IVAN – Integrated Virtual Agent Network

by

Aleksandr Berkovich

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Engineering and Computer Science
and Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology
May 1, 2002

Copyright 2002 Aleksandr Berkovich. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author
Department of Electrical Engineering and Computer Science
May 1, 2002

Certified by
Paul Milgrom
Thesis Supervisor

Accepted by
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY
JUL 31 2002
LIBRARIES

BARKER
IVAN – Integrated Virtual Agent Network
by
Aleksandr Berkovich

Submitted to the
Department of Electrical Engineering and Computer Science

May 1, 2002

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Electrical Engineering and Computer Science
and Master of Engineering in Electrical Engineering and Computer Science

Abstract

This paper describes IVAN (Integrated Virtual Agent Network), a project that proposes to improve online marketplaces, specifically those that involve the trading of non-tangible goods and services. IVAN is an intermediary architecture intended as a generalized platform for the specification and brokering of heterogeneous goods and services. IVAN makes it possible for both buyers and sellers alike to more holistically and comprehensively specify relative preferences for the transaction partner, as well as for the attributes of the product in question, making price just one of a multitude of possible factors influencing the transaction decision. The ability to make such specifications results in a more efficient, richer, and integrative transaction experience. The modular object-oriented design of the system allows for easy addition of new functionalities to the system. A special handling of the “core” concept ensures the competitiveness of IVAN marketplaces in the free-market environment.

Thesis Supervisor: Paul Milgrom
Title: Visiting Professor, MIT Department of Economics
Table of Contents

Nomenclature and concepts ................................................................. 5
Acknowledgements ................................................................................ 6
1. Introduction .................................................................................... 7
   1.1. Current online marketplaces and their limitations ......................... 7
   1.2. The goals of IVAN ................................................................. 7
   1.3. Related work ....................................................................... 8
2. Design ............................................................................................ 11
   2.1. Automated agent creation ....................................................... 11
   2.2. The matching mechanism ....................................................... 19
   2.3. Dynamics of the system ......................................................... 26
3. Implementation ................................................................................. 27
   3.1. Market choice ...................................................................... 27
   3.2. Implementation notes ............................................................ 28
   3.3. The program ....................................................................... 29
   3.4. Least Squares .................................................................... 31
   3.5. Quadratic programming ......................................................... 32
   3.6. The core ........................................................................... 33
   3.7. The weighted matching problem ............................................ 34
   3.8. Materials and support ............................................................ 38
4. Experimentations ............................................................................. 40
   4.1. Problems with obtaining real life data ....................................... 40
   4.2. An example of using the program ............................................ 40
5. Discussion ...................................................................................... 44
   5.1. Least squares vs. adding the functions ...................................... 44
   5.2. Least squares vs. interpolation ............................................... 46
   5.3. Learning from the transaction history ...................................... 49
   5.4. How to ask questions ............................................................. 52
   5.5. How reliable are users' responses? .......................................... 54
   5.6. Choosing the best point between a buyer and a seller ................ 55
   5.7. The negotiation process ......................................................... 58
   5.8. The core ........................................................................... 59
   5.9. Continuous vs. discrete matching ............................................ 63
6. Recommendations ........................................................................... 65
   6.1. Aggregate matching .............................................................. 65
   6.2. The core ........................................................................... 66
   6.3. Market chains .................................................................... 68
Bibliography ......................................................................................... 70
List of Tables and Figures

Figure 1. Basic Utility Functions ................................................................. 12
Figure 2. A three-dimensional example of points asked .................................. 15
Figure 3. A three-dimensional example of approximation with the method of least squares .. 18
Figure 4. Matching between $m$ buyers and $n$ sellers ........................................ 22
Figure 5. Price formation ........................................................................ 25
Figure 6. Modular Dependency Diagram ...................................................... 29
Figure 7. A directional graph without weights ................................................. 36
Table 8. User specifications ........................................................................ 41
Table 9. Negotiation interval calculations .......................................................... 42
Figure 10. Matching .................................................................................. 43
Figure 11. Results of the polynomial interpolation ............................................ 47
Figure 12. Results of the polynomial least squares fitting ............................... 48
Figure 13. The deficiency of the current IVAN method ...................................... 51
Figure 14. The best point ........................................................................ 57
Figure 15. The baseball card market .............................................................. 60
Figure 16. The payoff difference .................................................................... 62
Figure 17. Two car producers ........................................................................ 67
Figure 18. A computer market chain ............................................................... 69
Nomenclature and Concepts

**Core** – a concept in the Game Theory. The core is a solution concept for coalitional games that requires that no set of players be able to break away and take a joint action that makes all of them better off.

**Least squares** – a curve-fitting numerical method. Given a set of design points $X$, a set of observation points $Y$, and a function $f$, this method defines all the term coefficients for the function $f$, such that the error $\|f(X) - c - Y\|_2$ is minimized.

**Linear programming** – a type of problems of the following form:

\[
\min cx,
\]

such that $Ax - b \geq 0$

and $x \geq 0$

**Nonlinear programming** – a type of problems of the following form:

\[
\min f(x),
\]

such that $g(x) \geq 0$

and $x \geq 0$
Acknowledgments

Several classes taught at MIT were very helpful for understanding the concepts and algorithms for this thesis, namely 6.046 Introduction to Algorithms, 18.327 Wavelets and Filterbanks, 6.252 Nonlinear programming, and especially 18.335 Numerical Methods.

Special thanks to Vladislav Gabovich who helped me with the system design and with the most ambiguous issues.

I would also like to thank Paul Milgrom for explaining the concept of the core and for all his suggestions; Eva Meyersson, who helped me to find a thesis advisor; and Andrei Bremzen for his help in clarifying some economical concepts.
Chapter 1. Introduction

1.1. Current online marketplaces and their limitations

Currently available online marketplaces, such as Chemdex [20], Priceline [21], Elance [22], etc. rely on price as the sole deciding factor in determining the final end product. This greatly impedes a user's ability to fully specify his desires. While in Priceline a user can specify and vary many attributes, such as date, number of stops, etc., the final bidding price does not vary according to those attributes. A user must manually specify different prices for different dates, if it is important to him. This deficiency results in a static bidding, where neither buyer nor seller can dynamically specify the importance of quantity or any other attribute of the product in price formation.

In addition, some of the systems, such as Ebay [12] and Amazon Auctions [13], have a tendency to foster a spirit of seller-biased adversarial competitiveness that ultimately reduces the overall welfare of all parties involved in the marketplace. In these systems, not only must a buyer first undergo the burden of uniquely identifying the exact product he is seeking, but furthermore, he must then enter into an inflexible and antagonistic bidding interplay with the seller.

1.2. The goals of IVAN

IVAN attempts to overcome the limitations of the present-day online marketplaces. The system makes it possible for both buyers and sellers to specify more comprehensively relative preferences for the transaction partner, as well as for the attributes of the product in question. Thus, price is just one of a multitude of possible
factors influencing the choice of trading partner and the decision to trade. IVAN allows both the buyer and the seller to exercise control. By allowing each party to choose and implicitly associate weights with relevant features from the underlying ontology, IVAN makes it possible to take into account the subtle differences in the characteristics of each party, so as to facilitate a more accurate match.

Using an automated user representation in the system, IVAN makes it possible to move away from the conventional bidding wars and, instead, maximize the welfare of all parties involved at the system level.

IVAN makes extensive use of transaction history to help new users determine their preference models.

IVAN pays special attention to the concepts developed in the Game Theory [29]. Specifically, it addresses the issue of competitive environment and ensures the long-term stability and success of the market. The system automatically regulates market negotiations, to ensure that all market transactions fall into the core subspace of possible solution space.

Finally, IVAN attempts to optimize marketplace transactions by creating chains of related transactions and, ultimately, suggests unified market solutions for deals involving multiple users who provide a range of products - from basic raw materials to complex end-services.

1.3. Related work

Unlike most online shopping systems, which generally operate in only one stage of the online shopping process [1], IVAN operates in three core stages – namely product
brokering, merchant brokering, and negotiation – to provide a unified experience that better facilitates economically efficient and socially desirable transactions. IVAN unites features of the Market Maker [8] and Tête-à-Tête [9] projects at the Media Lab, and extends these to create a more comprehensive solution. In particular, IVAN builds upon multi-attribute utility theory formulations, as introduced in Tête-à-Tête, to model relative user preferences and to quantify tradeoffs.

IVAN relates to first generation price-comparison systems such as BargainFinder [4] and Jango [5], but goes much further than the rudimentary functionality afforded by such tools. IVAN goes beyond just bid and ask prices to include the attributes of the transaction parties as dimensions for consideration and differentiation.

IVAN relates to second-generation value comparison-shopping systems such as Personalogic [2], MySimon [10], and the Frictionless ValueShopper [3] in that it offers an advanced decision support engine, based on multi-attribute utility theory, which meaningfully facilitates the exchange of complex and heterogeneous products. It differs from these systems in that it (i) allows both parties (buyers and sellers) to search for an optimal transaction partner and (ii) it automates the matching process between buyers and sellers. Furthermore, IVAN supports a non-linear and iterative user-interaction model that more accurately reflects the true nature of real-life transactions.

IVAN relates to online negotiation systems and auctions, such as Kasbah [6] and AuctionBot [7], and commercial systems provided by Moai [23], TradingDynamics [24] and others. IVAN is unique in the it proposes an integrative negotiation protocol and interaction model. This model, based upon bilateral argumentation, embodies an
appropriate blend of formality and efficiency, and provides an alternative to the adversarial competitiveness of online auctions.

Additionally, IVAN relates to work in operations research done in the domain of dynamic pricing of inventories [14, 15]. Specifically, it addresses the issue of how sellers should dynamically shift their valuations when demand is price-sensitive and stochastic, and the seller's objective is to maximize expected revenues. Moreover, the algorithms for matching buyers and sellers are fundamentally based on flow algorithms as encountered in combinatorial optimization and network theory [16].

Finally, IVAN builds on work done in the area of market-oriented allocation mechanisms [17, 18, 19]. It builds upon economic theory, to formulate the problem in economics terminology [11] with optimization heuristics, such as maximization of aggregate surplus, that derive directly from the literature.
Chapter 2. Design

Since IVAN is designed to capture all the stages of the transaction process, it is divided into three major parts: the creation of automated agents representing market users, the buyer/selling matching mechanism, and finally, the buyer-seller negotiation process.

2.1. Automated agent creation

An agent represents each distinct buyer or seller within IVAN. The "buyer agent" embodies the buyer's preferences with respect to the desired resource. Similarly, the "seller agent" embodies the preferences and interests of the seller. The system uses three steps to fit each agent as closely as possible to its owner's needs on the market. Each step will now be discussed.

Step 1. Choosing a correct profile

In order to serve the various types of buyers and sellers, each market comes with a set of profiles. Each profile represents a distinctive group of users (e.g., aggressive buyers, students, millionaires, etc). Of course, profiles introduce a big limitation to the system by offering users a limited amount of choices. Those profiles, however, are very tunable. Moreover, new profiles can easily be added by an administrator, as well as by a knowledgeable user.
Internally, each profile is a set of $n$ 1-dimensional functions, where $n$ corresponds to the number of attributes in the system. Each function answers the following question: "Keeping the other attributes constant, how would a user’s price preferences react to the change of a particular attribute?" If the function is constant, for example, the user is indifferent to the quantity of a particular attribute in the final product. If the function is quadratic, the user’s desired final price is very sensitive to a particular attribute. Figure 1 shows currently supported functions that can be used to form a profile.

<table>
<thead>
<tr>
<th>Name</th>
<th>Functional Generalization</th>
<th>Graph Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>UF$_1$</td>
<td>linear, positive slope</td>
<td><img src="image1" alt="Graph" /></td>
</tr>
<tr>
<td>UF$_2$</td>
<td>quadratic</td>
<td><img src="image2" alt="Graph" /></td>
</tr>
<tr>
<td>UF$_3$</td>
<td>quadratic</td>
<td><img src="image3" alt="Graph" /></td>
</tr>
<tr>
<td>UF$_4$</td>
<td>linear, negative slope</td>
<td><img src="image4" alt="Graph" /></td>
</tr>
<tr>
<td>UF$_5$</td>
<td>quadratic</td>
<td><img src="image5" alt="Graph" /></td>
</tr>
<tr>
<td>UF$_6$</td>
<td>quadratic</td>
<td><img src="image6" alt="Graph" /></td>
</tr>
<tr>
<td>UF$_7$</td>
<td>step function</td>
<td><img src="image7" alt="Graph" /></td>
</tr>
<tr>
<td>UF$_8$</td>
<td>- step function</td>
<td><img src="image8" alt="Graph" /></td>
</tr>
<tr>
<td>UF$_9$</td>
<td>constant</td>
<td><img src="image9" alt="Graph" /></td>
</tr>
<tr>
<td>UF$_{10}$</td>
<td>impulse function</td>
<td><img src="image10" alt="Graph" /></td>
</tr>
<tr>
<td>UF$_{11}$</td>
<td>- impulse function</td>
<td><img src="image11" alt="Graph" /></td>
</tr>
</tbody>
</table>

Figure 1: Basic Utility Functions
This figure shows a list of utility functions currently supported by the system. These functions are used to create a
user profile. They estimate a user’s willingness to pay for a quantity of a particular attribute.

Notice that all functions, except for the last two, are monotonic. This assumption comes from the fact that if an attribute can be quantified (i.e., a certain quantity of this attribute can be added to the final product), it will usually follow the "more is better" axiom of economics (with a plus or minus sign). The last two cases are reserved for special non-quantative attributes, such as color, and are treated in a special way.

As a user chooses a profile, his chosen utility functions are listed to him. He may change any of these functions and, therefore, redefine his profile. However, this functionality is designed for advanced users and will most often be unnecessary.

**Step 2. The Q&A session**

In this step, the system extracts information about a user's utility function (chosen in the previous step) by prompting the user to answer a set of questions.

First, IVAN finds the *negotiating subspace* specific to the user. In order to do this, the system asks the user for his optimal deal parameters in the $n$-dimensional space (i.e., the most desired values of product attributes) and his reservation price for it. The rest of the paper refers to this user-specific point as “the central value” or “the central product.” Here, the buyer/seller can conceivably enter 0/infinity, but, in this case, clearly no matching party will be available later. Therefore the buyer/seller is always interested in entering the maximum/minimum price for which he could potentially buy/sell the product. Next, the user is asked to define the boundaries (floor/ceiling) for each attribute. All offers outside of these boundaries will automatically be rejected.
Second, the system figures out a user’s attribute-specific 1-dimensional utility functions. It defines the coefficients for the functions chosen during the profile creation. By choosing certain functions in the profile, a user only specified a general utility curve for each attribute. The coefficients of the functions are still undefined.

In this part of the Q&A session, IVAN asks the minimum amount of questions that would define these coefficients. The system offers a user to evaluate several products and asks him to provide reservation prices for them.

As described above, each attribute is associated with a 1-dimensional function, whose coefficients are unknown. Three points are enough, for example, to determine the coefficients of a quadratic function. Thus, it is sufficient for a user to evaluate three different products to define this function exactly. Since the products span an n-dimensional space, only the dimension that represents the attribute in question has to vary, while the other dimensions have to be kept constant. Since the user already gave his price for the “central” product, this task becomes very easy.
Figure 2: A three dimensional example of points asked

Here, a system asks for a Central Point and two extra points about attribute $x$ (as it is quadratic and needs at least 3 points) and one extra point about attribute $y$. The questions asked concentrate around the central point. The distance between points is 10% of the size of the total range.

For each attribute, IVAN varies its value, while keeping the rest of the attributes at their central values (see figure 2). To make it as simple and clear as possible, the system asks several questions resembling the following: “...and how much are you willing to pay, if the Seller_Reputation is increased by 10% (to 7)?” Such questions easily relate to the “central product” price given by the user in the beginning of the Q&A session and, therefore, can be answered by a user with minimal effort.
**Step 3. Modeling an n-dimensional user Utility Function**

The next step is to obtain a general n-dimensional utility function from the current n 1-dimensional functions. IVAN utilizes the most commonly used tool for this type of problems – the method of least squares, or, as it is also called, the regression. *Least squares* is a curve-fitting numerical method. Given a set of design points \( X \), a set of observation points \( Y \), and a function \( f \), this method defines all the term coefficients for the function \( f \), such that the error \( r = \| f(X) - c - Y \|_2 \) is minimized.

In order to extract design points from available continuous 1-dimensional functions, the system first discretizes them and then retrieves some predefined number \( k \) of uniformly distributed points from each function. Recall that while one dimension varies, the rest are set at a user-defined central value. Therefore, each point retrieved from a 1-dimensional function has a unique equivalent in n-dimensional space. This way, with \( n \) attributes IVAN obtains \( n \cdot k \) points spanning like a star from the central value.

Next, the system looks up in its database the history of all previous transactions of users with the same profile and prompts a user to choose one of the following four options:

- Continue with the "idealistic" functions created and disregard all user history.
- Continue with the "idealistic" functions, but answer more questions.
- Add the user's previous final transaction points to his utility functions, if the user had a previous history with the chosen profile.
- Add to the user profile all the historical final transaction points obtained by choosing the current profile (all users).
The last three options attempt to define more accurately the overall $n$-dimensional utility function of the user, and, therefore, increase the accuracy of the automated agent that represents the user.

If the user chooses any of the two “history” options, IVAN retrieves the corresponding information from the database and adds new points to the ones already obtained through 1-dimensional functions. Otherwise, i.e. in case of the first two options, the system goes on to the next operation, the least squares calculation.

Now, as all the points have been collected, the system is ready to perform a least square regression to determine the $n$-dimensional utility function of the user. The values of attributes serve as the set of design points $X$, while the corresponding prices form the observation set $Y$. The sum of the profile functions serves as the base function $f$ for the least squares approximation.
Figure 3: A 3-dimensional example of approximation with a least squares method
In this example, a function of two variables was obtained from two simple original attribute functions \( f(x) = 0.25 \cdot x^2 + 1 \) and \( f(y) = 2.5 \cdot y \)

As the system calculated an n-dimensional user utility function, it checks whether a user has chosen to continue answering questions in the last menu (option two). If he did, IVAN will choose several points lying outside of the original attribute functions that are close to the user negotiation subspace border. Then the system will present these points to the user for evaluation with the prices suggested by the function just calculated for the user. As the user corrects these points, the system again recalculates the least squares using both "new" and "old" points and the same base function \( f \). (Note: the functionality described in this paragraph has not been added yet to the system)
Finally, the $n$-dimensional Utility Function obtained for the user serves as his agent on the market. Given any product description, it can output user’s reservation price or profits from possible transactions.

Discussion sections 5.1 and 5.2 explore in more detail the reasoning behind choosing least squares as a method for retrieving user’s agent function.

2.2. The matching mechanism

As the buyers’ and sellers’ respective agents enter the system, they are getting matched according to the following procedure:

*Step 1. Identifying the optimal point (deal) for each (buyer, seller) pair in the system*

For each buyer and seller that requested to be matched in the same market, the system needs to determine the product that maximizes the welfare of both parties involved. A product translates into a specific point in $n$-dimensional space spanned by the set of market-related attributes. Due to the specifics of the problem involved (see the discussion section 5.6 below for more info), IVAN chooses this point to be the solution of the following minimization problem:

$$
\min f(x) = ||x - b_{cv}|| + ||x - s_{cv}|| \quad //\text{sum of distances to the central values such that}
\quad g(x) = p_b(x) - p_s(x) > 0 \quad //\text{such that profit}>0,
$$

where $b_{cv}, s_{cv}$ are buyer’s and seller’s central points and $p_b(x), p_s(x)$ are their price functions.

Being a nonlinear programming problem, the corresponding algorithm is relatively hard to implement, as the calculation of the norm involves a square root
function. For this reason, I amended the system to perform instead the following minimization:

$$\min f(x) = ||x - b_{cv}|^2 + ||x - s_{cv}]|^2$$  //min sum of distances$^2$ to the central values
such that $g(x) = p_{b}(x) - p_{s}(x) > 0$  //such that profit$>0$,
where $b_{cv}, s_{cv}$ are buyer’s and seller’s central points and $p_{b}(x), p_{s}(x)$ are their price functions.

While the solutions for these two problems are not always identical, the switch does not affect the ultimate goal of finding the most reliable point between two users (sec 5.6). In fact, the second version, as it squares the distances, puts more weight on more distant points, and therefore, throws away the solutions that lie close to one user, but are too far away from the other. So it tends to balance out the distance from the user central values.

The main reason for a change, however, is the fact that the second problem turns out to be much easier. Since both $f(x)$ and $g(x)$ are at most second degree polynomials, the task can be identified as a quadratic programming problem that has many relatively easy solutions. IVAN uses an algorithm described in *Nonlinear programming* [27] book by D.Bertsekas. See implementation section 3.5 for the algorithm specifics.

**Step 2. Create a table of surpluses based on (buyer, seller) optimal points from Step 1**

As all the pairs of buyers and sellers finish with the procedure described in Step 1, their corresponding pair wise optimal values get stored in a two-dimensional database table $B$. The cells $B[i,j]$ of $B$ hold the total surplus generated by a possible transaction between a buyer $i$ and a seller $j$ and their reservation prices. As mentioned before, a
surplus is a difference between the reservation price of the buyer and the reservation price of the seller for a particular product.

So for each \((buyer\_agent,\ seller\_agent)\) pair on the market that can generate a positive surplus, IVAN associates a price interval \([seller\_reservation\_price, buyer\_reservation\_price]\), in which future negotiations will take place.

**Step 3. The core adjustment**

Each buyer and seller, however, is not alone on the market. In fact, they constantly compete against each other and, therefore, affect the general price trends on the market. In the next step, IVAN takes care of this issue. The system ensures that independent of the result of the next cycle negotiations between buyers and sellers, no subset of users will be dissatisfied with the result the system produces and would not conclude that they are better off trading outside of IVAN.

This set of product specifications is called “the core.” Core is a solution concept for coalitional games that requires that no set of players be able to break away and take a joint action that makes all of them better off.

To take care of this problem, IVAN tightens the boundaries of the negotiation intervals obtained in step 2 above. The section 3.6 describes the algorithm that drives this operation while the section 5.8 discusses in great detail why and how the core should be used.
Step 4. Optimizing the system matching for the overall welfare maximization

At the end of a predefined time $t$, which can be either a constant, such as one day, or a variable that depends on the number of active participants on the market, IVAN stops registering new agents and performs a matching session. The system takes the contents of the table obtained in step 3, which represent the surpluses of all possible pairs on the market and selects the best pairs. The algorithm that decides which users get matched is designed in a way to ensure that the final buyer-seller pairing set maximizes the combined welfare of all parties involved in the matching procedure.

Here is how the problem looks in mathematical terms. Suppose there are $m$ active buyer and $n$ active seller agents on the market (figure 4). First, an algorithm builds a graph $G$ with vertices being buyers and sellers and edges being transactions between them. Each edge has a weight assigned to it that is equal to the surplus of the transaction between a buyer and a seller at their “best value” product (section 2.2, step 1). Now the problem translates into finding a set of edges (i.e. pairs) that maximizes the sum of weights, with a constraint that at most one edge can come out of each vertex.
Figure 4: Matching between $m$ buyers and $n$ sellers

In this example, there are 3 buyers and 2 sellers. Buyer2 and seller2 have no edge connecting them, as their negotiation subspaces do not intersect. The numbers on the edges indicate a surplus from a possible transaction. Matched pairs are indicated with bold edges. The total surplus is $81.25.

Formally,

$$
\max f(x) = \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij}(b_i - s_j))
$$

such that

$$
\sum_{j=1}^{n} (x_{ij} b_i) \leq 1,
\sum_{i=1}^{m} (x_{ij} s_j) \leq 1,
\ x \in \{0,1\},
$$

given reservation prices $b_1, ..., b_m$ for the buyers and reservation prices $s_1, ..., s_n$ for the sellers.

This problem is called a weighted bipartite matching. Another version of it is called an assignment problem. The difference is that in the latter, the number of pairs that fall into the final solution is maximized first. There is no particular reason why IVAN uses one version of the problem over the other. It just seemed to me that the number of pairs matched is not that important for the final solution.

There are several ways of solving a weighted bipartite matching problem. One can easily notice, for example, how this problem can be transformed into a linear programming task. For that, one needs to notice that a constraint $x \in \{0,1\}$ can be transformed into the whole continuous interval $[0,1]$ without changing a solution set. This trick is based on the fact that the solutions to the linear programming problems always lie on the boundaries (in this case, points 0 and 1).

Linear programming tasks have been solved millions of times and numerous amounts of preprogrammed libraries exist for them. Nevertheless, linear programming is
not the most efficient solution in this case. The most commonly used simplex method, for example, takes an exponential time in the worst-case scenario, and therefore cannot be acceptable. For bipartite matching, alternative solutions have been found that take into account the specifics of the problem and guarantee polynomial-time solutions. A Two-sided matching [30] by Alvin E. Roth and Marlida A. Sotomayer provides an excellent review of weighted matching problems applications and different ways of solving them.

The solution implemented in IVAN is based on a problem that first relaxes the problem and disregards the weights of the edges, solves it, and then adds back the weight complexity. For a complete solution and the pseudo-code see the implementation section 3.7 below. V.M. Bondarev also describes this solution in a very understanding and easy format in his book Osnovy Programmirovaniya [26].

**Step 5. The negotiation process**

Now each pair that has been chosen in the matching process in step 4 has to mutually agree on their final price. This step is necessary, as for any positive-profit trade, the buyer and the seller prices differ. Moreover, the agents in the system can only approximate the real desires of the users and, therefore, cannot bind them with some system-calculated value. In this step, the system can only serve as an advisor, while the final transaction decision has to be made by buyers and sellers themselves. The discussion section 5.7 explains this reasoning in more detail.
In step 3, a system already pre-calculated the price interval, in which negotiations can take place. A buyer gets the higher end of the interval, and the seller – the lower one as an initial price suggestion. Both users do not know anything about the price that their opponent was offered. Now a buyer and a seller can make adjustments to the initial system-generated suggestion. Note that each user can make only one correction to his price (see sec. 5.7 for explanation). At this moment, the system ends up with both the ask and the bid price for the transaction product. If the price difference is still positive, the final surplus is divided in half, and the price gets set in the middle of the amended interval. Otherwise, if the final price of the seller is greater than that of the buyer, the deal gets cancelled.
Both parties can also decide to postpone the decision-making by ignoring the negotiation process, if they feel that they can obtain a better match in the next cycle. In that case they can obviously lose the current deal.

2.3. Dynamics of the system

Each buyer/seller can enter the system at any time. His/her Utility Function will be instantly calculated (see 3.1) and stored in the database. The system, however, will wait for the next matching cycle to actually give the user his transaction partner. As the cycle gets triggered, user’s agent gets compared to all possible partners (sec. 2.2, step 1). Finally, the results will be available for the system in terms of table entries (sec. 2.2, steps 2 and 3). If the negotiation process (sec. 2.2, step 5) fails or gets suspended, the user can stay for the next cycle or redefine his preferences.
Chapter 3. Implementation

3.1. Market choice

For the purposes of making use of all the features of the system, IVAN's infrastructure focuses on the context of a “services marketplace” in which language translation services are bought and sold. A benefit of concentrating specifically on “services marketplaces” and “information goods” is that this allows one to concentrate more on the attributes relevant to the parties attempting to engage in the transaction, without getting overwhelmed by the material details inherent to the product itself. For many information goods, the characteristics of the seller actually serve to define the good itself. For instance, in a language translation marketplace, the fact that the seller of the translation service is considered an expert, and has a high reputation rating associated with her, implicitly conveys the nature and quality of the “good,” in this case the translation service. Indeed, one can argue that in the context of services marketplaces, in which the service being bought and sold lacks tangible manifestation and is hence not as easily susceptible to objective evaluation, the ability to be able to ontologically segregate and prioritize the various subtle impinging factors gains significance and relevance. Hence, the choice of services marketplaces, in which intangible services and information are bought and sold, is an appropriate and fitting choice as the target application domain for this project.

In order to be able to participate in the IVAN marketplace, a “seller” creates a “selling agent” that is aware of its owner's level of expertise, availability, compensation
expectations, and other special constraints, such as requirements for the buyer. Similarly, a “buyer” creates a “buying agent” that understands the exact needs of its owner - such as degree of expertise desired, time sensitivity or urgency with which information is needed, range and type of compensation that the buyer is willing to offer the seller, and other special constraints, such as minimum requirements on the seller's reputation level. Additionally, the buying (selling) agents also encapsulate information on how different qualified sellers (buyers) can be rank-ordered according to the degree of relative preference. Subsequently, the system automatically matches buyers and sellers.

Finally, it is important to note that although the system is built around the “language translation” example, its structure is very general and can easily be ported to any other marketplace solution.

3.2. Implementation notes

IVAN is currently in a working condition and can be accessed at the following site: http://18.27.0.94:8080/. It runs on an Apache server [31]; the JSP support is provided by Caucho’s Resin software [32]. All the code is written in Java. Finally, MySQL database software [33] was used for storage. The main reason for choosing these particular products was the fact that they are free of charge for academic purposes. Moreover, they are immediately available for download on the Internet.

Due to the nature of the selected supporting products, IVAN is not platform-specific. The server can be hosted on any modern platform, and the user just needs to have an Internet browser to use the system.
3.3. The program

Since IVAN is implemented in JAVA, it follows the standard guidelines of Object-Oriented Programming. The whole program is split into four transparent layers: API, Math, low-level market abstractions, and high-level market abstraction (see figure 6).

![Figure 6: Modular Dependency Diagram](image)

IVAN consists of four layers, only one of which talks to the rest of the world.

As advised in Java literature, only one layer in a package is opened to the outside world, the API layer. It is usually done, so that a GUI can be independently built on top of the program. Although ideally an API should consist of only one class, I have decided to split it into two (Login and ExtraProcedures are used only by my version of GUI), as these classes are conceptually very independent. It seemed to me that mixing database querying with math functions would have only confused a potential user.
Math layer attempts to separate mathematical issues from the rest of the program building. It also builds the foundation for easier algorithm development. In particular, the Matrix class is designed to simulate as much as possible the equivalent class in MATLAB. In this class various matrix operations are implemented, as well as some important transformations. Matrix class greatly facilitated the implementation of various algorithms in IVAN, such as least squares, matching, and maximum finding.

The Function class also helped greatly to code the rest of the program. This class represents a general math function and makes it possible to implement algorithms with greater flexibility. For example, instead of creating two different procedures for linear and second-order least squares, IVAN uses a general Function that automatically expands data matrices, as needed. Since the Function is implemented for an n-dimensional case, it was also very useful for the quadratic programming implementation.

As for the other two layers, the higher- and the lower-level market abstraction, they were used internally for convenience purposes and do not present any significant interest for this paper.

The rest of the sections in this chapter describe the algorithms used for the most interesting and crucial parts of the system. In particular, Section 3.4 describes the algorithms used to implement a least squares method. This method plays a core role in the creation of the agent function that simulates user’s behavior on the market. Section 3.5 discusses the implementation of the quadratic programming problem, used to find the most appropriate trading product for a (buyer, seller) pair. Section 3.6 discusses the
implementation of the core. Finally, section 3.7 talks about the implementation of the weighted matching problem that is used to match users at the end of each market cycle.

### 3.4. Least Squares

Consider a linear system of equations having $n$ unknowns but $m > n$ equations. In terms of matrix equations, we wish to find a vector $x \in \mathbb{C}^m$ that satisfies $Ax = b$, where $A \in \mathbb{C}^m \times n$. This system of equations is overdetermined. Therefore, in general, it has no solution. We can, however, find $x$ that minimizes a residual $\| r \| = \| b - Ax \|$. This is the formation of a general (linear) least squares problem.

While theoretically there are many ways of solving the least squares problem, the one used in practice needs to have some very specific qualities: it needs to have a fast implementation and be error-stable (i.e. have a very small cumulative error). The one used in IVAN is based on the QR factorization. A book by L.Treffethen called *Numerical Linear Algebra* [25] helped me tremendously while searching for the most appropriate algorithm. In particular, in discusses which algorithms should be used in which situations.

Here is the top-level pseudo-code for the implemented algorithm. The whole program takes $\sim 2mn^2 - \frac{2}{3}n^3$ time flops, where $(m,n)$ is the size of the matrix $A$.

```plaintext
procedure LeastSquares(A,b)
1. Compute QRfactorization(A)
2. Compute the vector $Q^T b$ (already done in my version of step 1)
3. Solve the upper-triangular system $R \cdot x = Q^T b$ for $x$ (using backSubstitution(R, $Q^T b$))
```

31
The first step, where the QR factorization is done, dominates a least squares computation. Here a system performs a matrix transformation that finds an orthogonal matrix \( Q \) and an upper-triangular matrix \( R \), such that \( QR = A \). The \textit{Householder orthogonalization} is considered nowadays to be the best QR transformation algorithm:

\[
\text{procedure QRfactorization}(A) \\
\text{for } k = 1 \text{ to } n \\
x = A_{k,m,k} \\
v_k = \frac{\text{sign}(x_k) \cdot ||x|| \cdot e_j + x}{v_k} \\
v_k = v_k / ||v_k||_2 \\
A_{k:m,k:n} = A_{k,m,k:n} - 2 \cdot v_k^T (v_k^T A_{k:m,k:n}) \\
/\!/ \text{implicit Calculation of a product } Q^Tb \text{ (the most usual way of using QR factorization)} \\
b_{k:m} = b_{k,m} - 2 \cdot v_k^T (v_k^T b_{k:m}) \\
\text{end}
\]

As for the back substitution, its calculation is pretty straightforward. The program consecutively solves all the equations from the bottom to the top and takes only \( \sim m^2 \) time flops:

\[
\text{procedure backSubstitution}(R,b) \\
x_m = b_m / r_{m,m} \\
x_{m-1} = (b_{m-1} - x_m \cdot r_{m-1,m}) / r_{m-1,m-1} \\
x_{m-2} = (b_{m-2} - x_m \cdot r_{m-2,m-1} - x_{m-1} \cdot r_{m-2,m}) / r_{m-2,m-1} \\
\ldots \\
\ldots \\
x_j = (b_j - \sum_{k=j+1}^{m} (x_k \cdot r_{jk})) / r_{jj}
\]

3.5. Quadratic programming

To solve this problem I used a barrier method, described in detail in \textit{Nonlinear Programming} [27].
Given a function \( f(x) \) subject to constraints \( g_i(x) \leq 0 \), one can create an auxiliary “barrier” function \( B(x) = -\sum \ln \{ -g_i(x) \} \). The barrier method is defined by introducing a parameter sequence \( \{\epsilon^k\} \) with

\[
0 < \epsilon^{k+1} < \epsilon^k, \quad k = 0, 1, \ldots, \quad \epsilon^k \rightarrow 0.
\]

It consists of finding

\[
x^k = \arg \min_{x \in \text{interior}} \{ f(x) + \epsilon^k B(x) \}, \quad k = 0, 1, \ldots
\]

The remaining task is pretty straightforward. In the case of IVAN, one can even avoid picking the sequence \( \{\epsilon^k\} \). Since both \( f(x) \) and \( g_i(x) \) are known to be at most a polynomial of the second degree (attributes have constant constraints, while the inequality \( \text{surplus} > 0 \) is in the order of least squares approximation function), one can even calculate a derivative and then set \( \epsilon^k \) to 0.

For example, given a problem

\[
\begin{align*}
\text{minimize} & \quad f(x) = \frac{1}{2}(x_1^2 + x_2^2) \\
\text{subject to} & \quad 2 \leq x_1,
\end{align*}
\]

the barrier \( B(x) \) will be \(-\ln(x_1 - 2)\). Then taking a derivative of

\[
f(x) + \epsilon^k B(x) = \frac{1}{2}(x_1^2 + x_2^2) - \epsilon^k \ln(x_1 - 2)
\]

and setting it to 0 yields

\[
\begin{align*}
2x_1 - \epsilon^k/(x_1 - 2) &= 0 \\
2x_2 &= 0,
\end{align*}
\]

or, equivalently, the point \((1 + \sqrt{1+\epsilon^k}, 0)\). Now setting \( \epsilon^k \) to 0 would return \((2, 0)\).

### 3.6. The core

Given \( m \) buyers and \( n \) sellers trading the same product, computing the core is a very straightforward procedure:
1. Put all buyers and sellers in an increasing order.

\[ s_1 \leq s_2 \leq \ldots \leq s_n \]
\[ b_1 \leq b_2 \leq \ldots \leq b_m \]

2. Throw out extreme buyers and sellers, i.e.

- all buyers \( b_i \), for which \( b_i \leq s_1 \)
- and all sellers \( s_i \), for which \( b_m \leq s_i \)

3. Now simply to throw away the excess of buyers/sellers and set the interval accordingly:

\[
\text{if } m > n \\
\quad \text{if } s_n \leq b_{m-n+1} \\
\quad \quad \text{core} = [ \max(s_n, b_{m-n}), b_{m-n+1} ] \\
\quad \quad \text{else} \\
\quad \quad \quad \text{core} = \emptyset \\
\quad \quad \text{end} \\
\quad \text{end} \\
\text{if } m < n \\
\quad \text{if } s_m \leq b_1 \\
\quad \quad \text{core} = [ s_m, \min(s_{m+1}, b_1) ] \\
\quad \quad \text{else} \\
\quad \quad \quad \text{core} = \emptyset \\
\quad \quad \text{end} \\
\quad \text{end} \\
\text{if } m == n \\
\quad \text{if } s_m \leq b_1 \\
\quad \quad \text{core} = [ s_m, b_1 ] \\
\quad \quad \text{else} \\
\quad \quad \quad \text{core} = \emptyset \\
\quad \quad \text{end} \\
\quad \text{end}
\]

3.7. The weighted matching problem
Let there be given \( m \) applicants for \( n \) jobs and a matrix \( A \), whose elements \( A_{ij} \) indicate the profit of assigning a candidate \( i \) to a job \( j \). The goal is to find such a pairing between applicants and jobs that maximizes the total profit.

This problem is called a weighted matching problem. And in the context of IVAN it is used to match buyers and sellers on the market. In order to solve this problem, let’s first have a look at its simplified version, in which the utility of an applicant can be only either 0 or 1 (i.e. either present or not).

So the goal is to maximize the following function:

\[
\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_{ij} \rightarrow \max,
\]

where \( a_{ij} \) = 1, if an applicant is useful and 0, if not;
\( x_{ij} \) = 1, if an applicant \( i \) got assigned to a job \( j \) and 0, if not.

To solve this problem, let’s first create a graph \( G = (V, E) \), where vertices \( V = I \cup J \) consist of all buyer and seller agents and edges \( E \) consist of all possible transactions between them. Clearly, all edges have one end in \( I \) and one – in \( J \). Moreover, \( I \cap J = \emptyset \) (figure 7). Now let’s create some random (buyer, seller) pairing \( M \), for example, by letting the first buyer to choose his partner, then the second one to choose from the remaining sellers, etc. We’ll say that if vertex \( x \in M \), then \( x \in I^+ \) or \( J^+ \), depending whether \( x \) represents a buyer or a seller; otherwise, \( x \in I \) or \( J \). Next, we set the direction in the graph: if an edge \( (x, y) \in M \), we set its direction from \( J \) to \( I \); if an edge \( \notin M \), the direction is reverse. The goal is, obviously, to maximize the number of edges in \( M \).
Figure 7: A directional graph without weights

Here, the thick edges are the ones that belong to $M$. Vertices $1, 2, 3, 4 \in I^*$; $1', 2', 3', 4' \in J^*$; $5 \in I$; $5' \in J$

Graph $G$ has several paths formed by the edges, such as $3'-2-1'-1$ in figure 7. Clearly, any such path has interchanging elements from $I$ and $J$. Suppose we found a path $[s, \ldots, t]$ where $s \in I$ and $t \in J$. Such a path would have an odd amount of edges. Moreover, the amount of edges not belonging to $M$ would always exceed the ones that belong to $M$ (if two edges belong to the pairing $M$, they cannot share the same vertex). Then we can swap the direction in these edges, put in $M$ all the edges from the path that were not in $M$, and remove from $M$ the ones that were. Clearly, the number of edges in $M$ would then increase. For example, in a graph from figure 7, we can create a path $[5, 4', 4, 2', 3, 5']$, and swap edges $\{(4',4), (2',3)\}$ with $\{(5,4'), (4,2'), (3,5')\}$.

This procedure would be the base for our algorithm. And here is the pseudo-code for solving the problem:

```
procedure maxPairing
  1. (Initialization):
     1.1. Create the first pairing $M$ for example by using a “greedy” grab-first method described above.
     1.2. Build a directional graph $G$, as described above.
  2. (Main Cycle):
     2.1. For all vertices $i \in I$,
     find all vertices that are reachable from $i$.
     If any of these vertices is a vertex $j \in J$, then swap all edges in the path $(i, \ldots, j)$, as described above and fix $M$ and $G$ accordingly.
```

36
This algorithm uses a function that finds all vertices reachable from some vertex \( s \). The easiest way to create this function is to use a standard graph search algorithm and mark all the reached vertices on the way. Here’s an example of how to do it:

```
Procedure reachableVertices(vertex s)
1. (Initialization) \( a=z=1; \ Q[a]=s. \ for \ i=1,...,n \ R[i]=false; \ P[i]=false. \)
2. (main cycle)
   a. Here we look at all the vertices that follow after \( Q[a] \) in the graph. If
   for a vertex \( k \) \( R[k]=false \), then \( z=z+1; \ Q[z]=k; \ P[k]=Q[a]; \ R[k]=true. \)
   b. \( a=a+1. \) If \( a\leq z \), then go to 2.1.
3. (End) Return \( P. \)
```

Having solved the complementary problem, we can return to the original one with edges having weights. In our problem, we need to maximize the total surplus function. But, clearly, a minimizing problem is an equivalent one. Since it happens to be the one that is easier to solve, here is how the two can be converted to each other:

```
We need to maximize \( f = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_{ij} \rightarrow \text{max}. \)
Here I assume that matrix \( A \) has only positive elements.
Let \( r \) be the biggest element of \( A. \)
Now create Matrix \( A' \), such that \( a'_{ij}=r-a_{ij} \) for all elements of \( A. \)
Clearly, all elements of \( A' \) are also positive.
Now, \( f' = \sum_{i=1}^{m} \sum_{j=1}^{n} a'_{ij} x_{ij} = \sum_{i=1}^{m} \sum_{j=1}^{n} (r-a_{ij}) \cdot x_{ij} = (\sum_{i=1}^{m} \sum_{j=1}^{n} r \cdot x_{ij}) - f = \text{const} - f \)
So clearly by minimizing \( f' \), we maximize \( f. \)
```

And here is the algorithm that solves the minimizing problem. Steps 2.1 and 2.2 below merely reduce all elements in the respective row/column by the value of the smallest element in that row/column. This way, the smallest element in a row/column is
reduced 0, without changing the problem. Operation $E(I', J', \alpha)$ from step 2.5 subtracts $\alpha$ from each row $i \in I'$ and adds it to all columns $j \in J$. For more information and proofs, see the book *Osnovy Programmirovaniya* [26] written, conveniently, in Russian.

```plaintext
procedure minWeightedMatching(Matrix B)
1. (Initialization) Initialize Matrix BB and copy the contents of B into BB
2. (Main step)
   2.1. reduceByRows(B); //substract a min(rowi) from all elements of rowi
   2.2. reduceByColumns(B); //substract a min(columnj) from all elements of columnj
   2.3. use maxPairing procedure with a graph G(V, E) of m+n vertices; edge $(v_i, v_j) \in E$, if if $B_{ij}=0$
   2.4. if the result of 2.3 consists of $\min(m,n)$ edges, go to 3. Otherwise, save into $I'$ and $J'$ the numbers of vertices (from $I$ and $J$) that are reachable from unmarked vertices of $I$.
   2.5. $\alpha = \min Bij,$
        $i \in I, j \in J/J'$
        use operation $E(I', J', \alpha)$
        go to Step 2.1
3. (End). Calculate $f$ using BB according to the calculated pairings.
           Return the pairings and $f$.
```

### 3.8. Materials and support

Most of the system was implemented and built on a local home Linux-based computer, with some testing and designing being done on UNIX-based on-campus Athena workstations. The latest up-to-date version is temporarily available at [http://18.27.0.94:8080/](http://18.27.0.94:8080/)

The main algorithms used in IVAN were picked up long before the idea for this project even appeared. Initially, they were used for completely independent projects written for MIT Computer Science and Mathematics classes, namely 6.046 (Introduction to Algorithms), 18.327 (Wavelets and Filterbanks), 6.252 (Nonlinear programming), and
18.335 (Numerical Methods). IVAN, however, turned out to be a very convenient system to implement and use most of the material learned in the classroom for a real-life project.
Chapter 4. Experimentations

4.1. Problems with obtaining real life data

Unfortunately, it is very difficult to persuade other people to test a product without offering them some incentive. As a person who cannot afford to pay people to test my system, I was forced to skip that important part of the system creation. As for the friends, it seemed to me that they would not create a random sample from a population, and, therefore, would not bring useful results that are worth the trouble. In addition, the sample translation market that I created is so distant to their interests that they would not have been able to price any products fairly.

The only real testing of the program would be offering it to the general public, while fixing it in the meantime. That would effectively involve creating a commercial product, for which I, obviously, have neither time, nor money.

So, all the results and estimations of the user behavior for this paper were obtained through either personal experience of dealing with the program or just through plain “mental experiments.”

4.2. An example of using the program

In order to facilitate the understanding of IVAN and to show that it really works in practice, I will follow closely one market example. In this section, I will use a market, where users buy and sell translation services.
First, as an administrator, I have to create market specifications and indicate which attributes does the market accept. Since it’s a translation market, I found the following attributes to be relevant: *buyer reputation*, *seller reputation*, *seller skill*, *number of words*, and *time to completion*.

Next, I need to create several profiles. For this example, I wrote two profiles that are supposed to be used by a professional and casual buyer and two profiles for the seller. All the profiles are written in a standard and easy to use XML format.

Now I can enter the actual user data. Here I have chosen five users: two professional buyers (buyer1 and buyer 2), one casual buyer (buyer 3), one professional seller (seller 1), and one casual seller (seller2). Table 8 below shows these users’ exact specifications.

<table>
<thead>
<tr>
<th>Buyer</th>
<th>Buyer 2</th>
<th>Buyer 3</th>
<th>Seller 1</th>
<th>Seller 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Buyer reputation</strong></td>
<td>10</td>
<td>10</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td><strong>Seller reputation</strong></td>
<td>Center: 7</td>
<td>Center: 7</td>
<td>Center: 7</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: [5,10]</td>
<td>Range: [5,10]</td>
<td>Range: [4,10]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-0.1x^2+2x-4.1</td>
<td>-0.1x^2+2x-3.1</td>
<td>Constant</td>
<td></td>
</tr>
<tr>
<td><strong>Seller skill</strong></td>
<td>Center: 5</td>
<td>Center: 5</td>
<td>Center: 4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>-0.1x^2+x+2.5</td>
<td>-0.1x^2+x+3.5</td>
<td>x+3</td>
<td></td>
</tr>
<tr>
<td><strong>Number of words</strong></td>
<td>4000</td>
<td>4000</td>
<td>4000</td>
<td>4000</td>
</tr>
<tr>
<td><strong>Time to completion</strong></td>
<td>Center: 60</td>
<td>Center: 60</td>
<td>Center: 50</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Range: [30.90]</td>
<td>Range: [30.90]</td>
<td>Range: [40.60]</td>
<td>35.100</td>
</tr>
<tr>
<td></td>
<td>-0.1667x+15</td>
<td>-0.1667x+16</td>
<td>-0.1x+12</td>
<td>-0.1143x+12</td>
</tr>
<tr>
<td><strong>Central value price</strong></td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 8: User specifications
Now, as the system has already prepared all the agents and their $n$-dimensional functions, I can simulate the end of the cycle by logging in as an administrator and triggering the system to match active buyers and sellers.

In this step, the system has to calculate the initial profit table, as was described in sec. 2.2 (step 2) and then adjust it, according to the calculated core. Table 9 below shows all the results that IVAN produced, when following these steps. Notice, for example, that although buyer1 has originally had a positive surplus with both sellers, these transactions were voided, as they fell outside of the core.

<table>
<thead>
<tr>
<th>Buyer 1</th>
<th>Seller 1</th>
<th>Others</th>
<th>Core</th>
<th>Negotiation interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>This pair</td>
<td>$P_b=5.03$</td>
<td>Buy</td>
<td>$5.03$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$P_s=4.54$</td>
<td>Sell</td>
<td>$-0.81$</td>
<td></td>
</tr>
<tr>
<td>Surplus</td>
<td>$0.48$</td>
<td></td>
<td>$6.03$</td>
<td></td>
</tr>
<tr>
<td>Surplus</td>
<td>$0.56$</td>
<td></td>
<td>$5.03$</td>
<td></td>
</tr>
<tr>
<td>Core: $(5.03, 6.03]$</td>
<td></td>
<td>Negotiation interval: $\emptyset$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buyer 1</th>
<th>Seller 2</th>
<th>Others</th>
<th>Core</th>
<th>Negotiation interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>This pair</td>
<td>$P_b=5.43$</td>
<td>Buy</td>
<td>$5.43$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$P_s=4.88$</td>
<td>Sell</td>
<td>$4.88$</td>
<td></td>
</tr>
<tr>
<td>Surplus</td>
<td>$0.56$</td>
<td></td>
<td>$6.43$</td>
<td></td>
</tr>
<tr>
<td>Surplus</td>
<td>$0.56$</td>
<td></td>
<td>$5.43$</td>
<td></td>
</tr>
<tr>
<td>Core: $(5.43, 6.43]$</td>
<td></td>
<td>Negotiation interval: $\emptyset$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buyer 2</th>
<th>Seller 1</th>
<th>Others</th>
<th>Core</th>
<th>Negotiation interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>This pair</td>
<td>$P_b=6.03$</td>
<td>Buy</td>
<td>$9.90$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$P_s=4.54$</td>
<td>Sell</td>
<td>$-0.81$</td>
<td></td>
</tr>
<tr>
<td>Surplus</td>
<td>$1.48$</td>
<td></td>
<td>$6.03$</td>
<td></td>
</tr>
<tr>
<td>Surplus</td>
<td>$1.48$</td>
<td></td>
<td>$5.03$</td>
<td></td>
</tr>
<tr>
<td>Core: $(5.03, 6.03]$</td>
<td></td>
<td>Negotiation interval: $(5.03, 6.03]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buyer 2</th>
<th>Seller 2</th>
<th>Others</th>
<th>Core</th>
<th>Negotiation interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>This pair</td>
<td>$P_b=5.99$</td>
<td>Buy</td>
<td>$5.99$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$P_s=5.00$</td>
<td>Sell</td>
<td>$5.00$</td>
<td></td>
</tr>
<tr>
<td>Surplus</td>
<td>$0.99$</td>
<td></td>
<td>$5.93$</td>
<td></td>
</tr>
<tr>
<td>Surplus</td>
<td>$0.99$</td>
<td></td>
<td>$5.23$</td>
<td></td>
</tr>
<tr>
<td>Core: $(5.23, 5.93]$</td>
<td></td>
<td>Negotiation interval: $(5.23, 5.93]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buyer 3</th>
<th>Seller 1</th>
<th>Others</th>
<th>Core</th>
<th>Negotiation interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>This pair</td>
<td>$P_b=7.00$</td>
<td>Buy</td>
<td>$7.00$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$P_s=3.64$</td>
<td>Sell</td>
<td>$-0.14$</td>
<td></td>
</tr>
<tr>
<td>Surplus</td>
<td>$3.36$</td>
<td></td>
<td>$5.58$</td>
<td></td>
</tr>
<tr>
<td>Surplus</td>
<td>$3.36$</td>
<td></td>
<td>$4.88$</td>
<td></td>
</tr>
<tr>
<td>Core: $(4.88, 5.58]$</td>
<td></td>
<td>Negotiation interval: $(4.88, 5.58]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buyer 3</th>
<th>Seller 2</th>
<th>Others</th>
<th>Core</th>
<th>Negotiation interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>This pair</td>
<td>$P_b=5.99$</td>
<td>Buy</td>
<td>$5.99$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$P_s=5.00$</td>
<td>Sell</td>
<td>$5.00$</td>
<td></td>
</tr>
<tr>
<td>Surplus</td>
<td>$0.99$</td>
<td></td>
<td>$5.93$</td>
<td></td>
</tr>
<tr>
<td>Surplus</td>
<td>$0.99$</td>
<td></td>
<td>$5.23$</td>
<td></td>
</tr>
<tr>
<td>Core: $(5.23, 5.93]$</td>
<td></td>
<td>Negotiation interval: $(5.23, 5.93]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Finally, IVAN performed the matching procedure on the remaining pairs. Here's what the system had produced:

So, in the end a professional buyer2 got matched with a professional seller 1, and casual buyer3 ended up with a casual seller2.

At this point, if both pairs negotiate their final prices successfully, their agents would expire and only buyer1 would remain active for the next matching.
Chapter 5. Discussion

5.1. Least squares vs. adding the functions

As it was already mentioned in section 2.1 (step 3), IVAN uses the method of least squares to convert user’s $n$ 1-dimensional attribute functions to an $n$-dimensional utility function. The choice of this particular approximation method, however, does not seem very clear and convincing and requires some clarification.

As it was stated in the design section above, all the attribute functions supplied and defined by the user are orthogonal to each other (since they lie on perpendicular planes). Moreover, the base function $f$ for the least squares operation is calculated as a sum of these attribute functions, and all the design points used in the approximation lie on them, as well.

The goal of least squares is to minimize the norm of the residual $r = f(X)c - y$. In our case, however, this statement is equivalent to $r = f(x_1, x_2, ..., x_n)c - [ f_1(x_1) + f_2(x_2) + ... + f_n(x_n) ] - [ f_1(x_1) + f_2(x_2) + ... + f_n(x_n) ]$. So, clearly, by taking $c = 1$, we can always achieve a zero residual.

This result then raises a natural question: what is the purpose of using this rather complex and time-consuming method of approximation, if exactly the same result can be achieved by simply extending the original attribute functions into $n$ dimensions and adding them up?

The answer to this question lies in the same section 2.1 (step 3). The system, it says, does not merely stop its data mining process after it finishes retrieving the
information from the attribute functions. Upon user’s request, it goes on and adds some
relevant transaction data from the past or even asks the user to evaluate more points. With
a high degree of probability, it can safely be assumed that none of these “new” points fall
on the original functions. Then, of course, the least squares operation seems required.

It is still not clear, however, why a system would add these new points to its
approximation and even choose the same base function for the least squares method in
the first place. After all, the new points (at least, the “historical” ones) are clearly “noisy”
and do not fit well into any of the original functions. While section 5.3 concentrates in
more detail on the problem with the noise, this section explains, why the base function
remains the same.

The main goal of using additional “random” points is to diverge from the very set
of points obtained on the original functions. While a user specifies exactly in his profile a
general form of his attribute preference curve, it is obviously an approximation.
Moreover, three points selected by the system on a function, although being able to
specify this curve exactly in mathematical terms, will clearly fail in reality. Any other
three points would have given different curve specifications. On such a huge
approximation scale, it is not very important, whether the least squares residual is exactly
zero or not. It is, after all, the purpose of the method of least squares to being able to pick
out a well-behaving function, given a set of the observation data. Moreover, as the next
section shows, a zero residual is not even always better. It is clear, however, that the more
points the systems receives, the better approximation it performs.
For these reasons, I argue, the inclusion of extra points is extremely beneficial to the system and, consequently, the method of least squares is much more preferable to the simple function addition method.

Another argument in favor of using the least squares method is purely a technical one. From the design point of view, it is likely that one might want to change the set of questions asked or even revamp the whole point retrieval process. In a well-designed system, this change should not affect any other modules of the program, including the “approximation” one. And while the function addition method heavily depends on the current design, the method of least squares does not and is, therefore, preferable from the design point of view.

5.2. Least squares vs. interpolation

While the previous section very strongly argued in favor of the least squares operation, it does not mention why it is better than any other curve-fitting method. Section 5.1 stressed the importance of a method that can reconstruct a function from any set of points. This section will compare the least squares operation to an alternative curve-fitting method.

Consider for a moment the following problem given as an example in [25]. We are given m distinct points $x_1, ..., x_m \in \mathbb{C}$ and data $y_1, ..., y_m \in \mathbb{C}$ at these points. Then there exist a unique polynomial function of degree $m-1$

$$p(x) = c_0 + c_1 x + \ldots + c_{m-1}x_{m-1}$$
with the property that for each \( x_i, p(x_i) = y_i \). Now, if we find these coefficients \( \{c_i\} \), then we’ll get a function that goes exactly through the original points. In terms of matrixes, this problem can be expressed as a *Vandermode* system:

\[
\begin{bmatrix}
1 & x_1 & x_1^2 & x_1^{m-1} \\
1 & x_2 & x_2^2 & x_2^{m-1} \\
\vdots & \vdots & \vdots & \vdots \\
1 & x_m & x_m^2 & x_m^{m-1}
\end{bmatrix}
\begin{bmatrix}
c_0 \\
c_1 \\
\vdots \\
c_{m-1}
\end{bmatrix}
= 
\begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_m
\end{bmatrix}
\]

While the polynomial solution gives a zero residual for the given points, its actual function behavior is not that pleasing (see figure 11). Near the end of the interval, \( p(x) \) exhibits large oscillations that are clearly an artifact of the interpolation process.

![Figure 11: A result of the polynomial interpolation](image-url)

In this example 10 data points were used. Axis scales are not given, as they are irrelevant.
This unsatisfactory behavior is typical of polynomial interpolation. Moreover, the fits tend to get worse, as more data are utilized.

Now, let’s solve this problem using least squares fitting technique. Given the same m points and data on them, consider now a polynomial \( p(x) \) with a degree \( n < m \). On a matrix level, this creates an over-defined equation system. The goal is to minimize the norm of the residual \( r = y - Xc \):

\[
\begin{bmatrix}
1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\
1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\
\vdots & \vdots & \ddots & \cdots & \vdots \\
1 & x_m & x_m^2 & \cdots & x_m^{n-1}
\end{bmatrix}
\begin{bmatrix}
c_0 \\
c_1 \\
\vdots \\
c_{n-1}
\end{bmatrix}
= 
\begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_m
\end{bmatrix}
\]

Figure 12 illustrates the result after using the least squares on the same points. While the new polynomial does not interpolate the data and produces a non-zero residual, it captures the overall function behavior much better.
5.3. Learning from the transaction history

The use of simple utility functions to represent user's preferences for each particular attribute severely oversimplifies user's preferences in the corresponding dimension. A well-functioning system must learn from the history of successful past transactions and improve the representation function of the user with each new committed transaction. In a situation where alternative online marketplace systems compete against each other for the clients, the one that represents user's preferences most accurately will be the most successful.

Recall that in a course of creating an agent in IVAN, a user can manually specify whether he wishes to use his own history, the history of all users with the same profile, or no history at all.

If asked to disregard all previous transaction history, the system takes \( n \) 1-dimensional functions obtained from the Q&A part (sec. 2.1, step 2) and performs the least squares operation on their discretized version to obtain an \( n \)-dimensional function that represents the user (sec. 2.1, step 3). As a result, the system obtains a very smooth and idealistic utility function that can only roughly approximate user's preferences. This function, however, can be improved with the inclusion of additional data. There are two steps in this process where historical data points can be inserted. Depending on whether the points have been added before or after the least squares method has been performed, different goals will be achieved.

In a sense, the addition of historical data points after the approximation has already been made gives a real-world correction to an otherwise idealistic function. As a result, the corrected function becomes “noisy,” as the actual deals rarely fall on the
system-defined curve and usually vary widely from one case to another. For a better approximation, this noise needs to be reduced. If the noise cancellation returns the function to its original form, it hardly means that the idealistic function was an excellent estimator. That rarely happens, although. Noise cancellation also helps IVAN in the calculations it performs the other parts of the system. IVAN, for example, eventually needs to run some maximization algorithm on its $n$-dimensional agents (section 2.2, step 1), so the functions representing these agents need to be smooth enough for the algorithm to work. Such a smoothing procedure exists. It is called a wavelet transformation and is often used in signal processing algorithms to cancel noise effects. Unfortunately, so far only 2-D wavelet theory has been developed. In the context of IVAN, this method is useful only for 1- and 2-attribute markets. For that reason, IVAN cannot use this seemingly attractive tool, and consequently does not add historical data after the least squares approximation results.
Figure 13: The deficiency of the current IVAN method

Here, 13a shows the real user preferences (data) and the ones reported in the transactions (data with noise). Even after taking into account all the real data, one can clearly see how bad the ideal function approximates user's behavior (b). Meanwhile, the wavelet method (used here with Debauche 21 filter) not only captures the oscillating nature of the original data, but also damps the noise level in this example by more than 20%.
The other option left is adding history points first and then performing the least squares operation. While still taking history into account, this option leaves the final function that represents the user to be of the same order as the original one. Therefore, the amended function still “looks” as idealistic. Considering, however, that some history learning has been attained, IVAN incorporated this option in its code.

Unfortunately, as figure 13 shows, sometimes the current method fails. It is especially true, if a user cannot find a satisfactory curve among the ones offered in the profile. Clearly a new solution has to be found that uses the first option mentioned above and handles well even those ugly cases. Wavelets were merely given as an example of such a solution, but another alternative method should be found.

5.4. How to ask questions

A lot of research has been done on how to ask user questions in order to receive the most accurate information about his preferences. Ely Dahan’s group at MIT Sloan School claims, for example, that respondents cannot even correctly price the product they are interested in. The only thing they can do is to compare products (which they call cards). If a respondent is given a list of products, the only thing he can do accurately is to comparatively rate them. Unfortunately, the effective usage of the comparative technique requires some a-priori knowledge of the product prices, which IVAN cannot possess due to its complete market-independence. So although I agree with the findings of Dahan’s group, IVAN ends up asking the exact prices for the products.

The current version of the system asks the users to give their reservation prices for some set of products. As described in section 2.1 (step 2), almost all the points
corresponding to these products are concentrated around the “central value” and diverge from it in only one dimension (the exception border and history points are rare and do not change the overall picture).

In the first version of the program, IVAN directly listed the product descriptions for the user to price. Strangely enough, however, such an approach turned out to be very cumbersome. Since there are many attributes that contribute to the price of the product, it used to take a user a long time to evaluate each product. Even my own experiences with IVAN in that initial implementation were very negative, as I had to carefully preselect all the product prices before all the testing. It turned out to be extremely difficult to price to the products chosen by the system on the fly.

The next version of the program (sec. 2.1, step 2) took care of this problem and significantly facilitated a human-computer interaction. This time, it took into account the fact that all the points differ only by one dimension from the central value. After listing the central value product given by the user, the rest of the regular questions look like the following one: “… and how much are you willing to pay, if the Seller_Reputation is increased by 10% (to 7)?” This type of questions, while asking for the same information, is generally much easier to reply to.

Friendliness of the user interface plays a big part of a successful system. Unfortunately, as S. Garfinkel correctly pointed out in his MIT Technology Review [34] article, this issue is overlooked nowadays too often. If IVAN ever goes commercial, more research has to be done on how to get the information out of a user in the most efficient way.
5.5. How reliable are users' responses?

The whole calculating engine of IVAN is based on the answers given by the user during the Q&A session. But how reliable are these answers? Can a program simply assume that the answers it receives from users are always correct? It turns out that usually the answer is ‘no’.

Some research has been done in MIT Sloan School on how to ask and even order questions in such a way that prevents respondents from making unintentional errors. In their FastPace [35] survey system, a user is asked to put a set of products in the order of preference. After each consecutive pick, however, the system eliminates the subspace of answers that is worse off than the one just picked. Consequently, only the cards that do not fall in that subspace remain on the screen for further ranking. The respondents are usually being the most attentive in the beginning of the questioning session and wear down towards the end. FastPace targets and eliminates exactly this common source of errors. The experiments performed with this program showed very impressive results.

Some other interesting methods keep getting invented to help respondents avoid making unintentional mistakes. Respondents, however, can also intentionally report their preferences erroneously. In the context of IVAN, it is actually quite easy to see, as the users are asked to give their reservation prices. Usually, however, nobody reports these figures truthfully. A buyer tends to lower it down a bit, while a seller puts it higher. IVAN tries to take this problem into account when estimating a transaction surplus (see Section 5.6).
5.6. Choosing the best point between a buyer and a seller

At one point in the agent-creation process, each user chooses a subspace in an $n$-dimensional space, in which he is willing to negotiate. And if the intersection of these subspaces for a pair of a buyer and a seller is not empty, the system has to figure out which point would satisfy both users the most. Being very difficult computationally, this task is also one of the most important ones for the system functionality.

The main purpose of IVAN is to optimize and facilitate the trading of very complex multi-attribute products. A human brain alone can hardly solve the problem of finding the exact specifications of the most efficient trading product in a realistic time. For that reason, one the most challenging and still unclear problems in designing IVAN is figuring out what exactly this "best" product is and how to calculate it.

If the market agents (i.e. functions) were good estimates of users’ real preferences, it is clear, that the best choice would be the one that maximizes the surplus (either individual or total – depending on the purpose of the system) of the transaction. In reality, however, as the system can only ask very few questions, it has very limited user information. Therefore, an agent created by the system can predict user’s behavior only to a certain extent.

The problem of finding the point that maximizes the total surplus involves finding a global maximum of an $n$-dimensional function, such as the difference between buyer’s and seller’s pricing functions. In most of the cases this procedure would return a border solution, i.e. the point that lays on the border of the intersection of the buyer and the seller negotiation subspaces. Such a point, however, might end up being "the best" only on paper, since IVAN estimates a user well only around his “central value” point and
does not give any guarantee on the borders (see Section 2.1, step 2). So it is very possible
that in some cases the system will end up offering a seller to trade his product at a
negative price.

A point selected to be "the best" in IVAN chooses another value criteria: reliability of the estimation. IVAN, as described in Section 2.2 (step 1), chooses the point
that is the closest to the user's actual preferences, among those that have a positive
surplus. This translates to the following formula:

\[
\min f(x) = |x - b_{cv}| + |x - s_{cv}| \\
\text{such that } g(x) = p_b(x) - p_s(x) > 0 \\
\text{where } b_{cv}, s_{cv} \text{ are buyer's and seller's central points and } p_b(x), p_s(x) \text{ are their price functions.}
\]

In most of the cases, unfortunately, IVAN ends up finding a point with almost a
zero surplus. Moreover, the claim that "it is only an estimate and in reality it might be
higher" will not hold here, as the whole purpose of choosing that point was to be as close
to the real user preferences as possible.
The intersections of a buyer and a seller subspaces

\[ \text{surplus} > 0 \]

\text{max surplus point}

\text{the best min distance point}

\text{buyer central value}

\text{seller central value}

(a)

(b)

\text{surplus} > 0

The intersections of a buyer and a seller subspaces

\text{max surplus point}

\text{the best min distance point}

\text{buyer central value}

\text{seller central value}

Figure 14: The best point

In (a) a positive surplus is attained for both parties, as the line between the central values intersects the positive surplus subspace. In (b), however, the surplus is zero.

There is an obvious trade-off between reliability of data (obtained near the central values) and the maximization of surplus (obtained near the border). IVAN chooses the former. But what purpose would possibly serve a zero-surplus product for the user of the system? Following the discussion in section 5.5, IVAN assumes that no one would give his reservation prices correctly. A buyer will always lower it down, and a seller will put it higher. Therefore, both will end up getting a positive profit even in a supposedly zero-surplus point.

While I have provided here a definition of this trade-off problem and suggested a solution, in reality more experiments are needed for further improvements. Here is one easy experiment design framework:

1. Find a point \( x_p \) that maximizes the profit.
2. Take \( a = \text{some percentage of the maximum profit} \).
3. Minimize distance for all points where surplus exceeds \( a \) and obtain a new point \( x_a \).
4. Then the “best value” is the point \( x \in \{x_a, x_p\} \), for which the sum of the distances to the central values of the buyer and the seller is smaller.

5. Now change \( a \) to another level and repeat the previous steps.

This algorithm simply cuts off the surpluses at level \( a\% \), rather than at 0 and attempts to avoid both extremes by choosing, instead, some point in between. Intuitively, it should result, in general, in a better performance.

A more complex and better solution, however, should be smarter and use the actual structures of both \( \text{minDistance} \) and \( \text{maxProfit} \) functions to determine the most valuable point for both users.

### 5.7. The negotiation process

The final price negotiation currently implemented in IVAN is actually quite simple.

When a buyer and a seller get matched, the system calculates a price interval where they can negotiate. By construction, this interval lies within the core (see Section 2.2, step 3), so any price within its boundaries would be an acceptable final choice. Then a seller receives a lower end price of the interval, and a buyer – a higher. This initial setting is quite self-explanatory, as it makes possible for any point within the interval to become a final price.

Next, a buyer and a seller make adjustments to the suggestion made by the system. IVAN in set up, so that they can make only one correction to the price. Finally, the system looks at users’ resulting positions. If their price difference is positive, the final surplus is divided in half, and the price stabilizes in the middle of the amended interval.
Otherwise, if the final price of the seller ends up being greater than that of the buyer, the deal gets cancelled.

The reason behind such a strange negotiation scheme is quite simple. The system operates and estimates users' behaviors in the n-dimensional space and, consequently, is very imprecise. Several real-world experiments have been done on how well such systems can predict users' behavior. A California-based company Perfect.com [28], for example, also performed n-dimensional estimations and was forced to switch its system to the advisory status. Therefore, some user corrections to the system-determined prices are necessary. These corrections also eliminate several problems, associated with advisory-only systems. Since the users have a chance to review and set their price floor/ceiling, a mandatory participation in the final deal, as in Ebay auctions, seems totally natural.

There is no particular reason why the final surplus is split in half, except for the common sense. That is what most of the similar systems do, as there is no point in favoring on type of participants over the other.

In the process of choosing the negotiation scheme for the IVAN project, I have decided to avoid any complex multi-step systems. To put it simple, I do not think that a more complex system would improve customer's satisfaction. So, keeping with a notion “keep it simple, stupid,” I have decided to keep the negotiation part as short and clear as possible.

5.8. The core
The original design of IVAN lacked completely the core-finding part. As it turned out, such a deficiency would have ended up hurting the system. If the final deal prices were systematically revealed to the public, IVAN would have lost its unsatisfied customers. Considering that any marketplace system should be still functional in the event of total openness of its participants, IVAN was accommodated accordingly.

Consider for a moment the following situation. Suppose that there is a market of baseball cards. One seller enters the market with a very rare card and sets his reservation price to $5. Suppose now that there are also three buyers that are interested in this rare card. Their reservation prices are $20, $22, and $24.

![Figure 15: The baseball card market.](image)

In this example, a seller did not get a monopolistic advantage, despite the situation on the market. $Seller_1$ and $buyer_2$ ended up being better off by trading on their own. Here a number in the circle indicates user's reservation price, while a number on the side of the edge – his surplus from the transaction.

The system in this case would match a 5-24 pair, as it is the one with the biggest surplus. Then, as described in the design section 2.2, the buyer and the seller would
adjust their final price preferences. Suppose the new spread was changed to 10-20. Then the system would divide the surplus in half and declare the final price to be 15.

Suppose now that the price was published on the Internet. A seller would then instantly get emails from the other buyers indicating that they would have offered a higher price. A seller then would realize that the system did not provide him with a good deal and will go elsewhere the next time.

What went wrong in the events described above? It seems like the seller and the buyer fairly split the surplus in half. The problem comes from the fact that as a sole seller on the market, the seller should have had an advantage in the free-market economy. Whether it is fair or not, is not the scope of this paper. The reality is that if a seller does not get a satisfactory result in this system, he switches to a more favorable one.

Mathematically speaking, any final price in the system should fall in the set of solutions called the core. Core is a notion in the Game Theory that indicates a set of solutions (in this case, a set of buyer-seller pair prices), in which no set of participants can exit a final arrangement and get a better deal for each other independently.

Now let's look at the deal above. The profit of the seller from the final arrangement is $10, and of the buyers – $0, $0, and $9 (figure 15). Now, for example, seller1 and buyer2 can pull out of the final arrangement and set their price to $19. Then the profit of both would increase. In follows then that the price of $15 is not in the core and should not become a final transaction price.

Consider, instead, that the same 5-24 pair was chosen, but the final price after the negotiations ended up being $23 (figure 16). Then no other buyer would be able to offer a better deal to the seller.
The same seller, and buyer, were matched. The price of $23 fell in the core range $[22, 24]$, so no group would want to pull out. Here a number in the circle indicates user's reservation price, while a number on the side of the edge - his surplus from the transaction.

The complete algorithm of finding a core for the market of $m$ buyers and $n$ sellers trading the same product can be found in section 3.6.

IVAN marketplace system, however, is not as simple. In IVAN, each pair ends up trading its own unique product that is optimized particularly for them. So rarely do the products of any two pairs coincide. Moreover, the example with *the core* described above does not seem to help very much in this case, as it is deals with a single-product market. If the products are different, it is not clear how the product prices affect each other. It would be natural then to convert a current multi-product problem two a one-product one. In the case of IVAN, where all products are interconnected, this task turns out to be very simple.

Suppose IVAN already finished calculation the buyer-seller table of preferred products and prices (sec. 2.2, step 2). As described above, using this table the system then
creates a graph, with buyers and sellers being vertices. Each edge in this graph contains a product specification, a buyer price, and a seller price. Now the system can take any edge and ask all the market participants, what is their price for the product specified on the edge. Since the functions are already calculated, such a lookup takes only a constant time for each vertex. For some users, this product will be out of bounds of their interest. For the rest, the system can compile a new graph and put their reservation prices for the product in question in vertices. Since all the participants in this new graph trade the same product, finding the core in this problem can easily be accomplished. If this core ends up being empty, the system can safely delete the edge in the original graph. Otherwise, it should replace buyer and seller reservation prices at the ends of the original edge with the end values of the core from the complementary problem. Now, any price negotiated by a buyer and seller will automatically be in the core. Similarly, the same operation can be repeated for all other edges in the original graph.

5.9. Continuous vs. discrete matching

As mentioned in Section 2.3, IVAN uses a discrete-type timing to match users on the market. The system is triggered only at certain times, when sufficient number of users have entered the system or when some other criteria has been satisfied. Alternatively, IVAN could have matched buyers and sellers continuously on the market. As soon as a user would have created his agent, he would have immediately gotten matched with another agent on the market.

Both systems have their own advantages and disadvantages, as they serve different goals. A continuous system always has either (1 buyer - n sellers) or (n buyers -
I seller) type of configuration on the market, since it does not allow agents on both sides to accumulate. Only a large corporation that is confident enough that it would get interested participants on the opposite size can run such a system. Oracle, for example, currently uses this type of auctions in its outsourcing department to buy necessary supplies for its company.

Since the purpose of IVAN is to create more efficient general marketplaces and not only the monopoly-type ones, it must be designed to allow for $m$-to-$n$ auctions. For that reason, IVAN utilizes the discrete cycle-driven system.
Chapter 6. Recommendations

Due to a limited time frame allocated for this project (less than 9 months), only a few interesting features were developed and implemented for IVAN. In this section, I describe various ideas that either require more research or simply implementation time.

6.1. Aggregate matching

Currently, the system limits transactions between buyers and sellers to 1-to-1 type relationships, in which each particular buyer can only get matched with one seller and vice versa. An interesting extension to the system would be exploring the possibility of allowing a group matching. As an example, one can imagine a system matching procedure that allows several buyers to acquire together some particular product offered by one seller. Such automatic system-aided buyer coordination would enable group members to save significantly by bidding on a large retailer. In a reverse situation, several small sellers can cooperate together to serve a large-scale client, such as a big corporation.

While the idea is simple, its implementation is rather difficult. In such a group-based bidding, two major problems arise: computational complexity and members' coordination issue.

Each user request is represented in the system with a unique internal agent (sec 2.1). When a cycle is triggered, all active agents on the market try to match with each other (sec 2.2). Each possible (buyer, seller) pair, however, takes up significant system
processing time, as it requires solving a complex optimization problem, such as finding a maximum of a non-linear $n$-dimensional function. For $m$ buyers and $n$ seller the number of these tasks is $O(m\cdot n)$. If the system would also add an agent for each combination of buyers and sellers, as the group matching would require, the number of these tasks would become proportional to the exponents of $m$ and $n$. The only way to get around this computational overhead is to reduce the complexity of an algorithm that finds the optimal deal for each pair of buyers and sellers. That would, however, also reduce the overall “intelligence” and usefulness of the system. Another simple, but effective trick might spread the calculation of these computationally intensive problems over time. While currently the system starts all the calculations the cycle-triggering moment, it might also calculate everything right as the agents get added to the system.

Since the system chooses automatically all the coalition members that participate on both sides of the deal, it is also very important to ensure that all buyers and all sellers would agree on the same final product specifications and price. And since IVAN is ultimately only an advisor and cannot estimate each party's preferences exactly, the user coordination might become a significant issue that should not be overlooked.

6.2. The core

As I described in the section 5.8 above, the core calculation is a very important issue for the fee-market simulation systems and should be perfected as much as possible. The current implementation of IVAN, however, has a lot of shortcomings in this matter. For example, when it determines whether a set of sellers or buyers compete against each other, the system can currently only answer this question if these users are able to
produce the same product. If they do not, IVAN assumes that these users do not affect each other’s price formation procedure in any way and, consequently, can act monopolistically. Basically, the system does not understand how to determine competitiveness, if the products being produced/consumed are either complements or substitutes. Ideally, the system should be able to catch and handle both of these cases.

Consider for a moment the following example. There are several buyers and just two sellers on the market: Toyota and Nissan. Toyota, naturally, can produce only Toyotas and Nissan – only Nissans. Therefore, according to the current IVAN core calculation method (sections 3.6 and 5.8), they do not compete. But clearly this statement is false: a lot of people are indifferent between Toyotas and Nissans.

An alternative solution that takes care of this issue should base the core calculation on individual surpluses that participants would get from each deal, rather than...
on the products being traded. This way, the substitutes issue will automatically be solved. Interestingly, such a solution would also handle the core problem from its definition: that no two parties can exit and get a better payoff.

While this alternative core calculation does not seem very complicated, it was not implemented due to the lack of time. The issue would have been even simpler, had IVAN used only linear functions to represent users' agents on the market. With the current nonlinear design, however, one cannot assume that the surplus is transferable within a \((buyer, seller)\) pair.

As for the complements issue, it is not clear to me yet how to detect and treat these types of products.

6.3. Market chains

Another interesting addition to the system would be market chain automation. As an example, *computer components* and *computers* are two distinct markets. But a buyer in one market ends up being also a seller in the other (see figure 18).
An end user of this market chain is usually interested only in the price of a complete computer. A middleman that assembles computers, however, is interested both in the supply curve of the parts market, as well as in the demand curve on the computer market. If a system automatically completes the task of a product creation tree, the operational costs of a middleman, currently doing all the market research, would reduce significantly. This, in turn, would foster more competition and ultimately bring the end-product prices down, as well.

Such an addition to the system would obviously need an extensive testing and user input to check its usability. Here again a lack of time and means of getting a user input prevented me from developing further this idea.
Bibliography

[35] Fast Polyhedral Adaptive Conjoint Estimation: