsk policy. Adding history more than doubles it again. Adding brokering makes a much smaller improvement (roughly 20 percent).

2.2.3 Effect of connectivity

The number of direct connections each agent has to others (**C**) affects the process of trading in two ways. First, it determines the number of paths between pairs of agents. Second, the number of connections limits the number of proposals that can be forwarded (hence the number that will be dropped). Both of these factors affect the ability of agents to easily communicate with one another. The question here is how well each of the different trading policies copes with differences in connectivity. In all of the above work, the systems have six connections per agent. Here we compare the performance of systems with **C** = 6 to those with **C** = 3.

Reducing the number of connections in the case of the BidAndAsk policy increases the number of steps to a solution by factors of three to five. Note that C = 3 systems cannot reach a solution in any reasonable time with expiration less than four, or excess less than two percent. The large effect of connectivity primarily results from the degree to which the BidAndAsk policy is hostage to difficuties of communicating by forwarding in a random net. A secondary factor is that the BidAndAsk policy is hampered by agents that have no incoming connections. With **C** = 3, a 100 agent system will have an average of five agents with no incoming connections. These will tend to be among the last to complete trading. With six connections there are seldom any such agents.

The Crossing policy is far less sensitive to connectivity (see Figure 7 and Table 5). With expiration of three, the less connected system takes as many as four times the steps to satisfy all goals but with longer expirations the increase is less than a factor of two. The CrossHistory policy is even less sensitive to connectivity than the Crossing policy. The pattern of sensitivity is much like that of the Crossing policy however the penalty for expirations greater than three is about 25 to 50 percent, and for expiration equal to three it is about a factor of two.

The CrossHistoryBroker policy is affected by connectivity even less than the CrossHistory policy. The penalty is about 22 percent averaged over all conditions (expirations from one to seven, excess from zero to four percent). In some conditions the penalty is nearly zero. The CrossHistoryBroker policy differs from the other policies in that its sensitivity to connectivity is essentially the same for all expirations, including expiration of one.

In summary, the more effective the policy is at knitting together an emergent decentralized market, the less important is the number of connections to allocating resources in the static goal task (see Table 5). The addition of crossing behavor to the basic BidAndAsk policy gives the the largest reduction in sensitivity to the number of connections. Crossing changes the system from one that is dependent upon blind random walks (where connectivity is all important) to one with emergent cooperation. The addition of history and brokering makes further improvements, but they are relatively small.

Policy	**Expiration = 4 % Excess = 2**	**Expiration = 6 % Excess = 4**
BidAndAsk	3.42	5.51
Crossing	1.82	1.32
CrossHistory	1.53	1.23
CrossHistoryBroker	1.03	1.22

Table 5: The entries are the ratios of mean steps to solution for systems with C = 3 to systems with C = 6.

2.3 Trading with dynamic goals

We next investigate a task in which changes in supply and demand take place more rapidly than a static “solution” can be reached. Specifically, on each step of the simulation one pair of agents exchanges a goal. That is, one agent chosen randomly from the agents with goals gives its goal to a randomly chosen agent that lacked a goal. The notion that an agent gains a goal for each agent that loses a goal is somewhat artificial, but it maintains a constant total demand in the system and therefore a constant relationship between supply and demand. What is important in this task is how quickly individual agents achieve their goals when the system is in the steady state equilibrium between the effect of the forced changes of goals and the efforts of the agents to adapt to those changes.

The assumption that goals may change at every time step appears, at least implicitly, in a number of studies [e.g., KS89, WHH92, SHC96, YWI96]. This task mimics the real-world situation in which agents individually require resources in order to act on behalf of external entities, e.g., allocation of network bandwidth on behalf of human users in a networked meeting environment [YWI96]. The asynchronously occurring demands that an external entity makes on an agent may require the agent to obtain certain resources. These demands may also asynchronously disappear (e.g., the entity is finished with its use of the agent), ending the agent's need for the resources.

A global solution is never reached in this task. The more effective a trading policy, the more quickly an agent can acquire a given commodity and hence the smaller the proportion of agents with unsatisfied goals. As with the static goal experiments, the system is initialized randomly and goals are assigned to half of the agents. Then the system is run long enough to reach steady state, i.e., for every agent to have had its goal reassigned at least once (200 steps is sufficient). When an agent receives a new goal, it counts the number of steps needed to obtain the target amount of each individual commodity. After 200 steps the average of these counts is recorded. The proportion of agents with one or more goals unsatisfied is also recorded. The data reported below are averages from five of these runs for each condition.

2.3.1 Comparison of policies

This task reveals a rather different aspect of the trading policies. With one important exception, the ordering of the effectiveness of the policies is the same as with the static goal task. The BidAndAsk policy is the poorest, the Crossing policy is better (27% fewer mean steps per goal), the CrossHistory policy is better still (33% further improvement). However the CrossHistoryBroker is almost as bad as the BidAndAsk (see Table 6).

Policy	Mean steps/goal	% Agents unsatisfied
CrossHistory	4.32	15.2
Crossing	6.42	23.2
CrossHistoryBroker	8.12	24.4
BidAndAsk	8.79	27.2

Table 6: 100 agents, zero excess supply, C=6, expiration=7, 2 goals changed per step. The percent unsatisfied agents refers to the percent of agents with goals that have not fully satisfied those goals.

The policies differ from one another less in this task than in the static goal task where most measures of effectiveness show the Crossing policy to be about twice as effective as the BidAndAsk policy and the CrossHistory policy twice as good again. This is to be expected because the data from the static goal task are about extreme values – in effect the slowest trade among hundreds. The data in this steady state task reflect central tendencies of the distribution of trading latencies.

The CrossHistory policy is especially effective when an agent's goal changes during trading. An agent that loses its goal typically will have collected most if not all of the commodity required by that goal and sold any excess it might have had. That is, it typically will have ceased active trading. But it still actively remembers all proposals that pass its way. On the first step after the goal is lost, the agent generates proposals (asks) to sell all of its resources. Thus it is likely that it has a memory of bids that can immediately be crossed with its newly generated asks. Similarly, an agent that has just received a new goal was not an active buyer so it is likely to have a memory of asks that can immediately be crossed with its newly generated bids. Thus the emergent market created by the distributed history of prior proposals can quickly redistribute newly freed resources and satisfy new demand.

The CrossHistoryBroker policy performs poorly in this task primarily because agents with goals are inefficient brokers. This is discussed in a later section.

2.3.2 Effect of rate of change

The rate at which goals are changed has a rather strong effect on mean steps to satisfy a goal. Doubling the number of agents whose goals are changed on each step from two percent to four percent reduces the time it takes to satisfy goals (see Figure 9 and compare with Figures 8). The improvement is larger when the task is more difficult, e.g., when demand exceeds supply by 20 percent, changing four goals instead of two per step reduces the mean number of steps from 11.23 to 6.31. And, as it turns out, the increase in unsatisfied agents due to the larger number of agents receiving new goals on each step is approximately balanced by the more rapid satisfaction of those goals.

The primary reason why higher rates of change improve performance is that, as discussed above, agents that lose their goal are very effective sellers especially with the CrossHistory policy. Higher rates of change create more of these effective sellers.

There is also a less important and less interesting reason that is an artifact of the way the data are gathered. If an agent loses its goal before it has satisfied the goal, it does not contribute to the data. Since this tends to preferentially censor data from cases which for whatever reason happen to take longer to satisfy goals, it tends to reduce the mean. At higher rates of change this censorship happens more often and thus has a greater effect on the mean. However this cannot account for much of the effect when the rate of change is four percent.

2.3.3 Effect of excess or deficit supply

The relationship between supply and demand affects this task quite differently from the static goal task. In this task, the system is not constrained to have at least as much supply as demand. In fact, performance changes smoothly from conditions with substantial excess demand to substantial excess supply (see Figure 8). For the CrossHistory policy, the mean number of steps to satisfy each goal falls gradually from 11.23 with demand exceeding supply by 20 percent to 2.16 steps with supply exceeding demand by 20 percent. Likewise the percent of agents with all three goals satisfied falls from 37.2 to 6.4 over the same range.

The maximum number of steps to satisfy a goal, i.e., the worst observed delay, is more sensitive to supply issues. With demand exceeding supply that maximum was 84 steps; with supply equal to demand it was 21, and with supply exceeding demand by 20 percent it falls to 11.

In part, the steady state task is less sensitive to supply issues because it does not require that all goals be met. Certainly that fact allows the system to function with demand exceeding supply. But a more important factor is that new supply becomes available every step. All the policies are relatively insensitive to supply issues in this task simply because, sooner or later, newly available resources allow a trade to happen. Moreover, as discussed above, the CrossHistory policy takes especially good advantage of new supply because any given agent typically participates in several mini-marts at once. In each of these it is competing on a first-come first-served basis for newly available resources.

2.3.4 Effect of scale

The number of agents in the system has no effect on the mean number of steps to satisfy a goal provided that the number of agents that have their goals changed each step is scaled proportionally. Consider, for example, the CrossHistory policy with **C** = 6, expiration of 7 and supply equal to demand. A 100 agent system with two goals changed per step takes a mean of 4.32 steps to satisfy a goal, a 225 agent system with four goals changed per step takes a mean of 4.70 steps, and a 400 agent system with eight goals changed per step takes a mean of 4.58 steps. We see similar insensitivity to scale at other supply/demand ratios and with other policies.

3 Discussion

As we have seen, decentralized trading by agents in a random network can allocate resources successfully, if slowly, with no more than the BidAndAsk policy. These agents foster communication by forwarding proposals belonging to others. Such forwarding weakly connects all agents into a community that could be considered an emergent market, even if it is a rather inefficient one.

The addition of mini-mart behavior to the basic BidAndAsk agents adds stronger connections, e.g., trades initiated by third parties and purchases on behalf of other agents, that give rise to a more effective emergent market. These additions improve the trading effectiveness of individual agents and the system as a whole by factors of two to seven depending on policy, task, condition, and measure of effectiveness. Mini-mart behavior also substantially reduces the scaling exponent in the one task that is sensitive to the number of agents in the system (the static goal task with supply equal to demand).

Crossing and CrossHistory policies allow agents to be effective mini-marts by better exploiting the information available from incoming proposals. During steady state trading in the dynamic goal task, 25 percent of the trades executed by Crossing policies are cross trades and 53 percent of those executed by CrossHistory policies are cross trades. During the later stages of the static goal task, when relatively few proposals are circulating and relatively few agents are still trading, the proportion of cross trades is essentially the same as in the dynamic goal task.

The benefits from crossing and history are not restricted to the tasks chosen here nor to random nets with random forwarding; they can work in other tasks, other topologies, and with other communication schemes. The benefits of brokering depend much more on the specifics of the simulation experiments.

3.1 Brokering

Brokering as it is implemented here has four somewhat separable effects. First, a broker bids "on behalf of" another agent and attempts to resell to that agent immediately upon a successful purchase. This, in effect, extends the reach of the original bidding agent. Second, if the attempt to resell fails (because the remembered bid expires or the need has been otherwise met) brokering has the effect of moving resources to different parts of the network that are, in a sense, closer to demand. This movement may be useful when connectivity between agents with excess supply and agents with demand happens to be poor. Third, brokers magnify demand, i.e., the total demand of bids circulating in the system is larger than needed to satisfy goals. Magnified demand may be valuable when supply is scarce and hard to find, but the additional bids from brokers compete with bids from agents with unmet goals. And fourth, a combination of factors (explained below) conspire to restrict the amount of resources that agents with satisfied goals can broker.

This last effect accounts for the poor performance of the CrossHistoryBroker policy in the dynamic goal task. We chose the three primary colors as the three market commodities in order to facilitate the visual observation of the progress of trading. Colors are represented by an amount between zero and one of each primary. We therefore assume that each agent can own a maximum of one unit of each primary. We also decided that half the agents would have goals and half would not. When half of the agents have goals and total supply is near total demand, the goal for each primary is near one, i.e., near the maximum an agent can own. These choices, in combination, mean that agents with satisfied goals can buy for resale no more than the small difference between their goal amount and the one unit maximum. Thus agents with goals tend to make small brokering purchases that disperse any large concentrations of color. Agents with unsatisfied goals may need several trades to reassemble the dispersed resources.

To investigate the impact of the dispersion problem, we changed the CrossHistoryBroker policy so that agents with goals did not attempt to broker. That is, only half the agents – those without goals – act as brokers. In the static goal task this change made small improvements in trading: a ten percent reduction in steps-to-solution with three percent excess supply and twenty percent reduction with supply equal to demand. However the dynamic goal task is much more sensitive to dispersion. These modifications double the effectiveness of the CrossHistoryBroker policy in the dynamic goal task. The revised brokering policy regains its position as the best trading policy by a slight margin over the CrossHistory policy (mean steps to solve a goal of 4.14 with 14.4 percent of agents unsatisfied).

The effects of buying on behalf of another agent can be separated from the effects of magnification of demand and of the movement of resources within the network by examining the performance of a random broker policy. This policy, which inherits from the CrossHistory policy, is called the CrossHistoryRandomBroker. Recall that when an agent has an opportunity to broker (i.e., to buy from an ask or generate a bid for a commodity that is not needed), the agent passes the decision to its policy. The CrossHistoryBroker policy answers yes if it remembers an unfilled bid. The CrossHistoryRandomBroker does not remember unfilled bids, it simply answers yes randomly. Thus resources are moved and demand is magnified, but the results do not directly benefit other bidding agents.

In the case of the static goal task, the random policy allows brokering with a probability of 0.25. The results are uniformly worse than those for the CrossHistoryBroker policy: the random policy typically requires 30 to 50 percent more steps. Moreover, except for conditions where expirations are three or less, the random policy is also worse than the CrossHistory policy from which it inherits most of its behavior. Thus in most conditions, random movement of resources and magnification of demand are deleterious. However, for expirations of three or less, where forwarding of proposals is not so effective, the random broker performs better than the CrossHistory policy.

We see similar results in the dynamic goal task (see Table 7). With all agents brokering randomly, the performance is worse than the BidAndAsk policy. Inhibiting agents with goals from brokering eliminates the dispersion problem and improves performance by about thirty percent. Reducing the probability of brokering provides further improvements because random brokering is deleterious in these conditions, i.e., where connectivity and forwarding are plentiful. In conditions where connectivity and forwarding are limiting factors, e.g., with **C** = 3 and expiration = 1, the CrossHistoryRandomBroker policy performs as well as the CrossHistoryBroker policy (both produce mean steps/goal of about 11.5). And in these conditions they are the only policies that work at all. Brokering works in these conditions because the movement of resources replaces the missing forwarding of proposals. That is, proposals never reach further than immediate neighbors, but resources are sold and resold by brokers until they are purchased by an agent that needs them to fulfill a goal.

RandomBroker	Mean steps/goal
25% of all requests	9.85
50% of goal-less requests	6.77
25% of goal-less requests	6.27
12.5% of goal-less requests	5.81

Table 7: 100 agents, zero excess supply, C=6, expiration=7, 2 goals changed per step. For the first row, all agents agree to broker with probability 0.25. In the remaining rows, the agents with goals do not broker. Thus the overall probability of brokering purchases is the same for the first two rows.

We can conclude that neither the movement of resources due to brokering nor the blind magnification of demand are, by themselves, beneficial except when forwarding or connectivity is poor. In general, the benefit from brokering clearly stems from the cases where the broker makes a purchase on behalf of another agent and successfully resells to that agent.

3.2 Price

In economic trading price plays two roles, neither of which translates easily into the realm of computer resource allocation. Over time, prices are important inputs to producers and consumers who must make decisions about how much to produce or consume, and prices act in the short term to influence choices about which parties participate in trades. The first of these is known in economics as Adam Smith's "invisible hand". As Yamaki, et al. [YWI96] characterizes it: "Under certain conditions, the market prices reflect system-wide values, inducing agents to produce and consume appropriate amounts of the various resources." That is, prices act as signals to producers and consumers to alter their production or consumption. Software agents are typically insensitive to such signals. Some researchers use abstract agents that, by definition, change their supply and demand according to price [e.g., Wel93]. Other researchers externally control the agent's demands for resources and their funding so that human decisions may adapt parameters of the agents as a result of price signals [e.g., WHH92]. However in most work that simulates the allocation of computing resources in a price driven market [e.g., FYN88, KI92], prices may be influenced by supply or demand but price signals do not affect overall supply or demand.

The second role that price plays in economic trading is to favor one agent's needs over those of another competing agent as trading takes place. The buyer that offers the highest price gets preference, as does the seller that asks the lowest price. In the arena of human economics, this seems straightforward. But hidden within this simple statement are implicit assumptions about how people set prices that are problematic in artificial economies. It assumes that agents can meaningfully value their needs in competition with other agents, as well as their need for one resource versus another resource. Agents would then use those estimates of value to decide whether to raise or lower their prices. In the realm of human economics, children take years to learn to make such tradeoffs. Until they learn, their typical behavior is to offer to pay whatever they have for whatever they want. Usually parents contribute a degree of prudence by severely limiting the funds available to the child. Software agents often behave much like the most naive child. We usually program them to pay whatever they have for whatever they need. That is the case in the present work and in the Spawn system [WHH92]. Software agents do not set prices in the sense that humans do;

their prices are controlled directly by the interaction between their needs and their available funds.

The notion of price also embodies assumptions about the economy as a whole. In the human economy there is a rich network of interactions and feedback loops through which money moves in a counter-flow to a wide range of goods and services. Prices emerge from a myriad of choices between goods and services that in some way are substitutable for one another. The size and intricacy of this web imparts a stability to the "value" of the currency that we take for granted except in rare times of hyperinflation. Thus prices are more a phenomenon of the whole web than of a specific commodity. Outside of this rich economic web, prices lose most if not all of their meaning. Picture a wealthy banker marooned on a desert island with a poor fisherman. Without the economic web within which the banker and the fisherman normally live, what is the price of a fish? Artificial computer resource markets are much like this desert island. There is a small number of resources (often just one) together with a currency. The resources usually cannot substitute for one another. We argue that the web of trades in such simple simulations is not rich enough to support more than an illusion of meaningful prices. Without a rich economy of alternatives and tradeoffs, price mediated trading is no less arbitrary than the first-come-first-served trading used in this study.

The above not withstanding, if the mini-mart principles investigated here are to be adopted for decentralized resource allocation in real world applications in the Internet, prices based on real money will need to play a role in trading. Not only will many of the resources that are traded have real costs (and be owned by users that insist on receiving real payments), but also price incentives may be needed to foster cooperative behavior, e.g., buyers and sellers might offer micro-payments to encourage forwarding and crossing.

3.3 Metalevel messages and adaptation

Although agents in our simulations used fixed parameters during a run, in more general usage they would adapt parameters to current circumstances. For example, if there is reason to believe that an agent's proposals are likely to be dropped before encountering a trade, the agent ought to reduce the expiration times. If, on the other hand, there is reason to believe that proposals are likely to expire before encountering a trade, the agent should lengthen the expiration times. Informal experiments indicated that adaptation of expiration time during trading did not improve system performance. This is in part because the measures of effectiveness we use are influenced primarily by performance in the later stages of a simulation run when few if any proposals are being dropped. Another reason appears to be simply that the systems we investigate are homogeneous. However, real world applications of the mini-mart approach to resource allocation will not occur in homogeneous systems such as we have studied here. Agents will have different goals, different degrees of connectivity to other agents, and may well use different policies from one another. Global as well as local supply and demand will change at both short and long time scales. Thus agents will need to adapt parameters of their policies, and conceivably the policies themselves, to changing conditions. To do so they will need metalevel information, i.e., information about the state of the market as a whole.

Real world central markets supply consolidated information about prices and trading volume for each commodity traded. Daily digests of this information for major financial markets are published in newspapers. Real time, or slightly delayed information is available electronically. Traders use such information in making their own trading decisions. Arbitrageurs (both human and digital) may use metalevel information from multiple financial markets to identify short term imbalances between these markets that can be profitably exploited.

The general problem of monitoring distributed systems, i.e., assessing the state of a system made up of a large number of asynchronous processes has received much study [cf. BM96]. Distributed problem-solving systems address a similar issue to help agents adapt to one another and thereby make a more coordinated effort [e.g., Par85, DLC85].

In the present work we introduce metalevel messages, called announcements, to address that need. An announcement is a message that encapsulates what is known by a given agent about the state of the system. Some of this knowledge is from the agent's own trades and some is obtained and consolidated from incoming announcements. This form of metalevel message generation is cooperative in that the result of each agent's consolidation is consolidated again by each of its directly connected agents. At each step of this repeated consolidation data from many more agents is represented in the result. This distributed consolidation makes information available from a wide sample of other agents without swamping the system with message traffic.

Distributed consolidation must cope with the fact that information from a given agent for a given tick may reach the consolidating agent via multiple paths. Bid and ask prices can be safely consolidated by propagating the minimum ask price and the maximum bid price since minimum and maximum functions are not affected by duplication. Other statistics, such as average trade price, volume, number of expired proposals, and number of dropped proposals, are susceptible to distortion by duplication. However in the random nets we use here, the number of distinct agents represented in the data grows much more rapidly for the first few steps into the past than does the number of agents contributing duplicate data. For that reason, and because recent data is most informative in any case, we limited announcements to three steps of past history. Distributed consolidation in other systems with different degrees of connectivity or different topologies will face different degrees of distortion.

4 Conclusions

We investigate resource allocation in fully decentralized markets of agents, each one of which acts as a small scale market, operating in a random network with sparce interconnection. We use tasks in which the challenge for the agents is to find other agents with which to trade. Thus the focus of this work has been on investigating what sort of central market behaviors ought to be distributed to a decentralized collection of agents in order to best foster an effective and efficient parallel search for matching proposals.

We show that the behaviors that are most valuable are those that collaboratively initiate third party trades. We investigate two tasks: one where all agents must reach a mutual solution and one where agents must react to constantly changing goals. In both tasks, crossing with history is very effective. Brokering, if implemented with careful attention to the details of the trading situation, can improve effectiveness further. Moreover, brokering can compensate somewhat in circumstances where forwarding of proposals is poor.

We also show that the more effective a decentralized trading policy is at fostering the emergence of a pseudo-global market, the less sensitive the system is to poor connectivity, lack of excess supply of resources, and increases in system size. In fact, the defining characteristic of an emergent decentralized market should be its ability to transcend these challenges to effective trading.

5 References

[BC91] Bak, P., and K. Chen. Self-organized criticality. *Scientific American*. pp. 46-53, Jan. 1991.

[BM96] Babaoglu, O. and K. Marzullo. Consistent global states of distributed systems: fundamental concepts and mechanisms. In 10th International Workshop on Distributed Algorithms, WDAG '96, Bologna, Italy, Oct. 1996. *Springer-Verlag Lecture Notes in Computer Science (LNCS)*, vol. 1151, 1996.

[CH93] Cheriton, D. R., and K. Harty. A market approach to operating system memory allocation. Stanford University Department of Computer Science, 1993.

[DLC85] Durfee, E. H., V. R. Lesser, and D. D. Corkill. Coherent cooperation among communicating problem solvers. In *Proc. 1985 Distributed Artificial Intelligence Workshop*, pp. 231-276, Dec. 1985

[DS83] Davis, R. and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, vol. 20, pp. 63-109, 1983.

[FL72] Farber, D. J., and K. C. Larson. The structure of the distributed computing system – Software. In Proc. Symp. On Comput.-Commun. Networks and Teletraffic, J. Fox, Ed. Brooklyn, Ny: Polytechnic Press, Polytechnic Inst. of Brooklyn, April 1972, p. 539-545.

[FFH73] Farber, D. J., J. Feldman, F. R. Heinrich, M. D. Hopwood, K. C. Larson, D. C. Loomis, and L. A. Rowe. The distributed computing system. IEEE COMPCON Spring, 1973, pp. 31-34.

[FNS95] Ferguson, D., C. Nikolaou, J. Sairamesh, and Y. Yemini, Economic Models for Allocating Resources in Computer Systems, Market based Control of Distributed Systems, Ed. Scott Clearwater, World Scientific Press, 1995.

[FYN88] Ferguson, D., Y. Yemini, and C. Nikilaou. Microeconomic algorithms for load balancing in distributed computer systems. In *Proc. IEEE Int. Conf. On Distributed Computer Systems,* pp. 491-499, 1988.

[For91] Forrest, S. Introduction to the proceedings of the ninth annual CNLS conference. In *Emergent computation: Self-organizing, collective, and cooperative phenomena in natural and artificial computing networks*. Stephanie Forrest, Ed. MIT Press/North-Holland. 1991.

[GFS92] Gagliano, R. A., M. D. Fraser, and M. E. Schaefer. The simulation of decentralized control: A hostless resource allocation model. *Simulation*, Vol. 58, No. 6, 398-408, June 1992.

[GHJV94] Gamma, E., R. Helm, R. Johnson & J. Vlissides. *Design Patterns*. Addison-Wesley, 1994.

[GS91] Gagliano, R. A. and M. E. Schaefer. Decentralized control of access to resources in a network: Interdependence of strategies in a challenge ring model. In M. D. Fraser (ed.) *Advances in Control Networks and Large Scale Parallel Distrobuted Processing Models*, Ablex Publishing Co. Chapter 5, 145-170, 1991.

[Hub88] Huberman, B. A., Ed. *The Ecology of Computation*. North-Holland, 1988.

[KHH89] Kephart, J. O., T. Hogg, and B. A. Huberman. Dynamics of computational ecosystems. Phys. Rev. A, vol. 40, pp. 404-421, 1989.

[KI92] Kuwabara, K. and T. Ishida. Equilibratory approach to distributed resource allocation: toward coordinated balancing. In *Lecture Notes in Artificial Intelligence 830 – Artificial Social Systems – 4th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW*. C. Castelfranchi and E. Werner, Eds. Pp. 133-146, 1992.

[KS89] Kurose, J. F. and R. Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Trans. On Computers* 38: 705-717, 1989.

[Mae85] Maekawa, M. A square root of n algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comp. Sys.*, 3(2):145-159, 1985.

[MD88] Miller, M. S., and K. E. Drexler. Markets and computation: Agoric open systems. In *The Ecology of Computation*, B. A. Huberman (ed.), Elsevier Science Publishers B. V., North-Holland, 1988.

[Par85] Parunak, H. Van Dyke. Manufacturing experience with the contract net. In *Proc. 1985 Distributed Artificial Intelligence Workshop*, pp. 67-91, Dec. 1985

[RZ94] Rosenschein, J. S., and G. Zlotkin. *Rules of encounter: Designing conventions for automated negotiation among couputers*. MIT Press, 1994.

[San93] Sandholm, T. An implementation of the contract net protocol based on marginal cost calculations. Proc. of the National Conference on Artificial Intelligence, Wash. DC, AAAI Press, 1993.

[SHC96] Steiglitz, K., M. L. Honig, and L. M. Cohen. A computational market model based on individual action. In Market-Based Control: A paradigm for distributed resource allocation, S. H. Clearwater (ed.), World Scientific, 1996.

[Smi80] Smith, R. G. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. On Computers*. C-29(12), 1104-1113, 1980.

[Sut68] Sutherland, I. E. A futures market in computer time. *Communications of the ACM,* Vol. 11, No. 6, 449-451, June 1968.

[VT97] Vitek, J. and C. Tschudin (Eds.). *Mobile Object Systems: Towards the programmable Internet.* Second International Workshop, MOS'96, Linz, Austria, July 8-9, 1996, selected presentations and invited papers. Springer-Verlag, Berlin, 1997.

[Weg97] Wegner, P. Why interaction is more powerful than algorithms. *Communications of the ACM,* Vol. 40, No. 5, May 1997.

[Wel93] Wellman, M. P. A market-oriented programming environment ant its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, vol 1, pp. 1-23, 1993.

[Wel95a] Wellman, M. P. The economic approach to artificial intelligence. In *ACM Computing Surveys Symposium on Artificial Intelligence*, 27(3), 1995.

[Wel95b] Wellman, M. P. A computational market model for distributed configuration design. Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AI EDAM), vol. 9, pp.125-133, 1995.

[Wel96] Wellman, M. P. Market-oriented programming: Some early lessons. In *Market-based control: A paradigm for distributed resource allocation.* S. Clearwater (Ed.), World Scientific, 1996.

[WHH92] Waldspurger, C. A., T. Hogg, B. A. Huberman, J. O. Kephart, & S. Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18, 103-117, 1992.

[YWI96] Yamaki, H., M. P. Wellman, T. Ishida. A market based approach to allocating QoS for Multimedia Applications. In ICMAS-96, pp. 385-392, 1996.

6 Figures

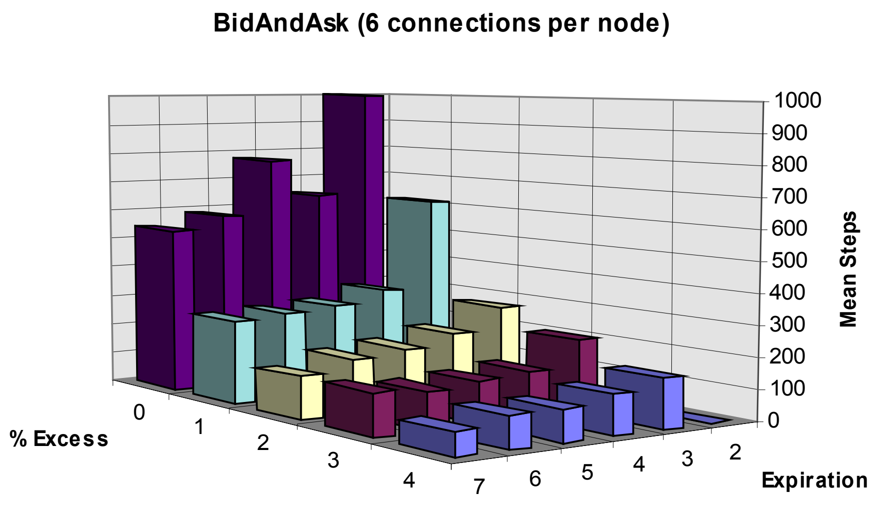

Figure 1: BidAndAsk policy, **N** = 100, **C** = 6. Mean steps to solution for 20 runs.

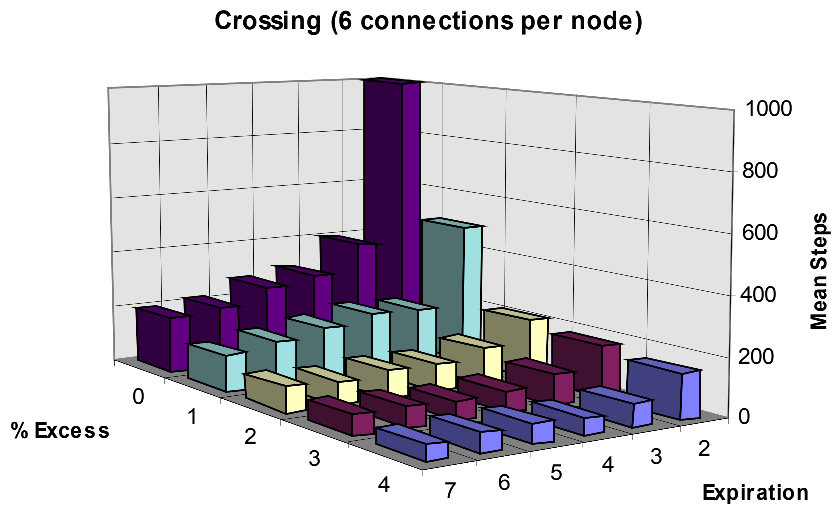

Figure 2: Crossing policy, **N** = 100, **C** = 6. Mean steps to solution for 20 runs.

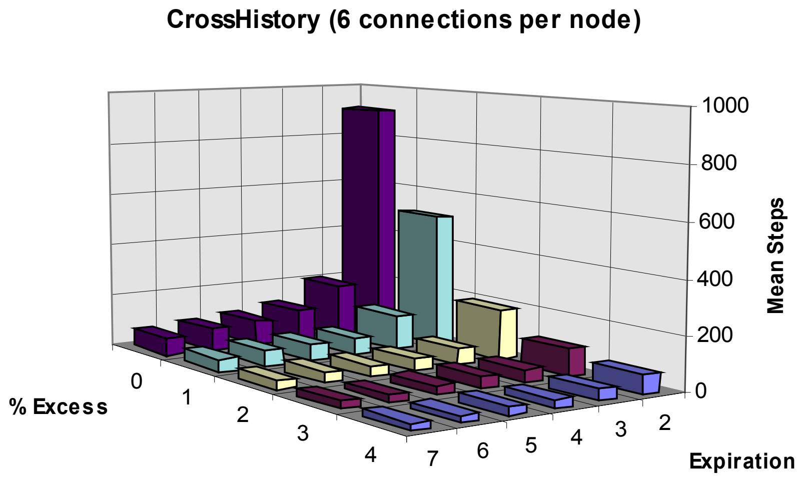

Figure 3: CrossHistory policy, **N** = 100, **C** = 6. Mean steps to solution for 20 runs.

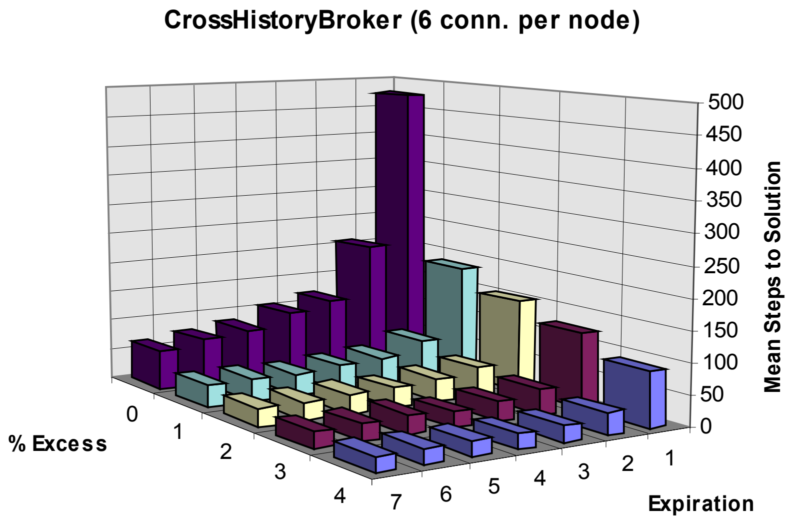

Figure 4: CrossHistoryBroker policy, **N** = 100, **C** = 6. Mean steps to solution for 20 runs.

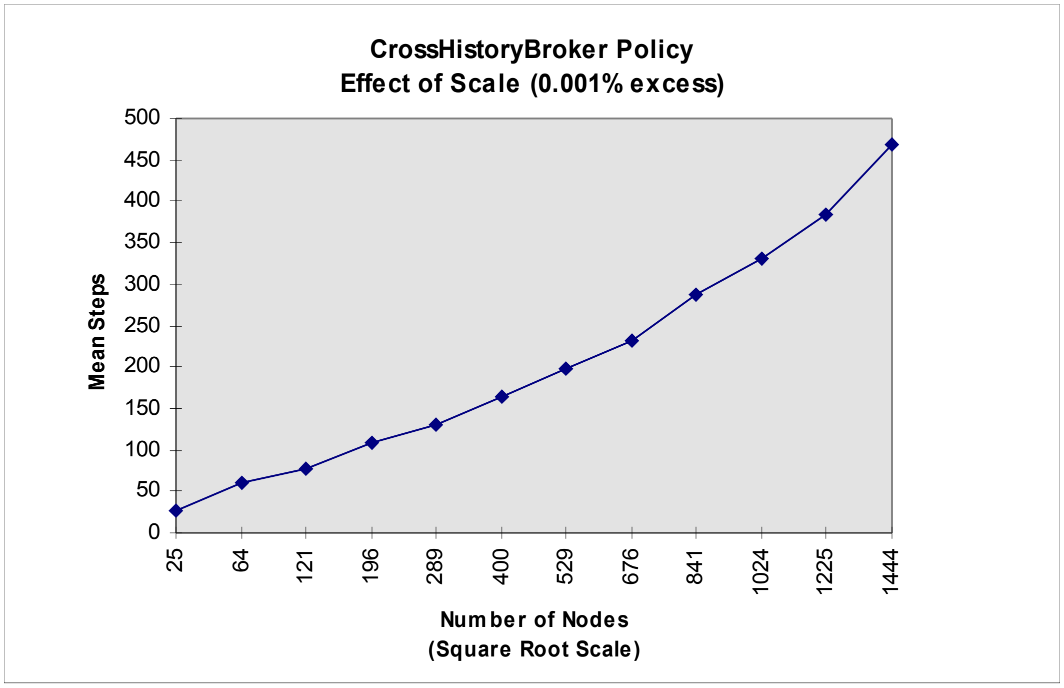

Figure 5: Effectiveness as a function of system size (6 connections, expiration of 7)

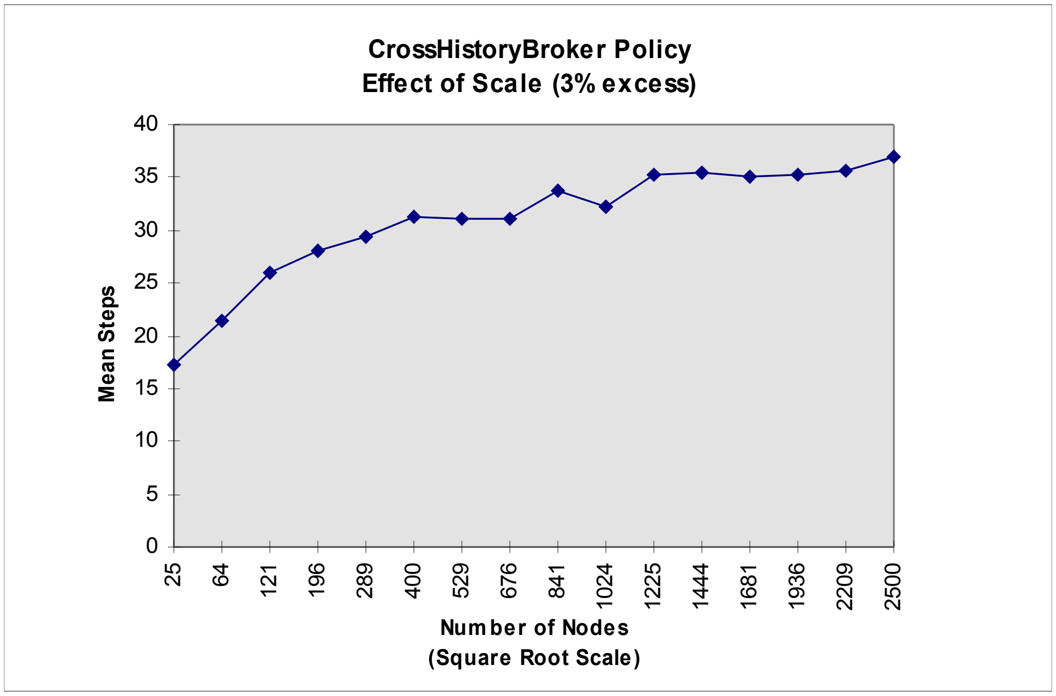

Figure 6: Effectiveness as a function of system size (6 connections, expiration of 7)

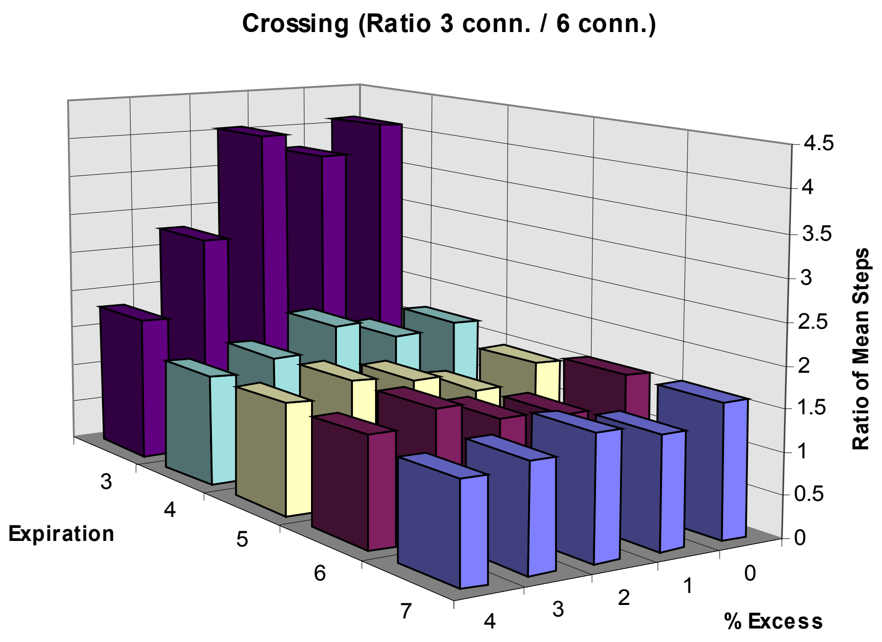

Figure 7: Effect of number of connections per agent

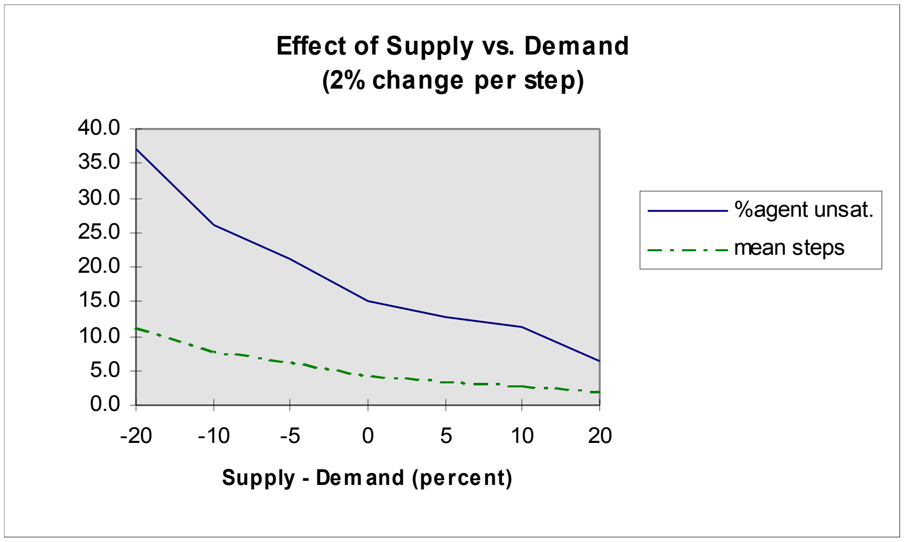

Figure 8: Effect of excess or deficit in supply. CrossHistory policy with C = 6, expiration = 7. The solid curve is the percent of those agents with goals for which the target amount of one or more commodities is not met. The dashed curve is the mean number of steps needed to satisfy a newly set target for one commodity.

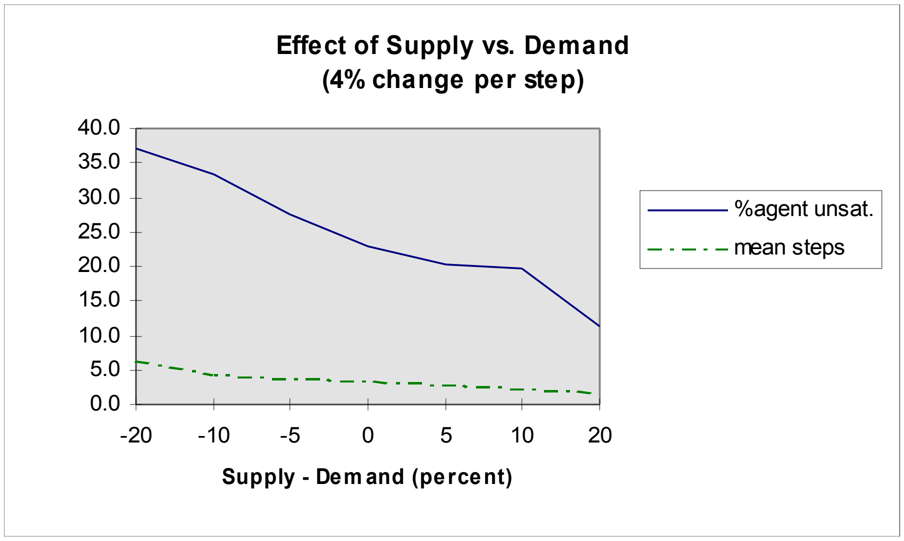

Figure 9: Effect of excess or deficit in supply. CrossHistory policy with C = 6, expiration = 7.