CHAPTER 4

MARKET-ORIENTED PROGRAMMING: SOME EARLY LESSONS*

MICHAEL P. WELLMAN
Dept of Electrical Engineering and Computer Science, University of Michigan
Ann Arbor, MI 48109, USA

1. Introduction

As the other chapters in this volume attest, the market-based approach offers a useful way to conceptualize and analyze distributed control problems, as well as to design and implement actual distributed control systems. For the past few years, I have been exploring this approach not only as a methodology for solving particular distributed control problems, but also as a generic *programming paradigm* for the development of distributed systems based on these methods. By a programming paradigm, I mean that the exercise of defining a computational market leads to the specification of a procedural solution to the underlying allocation problem facing that market. I call this approach "market-oriented programming".†

To support the practice of market-oriented programming, we need an infrastructure for building market-based systems—that is, a market-oriented programming *environment*. Such an environment would provide a set of generic constructs for specifying the elements of a computational economy, and implement some of the facilities to manage the interaction of these elements, according to defined protocols.

In this chapter, I provide an overview of our experience to date with market-oriented programming. After some further elucidation and motivation of the basic ideas, I describe the market-oriented programming environment we have been developing and using. The following sections outline some of the applications we have explored with the market-oriented approach. This leads up to a discussion of the lessons learned from our experience—some guiding principles and observations that were not obvious at the outset of this research (at least to us). We conclude by considering the relation to other research and some of the promises and pitfalls of market-oriented programming.

---

*Portions of this chapter are from a paper presented at the *Fifth International CGE Modeling Conference* Waterloo, Ontario, Canada, October 1994.

†The name, and the concept of layering constraints on top of object-oriented methodology, follows that of Shoham's "agent-oriented programming"[21].

1.1. General Equilibrium

Many of the market-based schemes that have been proposed for distributed resource allocation have focused on allocating a single resource, such as computation time, network bandwidth, or some other particular good. In proposing computational markets as a general programming paradigm, however, we must also address problems involving simultaneous allocation of multiple resources. Compared to the single-good case, the general problem is far more complicated. Instead of finding a single price for the scarce resource, we must now find a set of prices that balance supply and demand for all the goods. Because the goods are interrelated, the search space has dimensionality equal to the number of goods.

In economics, the concept of a set of interrelated goods in balance is called *general equilibrium*. Examining the market for a single good in isolation constitutes a *partial equilibrium*, which can be much simpler to compute and analyze, but can also neglect significant interactions. In market-oriented programming, we take *general* equilibrium to be the gold standard solution, but admit partial equilibrium as one way to trade solution quality for computational efficiency.[‡]

The connection between computation and general equilibrium is not all foreign to economists, who often appeal to the metaphor of market systems "computing" the activities of the agents involved. Some apply the concept more directly, employing *computable general-equilibrium* (CGE) models to analyze the effects of policy options on a given economic system[22]. Indeed, some of the methodological devices of CGE modeling—including particular agent models and good structures—are applicable to market-oriented programming. Others are not, as the economists are typically concerned with reasoning *about* a distributed system, and in so doing are free to perform a centralized analysis.

The view espoused here also resonates with Artificial Intelligence researchers' interpretation of modules in a distributed system as autonomous agents. In order to reasonably view a distributed system as a multiagent one, we must be able to attribute to the agents particular knowledge, preferences, and abilities, which in turn dictate their rational behavior. This rationality abstraction is shared by Artificial Intelligence and Economics, and is the common element that binds the complementary disciplines in this context.

In market-oriented programming we take the metaphor of an economy computing a multiagent behavior literally, and directly implement the distributed computation as a market price system. That is, the modules, or agents, interact by offering to buy or sell commodities at fixed unit prices. When this system reaches equilibrium, the computational market has indeed computed the allocation of resources throughout the system, and dictates the activities and consumptions of the various modules.

[‡]Since general equilibrium can be difficult to compute, especially in a distributed manner, we must admit many potential compromises. In any practical deployment of market-oriented programming, provision for behavior out of equilibrium seems a necessary component.

1.2. Motivation

There are several motivations for the market-oriented approach, and these are quite distinct from the motivations for CGE modeling in economic applications. With respect to the latter, general-equilibrium is a *descriptive* theory of the behavior of entities in a real-world economic system. Effects of alternate policy options are determined by rational reactions of maximizing agents playing according to a set of constraints and rules that are deemed to approximate the actual real-world system. In designing or programming a distributed system, the designer/programmer usually has quite a bit more freedom to determine the rules and constraints of the system, as well as the behavior and even the existence of the various entities[16]. We are thus adopting a market framework for *prescriptive* reasons, and these require some examination and justification.

In deciding to adopt a particular approach to distributed resource allocation, we are concerned with three classes of properties:

- *Results.* What is the quality of the allocation of resources produced by the system?
- *Computation.* How computationally intensive is the allocation process? How effectively is computation distributed?
- *Design.* How easy is it to design and specify a system for a given distributed allocation task? What analytical tools are available to predict the behavior of a system? How effectively can we engineer the system to achieve desired results?

The market-based approach offers potential advantages along each of these dimensions.[§] With respect to results, the attraction of the market framework is that it leads to Pareto-optimal allocations, under well-characterized conditions.[¶] Pareto optimality is clearly a desirable quality for any allocation mechanism, and adopting a market framework gives us one large class of sufficient conditions to achieve it. Moreover, the second welfare theorem tells us that *any* Pareto optimum is in principle achievable via the competitive mechanism. Thus, under the classical conditions, the market approach places a lower bound on the quality of solutions (Pareto efficiency), and no upper bound (any social optimum must be Pareto optimal).

The computational dimension is actually (at least) two dimensions: overall complexity and distributivity. It is not clear that the market approach offers any advantages in overall complexity, because in principle any computational savings could also

[§]Note that most of the advantages discussed are particular to the competitive mechanism, not simply the use of markets or economic principles. Other researchers have explored more general game-theoretic frameworks for distributed computing[16], but these do not necessarily enjoy the decentralization and analyzability properties of the competitive mechanism.

[¶]The so-called *classical conditions* are essentially that preferences and technologies are monotone, smooth, and convex.

be achieved by implementing the decomposition at a central source.[‖] It does, however, directly support distributivity, in two ways. First, it distributes decision making across individual agents, each with a limited scope of concerns. In the competitive mechanism particularly, each agent is concerned only with prices. When the classical conditions are satisfied and each agent is small with respect to the overall economy, these prices are sufficient summaries of the entire environment—the preferences, capabilities, and even existence of other agents is separable from the agent's decision problem.

Second, market price mechanisms support a natural decentralization with respect to commodities. If the bids are collected and clearing prices calculated for each good separately, then the market mechanism is itself distributed, with each auction or clearinghouse operating with limited dimensionality.

Perhaps the greatest potential advantages of the market approach are in the realm of design. These include:

- *Modularity.* The basic element of computation is an agent, which has a well-scoped set of capabilities (technology) or resources (endowment), and a well-defined and limited objective to maximize (profits or utility).

- *Regular interactions.* All interactions among agents are via trades at established prices, mediated by a uniform market mechanism.

- *Incrementality.* Under the assumptions of perfect competition, we can determine whether adding a *new* agent (or similarly, deleting an existing agent) will affect the behavior of the system, simply by inspecting the agent's profitability at the *current* equilibrium prices[19].

- *Analyzability.* The system as a whole is amenable to analysis using the full tool kit of general-equilibrium theory.

- *Incentive engineering.* The economic system can be modified to achieve desired results via a process (formal or informal) of incentive engineering.

It is in order to focus on design that I characterize the market approach to distributed computation as a "programming paradigm". I elaborate and comment on all of these points further in the discussion below.

2. The WALRAS System

[‖]Ultimately, arguments for computational benefits of distributed computation must rest on a cost of communication, else each module or agent could simply transmit its entire state to a central processor, which could then do no worse than replicating the distributed mechanism. Rather than present evidence for any particular cost measure, we simply note the increasing prevalence of distributed systems, and take distributivity as an exogenous constraint.

Our actual market-oriented programming environment, called WALRAS, is essentially an object-oriented implementation of general equilibrium theory. The object-oriented software-development methodology suits this problem well, as various types of agents are minor variants of other types.** General equilibrium theory's two basic classes of agents, consumers and producers, stand at the top of the hierarchy, with more specialized versions (e.g., those with particular forms of preferences or technologies) defined as subclasses. For example, a portion of the producer class taxonomy is depicted in Figure 1.

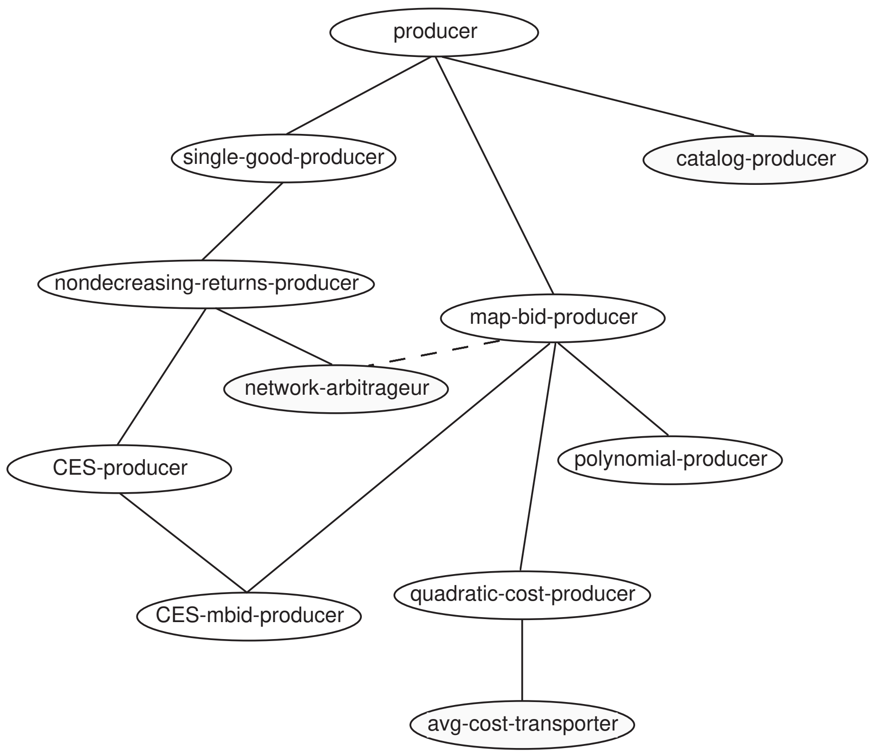

Figure 1: Portion of the WALRAS producer taxonomy.

Producer types differ on the form of their technologies (for example, single vs. multiple output, decreasing vs. nondecreasing returns, production functions with particular analytic forms), in their bid formats (point bids versus demand mappings), as well as other features. In the taxonomy shown, the shaded producer types are specific to particular application domains (the transportation and design economies, discussed briefly below). To implement a new agent type, the developer selects the existing type or types that already specify the bulk of the agent's computation (bookkeeping and communication functions, for example), then supplements this with specific methods

** Wang and Slagle[27] describe some of the advantages of object-oriented languages for representing general-equilibrium models. Unlike WALRAS, their proposed system is designed as an interface to conventional model-solving packages, rather than to support a decentralized computation of equilibrium directly.

particular to the new agent being defined.

Agents in WALRAS interact exclusively via a distributed bidding protocol. The consumer and producer agents bid so as to maximize utility and profits, respectively, subject to feasibility constraints and a competitiveness assumption. Each agent submits one bid for each good it has an interest in. Bids are demand functions, specifying the quantity demanded for any possible price of the good, under the assumption that the prices of the remaining goods are fixed at their current values. The underlying WALRAS infrastructure manages the distributed bidding protocol, passing along bids as they are generated asynchronously by the various agents. New clearing prices are computed as bids are revised, and the system notifies interested agents of relevant changes. Given that such an equilibrium exists, along with some additional assumptions (namely *gross substitutability*—the property that raising the price of one good will not decrease demand for another), this equilibrium is *stable*[2], and this price-adjustment process eventually converges to it.

3. Experience with MOP

To date, we have applied this approach to some simple problems in transportation planning and distributed engineering design. Full descriptions of these applications have been presented elsewhere; we include only summary reports below. Current work is developing more complex models in these domains, as well as investigating new applications in traffic management, allocation of computational resources, and provision of distributed information services. We are also performing research on the principles and engineering tradeoffs underlying market-oriented programming. This includes both theoretical and experimental analysis, as well as a distributed implementation currently under development.

We present summary descriptions of some applications below in order to convey a sense of the variety of tasks that can be tackled with the market-oriented approach. Based on our experience with these and other development efforts, we have distilled some general lessons about programming within this paradigm, presented in the succeeding section.

3.1. The transportation economy

The WALRAS transportation economy[28] implements a market-oriented approach to the distributed multicommodity flow problem. In the multicommodity flow problem, the task is to allocate a given set of cargo movements over a given transportation network so as to minimize cost. The transportation network is a collection of locations, with links (directed edges) identifying feasible transportation operations. Associated with each link is a specification of the cost of moving cargo along it, as a function of total quantity moved. We assume that the cargo is homogeneous, and that amounts of cargo are arbitrarily divisible. A movement requirement associates an amount of cargo with an origin-destination pair. A solution to the problem specifies the amount to transport on each link in order to satisfy all requirements at minimum total cost.

In the distributed version of the problem, we decentralize the decision making by allocating separate responsibility for each movement requirement.

In the WALRAS formulation of the problem, we include two types of goods.

1. Transportation on a link (i, j), representing an amount of cargo transported from i to j. There is one of these goods for each edge in the network.

2. Basic transportation resources. This can be interpreted as an amalgam of the factors of transportation: vehicles, fuel, labor, etc.

The consumers in the transportation economy are called *shippers*. Each shipper is associated with a movement requirement of the form:

Move x units of cargo from location i to location j.

The shipper's task is to find the minimum-cost way to satisfy this requirement. A price-taking shipper would simply calculate the shortest path in the network, where costs are the current prices of the transportation goods. The actual behavior of shippers in our implementation is more complicated, however.

Shippers are endowed with basic transportation resources, and use income from this endowment (in addition to profit shares of the producers) to purchase transportation goods, according to their path analysis. Note that the shippers are nonstandard consumers in that their problem cannot be cast as maximization of a utility function, strictly speaking. For that reason, the transportation economy is not really an instance of the general-equilibrium framework.

Carriers are agents of type producer who have the capability to transport cargo units over specified links, given varying amounts of transportation resources. We associate one carrier with each available link. The production function for each carrier is simply the inverse of the cost function for that link.

In the case of a decreasing returns technology, the producer's (carrier's) optimization problem has a unique solution. The optimal level of activity maximizes revenues minus costs, which occurs at the point where the output price equals marginal cost. Using this result, carriers submit supply bids specifying transportation services as a function of link prices (with resource price fixed), and demand bids specifying required resources as a function of input prices (for activity level computed with output price fixed).

A second class of producers, called *arbitrageurs*, act as specialized middlemen, monitoring isolated pieces of the network for inefficiencies. Each arbitrageur is associated with a triple of locations (i, j, k), and produces transportation from i to k by buying capacity from i to j and j to k. Its production function simply specifies that the amount of its output good, transport from i to k, is equal to the minimum of its two inputs, transport from i to j and j to k. If the price of the output exceeds the sum of prices of its inputs, then its production is profitable. Its bidding policy is to increment its level of activity at each iteration by an amount proportional to its current profitability (or decrement proportional to the loss).

To incorporate arbitrageurs into the transportation market structure, we first create new goods corresponding to the transitive closure of the transportation network. Next, we add an arbitrageur for every triple of locations (i, j, k) such that (1) $i \rightarrow j$ is in the original network, and (2) there exists a path from j to k that does not traverse location i. These two conditions ensure that there is an arbitrageur for every pair connected by a path with more than one link, and eliminate some combinations that are either redundant or clearly unprofitable.

We have run this economy in WALRAS under three different configurations. In model S (shippers only), the shippers pay proportional shares of the total cost on each link, and thus prices are set to average cost. Since average cost is below marginal cost (for a congested network), this gives shippers an incentive to overuse common links. The result is a suboptimal allocation, known in transportation science as the *user equilibrium*. This is an example of the classic *tragedy of the commons*.

In model SC (shippers and carriers), we "privatize" the network by creating a carrier agent for each link. Carriers price the transportation goods at marginal cost, which give the shippers the proper incentives. The result is a global optimum, or *system equilibrium*.

By introducing arbitrageurs, we decentralize the problem further. In model SCA (...and arbitrageurs), shippers may purchase directly a good corresponding to their origin-destination pair, so there is no need for path analysis. No agent is concerned with more than three goods. The result is the system equilibrium. The behavior of the SCA model is isomorphic to a standard algorithm for distributed multicommodity flow[10].

In ongoing and future work, we are extending the transportation economy in several ways:

- exchange of goods *at* a location, in addition to transport *between* locations
- production of goods integrated with transport
- explicit treatment of time of delivery, delays, and intertemporal allocation of resources

3.2. The Blue-Skies economy

Particular versions of the first two extensions listed above have been incorporated in the *Blue-Skies* economy, a simple model of information service provision on a network[15]. In this model, a popular information service[†] is available on the internet, and local agents decide whether to serve as mirror sites for the service.

The Blue-Skies economy modifies and extends the transportation economy in several respects. In addition to transport on a link and basic transportation resources (in this case, *effective network bandwidth*), the available goods include:

[†] Of which the canonical example is *Blue-Skies*, a weather-information server based at the University of Michigan[17], hence the name.

1. Information goods *at* a location. For example, *Blue-Skies*@local-site1, representing an amount of information service available at a particular node in the network. There is one of these goods for each (*service*, *node*) pair. Amount of the good is a combined measure of quantity and quality of service.

2. Basic computational resources. The various resources needed to provide caching service: storage, processing, plus another good that is an amalgam of the two.

Consumers in the Blue-Skies economy are end users of information services. As in the transport economy, the consumers trade generic resources (network and computational) for the good they care about—Blue-Skies at their local site. Unlike shippers in the transportation economy, the Blue-Skies consumers are not given an absolute quantity requirement, but rather have preferences for various combinations of resources and service levels. By specifying a positive preference for retaining some of their endowed resources, we allow that the consumers have some other (implicit) uses for the resource.

In addition to carriers and network arbitrageurs, the Blue-Skies economy includes three other producer types. *Manufacturers* produce particular computational resources (e.g., processing, storage) or information goods (Blue-Skies) from the generic (amalgamated) resource goods. *Delivery arbitrageurs* are similar to transportation network arbitrageurs, except they bundle information services at a location (e.g., Blue-Skies@internet) with transport from that location to another (e.g, local-site), to produce that service at the second location (Blue-Skies@local-site). Finally, the *mirror provider* has the capability of transforming local storage and other resources into provision of the information service at its local site. It can also choose to access the service from another site instead of mirroring, in which case it acts just like the delivery arbitrageur.

Depending on the initial configuration, the Blue-Skies economy exhibits a range of interesting behaviors. If there is sufficient demand at a local site, the mirror provider will set up a local cache. If there is not sufficient demand locally, the mirror site may still be profitable if the service can be resold to other sites.

For some values of the parameters, the behavior oscillates. When there are no mirror sites, there is a lot of network traffic and a relatively high price of bandwidth, and thus it appears profitable to set up a mirror site. But once the mirror is set up, traffic decreases as does the price of bandwidth, and so direct access to the internet appears to be cheaper. This phenomenon is a direct consequence of applying the competitiveness assumption when there are a small number of agents. With a larger number (bigger network, more diverse set of services available), the effect of one mirroring decision would have a negligible impact on overall traffic, and hence this oscillation would be ameliorated.

3.3. The design economy

The WALRAS design economy[29] solves problems in distributed configuration design. Roughly speaking, the configuration design problem[6,14] is to select a set of *parts* to perform a set of specified *functions*, maximizing some criteria subject to feasibility constraints. The set of choices is defined by a collection of *catalogs*, which list the parts available to perform each function and specifies their features.

The design economy distinguishes two classes of goods. First are basic resource attributes: resources that are required by the components in order to realize the desired performance, but are limited or costly or both. Generally, we desire to minimize our overall use of resources. The second class of goods are performance attributes, which measure the capabilities of the designed artifact, and which we typically desire to maximize.

Each part in a catalog is associated with a vector of resource and performance goods. The resources can be interpreted as input goods, and the performance goods as output. In this view, the *catalog producer* is an agent that transforms resources to performance. Thus, to specify a catalog producer's technology, we simply form a set of the attribute-value tuples characterizing each part. We then go through and negate the values for the resource goods, leaving the values for performance goods intact.

The consumer agent in a design economy is conceptually the end user or customer for the overall design. The consumer is endowed with the basic resource goods. The idea is that the consumer then (effectively) sells these resources to the various catalog agents, in exchange for performance goods. Its utility function reflects the desire to maximize performance as well as to minimize resource usage.

The consumer agent bids to maximize utility subject to its budget constraint, in the conventional manner. Catalog agents submit bids reflecting their choice of parts maximizing profits (or no part, if all are unprofitable at the going prices). We have run the design economy on several small problems, both real and contrived. In some cases, it produces an optimal or near-optimal design with a few iterations. In others, however, it fails to produce reasonable or even feasible designs at all. The problem is that nonconvexities inherent in the discreteness of the catalog (or real economies of scale) mean that there is often no competitive equilibrium of the system (or no smooth path for the progressive equilibration method to find one).

Continuing work is extending the design economy in several ways:

- Relax the competitiveness assumption for producers with increasing returns. That is, allow the producers to account for the influence of their own decisions on prices.

- Use the solution for a smoothed and convexified version of the problem as a bound or approximation to the actual solution.

- Replace price-taking bidding policy by quantity-taking behavior for catalog producers. (This in one plausible modification of strict competitive behavior.)

- Combine market- and constraint-based methods in a hybrid search scheme (e.g., integrate with the ACDS system[6]).

3.4. Other applications

In addition to these major tasks, we have also built a variety of other computational economies using WALRAS.

In developing the system and testing various algorithms, we have found it very useful to define *artificial economies*, based on simple parametrized agents (typically with CES—constant elasticity of substitution—preferences and technologies). The advantage of these economies is that we can test the equilibrium directly, and characterize the performance of the system in terms of generic parameters. Some of the lessons enumerated below are based on experiments with such artificial economies.

Doyle has been applying WALRAS to problems of allocating computational resources in the context of *distributed reason maintenance*[8]. In his reasoning economy[7], computational resources are allocated to competing planning and replanning functions according to their projected effectiveness and importance. As part of this effort, Bogan implemented a processor rental model using WALRAS for the allocation of limited computer processors among many competing tasks[4].

Finally, we are currently designing a market-oriented framework for allocation of resources within the University of Michigan Digital Library project[3]. In the digital library, complex search and retrieval tasks over a distributed body of collections will be performed collectively by a diverse and distributed set of mediator agents. These agents will be capable of a variety of information services, such search, query optimization, and information synthesis. We plan to apply market-oriented programming techniques to set up an economy of information services, with each mediator agent an "information entrepreneur", so as to allocate computational and other resources across the distributed system.

4. Lessons Learned

In the process of implementing the WALRAS system and developing the applications described above, we have learned quite a bit about the practice of market-oriented programming. The following list is our first attempt to enumerate these lessons. They are unfortunately not as crisp and definitive as we would like (and in fact should all be regarded as tentative), given that our experience is still quite limited. Nevertheless, this experience does strongly influence our current practice, and may prove useful for others interested in exploring this approach.

4.1. Get the goods right

Our first lesson is that the most important design parameter of a computational economy is the set of goods. Once the goods are defined, most everything else (i.e., the agents) follows quite straightforwardly. Since all agent activities and values must be defined in terms of the goods, goods can be viewed as the basic vocabulary of the system. As with all vocabularies, selecting an appropriate set of primitives is essential.

In the design economy, for example, the goods are the basic design attributes (resources and performance). Choice of a different set of goods (for example, based on parts) would have led to a completely different configuration of agents.

4.2. Avoid saturation of utility

One specific component of getting the goods right is making sure that there are no excess quantities of the goods without consumption or production value. This happens, for example, in the transportation economy, if shippers can meet their requirement without expending all of their endowment of basic resources. This excess seriously complicates the demand policy for shippers, because they are indifferent about their expenditure of endowment, yet they must earn enough in exchange to purchase their required transportation goods.

We avoid this problem in the Blue-Skies and design economies by assigning the consumers positive preference for retaining some of their endowments. The model presented by Steiglitz et al.[23] uses "gold" as a generic resource that isn't directly consumed, but yet is assigned positive utility. In all of these cases, the consumptive value for the resource is a proxy for other, unnamed, productive uses of the resource. Although it is preferable when possible to be explicit about the value of everything, as a last resort we can always avoid saturation by letting the agents consume the resource (i.e., "eat money"). Note that when there are multiple resource goods of this type, we need to make them close substitutes to avoid having an excess of one or the other.

4.3. Producers do all the work

The fundamental purpose of a market-oriented program is to compute *activities*—what resources should be used for in a distributed system. These activities are described primarily by the production behavior of the producers. To model a choice of actions, we define a producer whose technology consists of those actions.

The role of consumers in a computational economy is crucial, but limited. Consumers define basic values via their utility functions, and the relative importance of the different consumers' values is defined by endowments.

The roles of the two agent types is distinguished most clearly in the design economy, where the space of possible designs is represented as producers' technologies, and the design objectives represented by the consumer's utility function.

4.4. Privatization averts commons tragedies

Overuse of commonly owned resources is a frequent pitfall in distributed systems[20,25]. By assigning control of the common resources to individual agents, we can often internalize the externalities involved. This is precisely what happened when we added carriers to the transportation economy—the solution moved from a (suboptimal) user equilibrium to the (optimal) system equilibrium.

4.5. *Arbitrageurs increase decentralization*

In the transportation economy, introducing network arbitrageurs dramatically decentralized the problem, so that shippers no longer had to perform any path analysis on the network. Instead, goods were translated to other, essentially equivalent combinations of goods, by a chain of arbitrageurs. Arbitrageurs achieve this translation by effectively implementing *identities*. For example, if one of the identities in our system is that one left shoe plus one right shoe is a pair of shoes, then we could enforce that by introducing arbitrageurs with the *technology* of creating pairs of shoes from individual shoes (or individual shoes from pairs). Once a competitive, constant-returns, fixed-proportion producer is in the system, the price ratios are constrained to obey the relationship enforced by the identity.

More generally, the fact that all agents interested in a particular good must communicate with a central auction for that good may seem like a strong centralization constraint. With arbitrageurs, one can decentralize the communication arbitrarily by defining multiple versions of the same good. Agents interested in the good may communicate with the auction for any of its versions, whichever is most convenient. The versions are linked by arbitrageur agents who have the trivial technology to transform one of the versions to another. As long as all the versions are so connected, the equilibrium is unchanged.

4.6. *Interleave modeling and analysis*

To design a system to achieve a particular result, several analytical relationships are available. For example, we often have to choose endowments and utility parameters so that the resulting consumptions will be in a reasonable range. To determine these values, the basic facts of competitive analysis prove very useful. For example, the following facts often help us constrain the equilibrium prices.

- For competitive producers, price equals marginal cost.
- For competitive consumers, the price ratio between goods equals their marginal utility ratio.

To compute endowments, we can often obtain good bounds by considering rudimentary material balance constraints. In order to achieve a certain aggregate level of consumption, there must be adequate endowment of resources in the system. Often the technology description gives us a simple way to derive lower bounds on the amount of endowment necessary to produce a given output, and upper bounds on the amount of endowment sufficient.

We made extensive use of these relationships in our experiments with the Blue-Skies economy[15]. The underlying theory proved instrumental in allowing us to choose utility parameters and endowments that would lead to given levels of service provision to the consumers. We can often determine these values without computing an exact equilibrium, because most of the equations require only locally relative prices.

In performing the calculations, we found that even the simple parametrized utility functions had enough degrees of freedom to support the variety of solutions we wanted the system to reach.

Note that to apply any of this theory, we must insist on economic purity. Although market-inspired mechanisms may have some of the flavor of competitive systems, the exact analytical results typically depend on the assumptions of a purely competitive system.

4.7. Tentative computational lessons

In our computational experience with WALRAS, we have observed some regularities of behavior that were not obvious a priori. Our firm and precise theoretical results are limited to (eventual) convergence given strict assumptions. Our very tentative and imprecise impressionistic results are much broader.

The equilibration process scales well with the number of agents. The number of iterations required to reach equilibrium does not seem to rise with the agent population, and in fact seems to dip slightly. (If all the agents are running on a single machine, this means that the overall process scales linearly.) We attribute this to the fact that more agents means that each is less important, and the effects of an improper competitiveness assumption are ameliorated.

The equilibration process scales noncatastrophically with the number of goods. Increasing the number of goods does increase the time to convergence, but not as much as we had expected. Figure 2 plots the average number of iterations required to reach equilibrium (up to a fixed, very small, numeric tolerance) for a set of CES pure exchange economies, varying the number of goods from three to eleven. We have no good theory explaining the observed or expected relationships.

Asynchrony reduces oscillation. If all the agents bid on the same good at the same time, aggregate demand tends to overcompensate for the previous price. This observation is consistent with an analogous exploration of computational ecosystems[12].

Incremental bidding can simplify agent behavior. The basic WALRAS bidding process might be considered by some unreasonable in expecting agents to bid demand price-quantity correspondences, rather than simply point-price demands. We can relax this requirement by accumulating individual points in the demand curve over time, thus approximating the curve. This kind of scheme usually works, as long as the choice of which points to submit is careful to ensure that in the long run all relevant points are updated.

Complementarities are often not fatal. Strictly speaking, convergence of the equilibration algorithm depends on gross substitutability. In practice, we have run

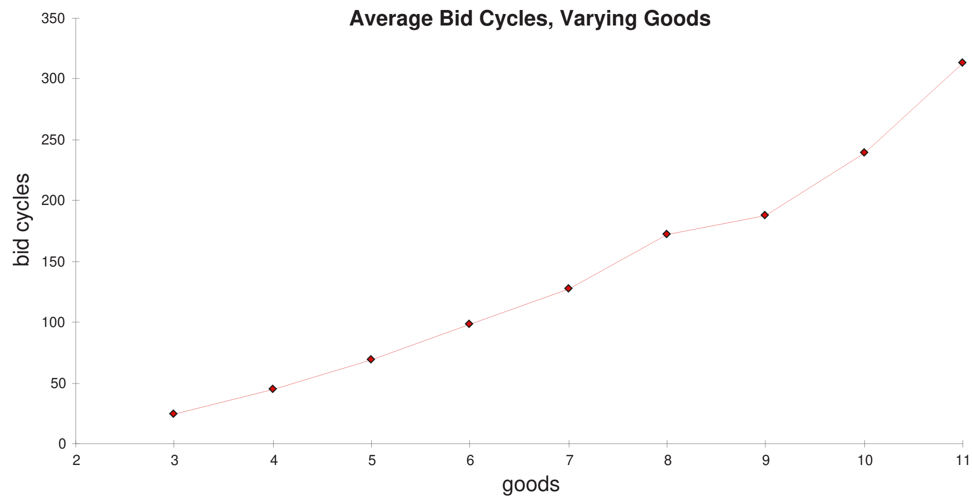

Figure 2: Cycles to convergence for randomly generated CES economies, as a function of the number of goods. This graph represents a total of 367 test runs, with the number of agents (consumers only) varying from three to twenty. (Since the cycles to convergence is insensitive to the number of agents, we aggregated the runs for each number of goods.) In each cycle, each agents submits an average of one bid to one auction.

many economies with complementarities, and typically get convergence anyway. For example, arbitrageurs almost always introduce extreme complementarities in production (the left-shoe, right-shoe example). It could be that production complementarities among the inputs are inherently less problematic, as there will always be a substitution relationship between the inputs and outputs (i.e., raised price of the output means more demand for the inputs). We are currently seeking a more precise characterization of what circumstances and what degree of complementarity can be tolerated.

Nonconvexities can be fatal, and often are. This was observed routinely in the design economy application. This observation is not surprising, but is included on the list so that one doesn't get the impression that everything is rosy.

We intend to present concrete data in partial support of these assertions as our investigations progress.

5. Conclusions

This chapter—and the entire volume for that matter—presents a representative sampling of computational markets solving distributed resource allocation problems. Investigations to date have focused largely on allocating computational resources[1,9,11,13,24,26], perhaps because these are most amenable to automation given that everything is already happening inside computers. But with the increased programmability of automatic control systems and deployment of decision support systems, it will not be surprising to find more applications to such problems as distributed climate control[5] or vehicle scheduling[18].

Although few of these other cited systems are based directly on the general-equilibrium framework (exceptions include some of those surveyed by Ferguson et al.[9]), we expect that results from these efforts will add to the lessons applicable to market-oriented programming in general. In our relatively short time pursuing the approach, we have indeed collected many methodological rules of thumb that usefully guide us in building models. Although these rules go beyond the strict economic theory, most of them can be rationalized in economic terms.

Continuing research aims to substantiate and elaborate these early lessons, and to expand the scope of the methodology by tackling large, realistic applications. Our particular emphasis right now is designing and implementing a computational economy of information services for the University of Michigan Digital Library project[3], mentioned earlier.

We expect that as our technical capacity to distribute computation increases, so will the need for principled allocation mechanisms. This coupled with the growing interest in autonomous software agents, and the emergence of electronic commerce on the internet, suggests that the time for a serious look at computational economies is at hand.

Acknowledgments

This work was supported in part by a grant from the University of Michigan Rackham School of Graduate Studies, and an NSF National Young Investigator award. Students at the University of Michigan—John Cheng, Vishal Grover, Philip Haar, Tracy Mullen, William Walsh, and Jeremy Wee—have contributed substantially to this project. Special thanks to John Cheng, Scott Clearwater, and Tracy Mullen for helpful comments on an earlier draft of this chapter.

Information on this project is available on the world-wide web at:

`http://ai.eecs.umich.edu/people/wellman/MOP.html`

6. *

References

[1] Agorics, Inc. An introduction to agoric computation. Technical Report ADd006P, Agorics, Inc., Los Altos, CA, 1994.

[2] Kenneth J. Arrow and Leonid Hurwicz, editors. *Studies in Resource Allocation Processes*. Cambridge University Press, Cambridge, 1977.

[3] William P. Birmingham, Karen M. Drabenstott, Carolyn O. Frost, Amy J. Warner, and Katherine Willis. The University of Michigan Digital Library: This

is not your father's library. In *Proceedings of Digital Libraries '94*, pages 53–60, June 1994.

[4] Nathaniel Rockwood Bogan. Economic allocation of computation time with computation markets. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May 1994.

[5] Scott H. Clearwater, Rick Costanza, Mike Dixon, and Brian Schroeder. Saving energy using market-based control. In this volume.

[6] Timothy P. Darr and William P. Birmingham. Automated design for concurrent engineering. *IEEE Expert*, 9(5):35–42, 1994.

[7] Jon Doyle. A reasoning economy for planning and replanning. In *Technical Papers of the ARPA Planning Initiative Workshop*, February 1994.

[8] Jon Doyle and Michael P. Wellman. Rational distributed reason maintenance for planning and replanning of large-scale activities (preliminary report). In *DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 28–36, San Diego, CA, November 1990.

[9] Donald F. Ferguson, Christos Nikolaou, Jakka Sairamesh, and Yechiam Yemini. Economic models for allocating resources in computer systems. In this volume.

[10] Robert G. Gallager. A minimum delay routing algorithm using distributed computation. *IEEE Transactions on Communications*, 25:73–85, 1977.

[11] Kieran Harty and David Cheriton. A market approach to operating system memory allocation. In this volume.

[12] J. O. Kephart, T. Hogg, and B. A. Huberman. Dynamics of computational ecosystems. *Physical Review A*, 40:404–421, 1989.

[13] James F. Kurose and Rahul Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Transactions on Computers*, 38:705–717, 1989.

[14] Sanjay Mittal and Felix Frayman. Towards a generic model of configuration tasks. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1395–1401, Detroit, MI, 1989. Morgan Kaufmann.

[15] Tracy Mullen and Michael P. Wellman. A simple computational market for network information services. Submitted for publication, 1995.

[16] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, 1994.

[17] Perry J. Samson, Ken Hay, and Jeffrey Ferguson. Blue-skies: Curriculum development for K-12 education. In *American Meterological Society Conference on Interactive Information and Processing Systems*, Nashville, TN, January 1994.

[18] Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the National Conference on Artificial Intelligence*, pages 256–262, Washington, DC, 1993. AAAI.

[19] Herbert E. Scarf. The allocation of resources in the presence of indivisibilities. *Journal of Economic Perspectives*, 8(4):111–128, 1994.

[20] Scott Shenker. Congestion control in computer networks: An exercise in cost-sharing. Prepared for delivery at Annual Meeting of the American Political Science Association, August 1991.

[21] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[22] John B. Shoven and John Whalley. *Applying General Equilibrium*. Cambridge University Press, 1992.

[23] Ken Steiglitz, Michael L. Honig, and Leonard M. Cohen. A computational market model based on individual action. In this volume.

[24] Michael Stonebraker, Robert Devine, Marcel Kornacker, Witold Litwin, Avi Pfeffer, Adam Sah, and Carl Staelin. An economic paradigm for query processing and data migration in Mariposa. In *Third International Conference on Parallel and Distributed Information Systems*, Las Vegas, NV, 1994.

[25] Roy M. Turner. The tragedy of the commons and distributed ai systems. In *Twelfth International Workshop on Distributed Artificial Intelligence*, pages 379–390, Hidden Valley, PA, 1993.

[26] Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffrey O. Kephart, and Scott Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18:103–117, 1992.

[27] Zhi Wang and James Slagle. An object-oriented knowledge-based approach for formulating applied general equilibrium models. In *Third International Workshop on Artificial Intelligence in Economics and Management*, Portland, OR, 1993.

[28] Michael P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–22, 1993.

[29] Michael P. Wellman. A computational market model for distributed configuration design. *AI EDAM*, to appear.