Safe Exchange Planner

Tuomas Sandholm and Vincent Ferrandon
{sandholm,vf1}@cs.wustl.edu
Department of Computer Science
Washington University
St. Louis, MO 63130-4899

Abstract

Nondelivery is a major problem in exchanges, especially in electronic commerce: the supplier might not deliver the goods or the demander might not pay. Enforcement is often difficult if the exchange parties are software agents, anonymous, bound by different laws, or if litigation and escrow services are expensive. Recently, a game-theoretic self-enforcing method for carrying out exchanges was presented [7, 8]. The exchange is divided into chunks, and each exchange party delivers its next chunk only after the other party has completed the previous chunk. The chunking and chunk sequencing should be done so that at every point of the exchange, both parties gain more by completing the exchange than by vanishing. This paper operationalizes the theory by presenting a design of a safe exchange planner, which is offered on the web. Exchanges differ based on whether there is one or multiple distinguishable items to exchange, whether there is one or multiple indistinguishable units of each item, whether the items are independent or not in terms of the supplier's cost and the demander's valuation, whether the units are dependent or independent in this sense, and whether the goods are countable or uncountable. For these different settings, we present chunking and chunk sequencing algorithms that provably find safe exchanges that optimize different simplicity criteria. We also present interface designs that minimize the amount of information that the user has to input in each setting.[1]

1 Introduction

Nondelivery is a major problem in exchanges, especially in electronic commerce: the supplier might not deliver the goods or the demander might not pay. A recent study shows that 6% of consumers with on-line shopping experience reported products or services that were paid for, but never received [3]. The problem is exacerbated by the trend that electronic markets act only as information services that match buyers and sellers, while leaving the execution of the transactions up to the contract parties. Even if attempted, enforcement of exchanges is often difficult if the exchange parties are software agents (an agent can vanish easily by killing its process and its connection to the real-world party that it represented can be hard to trace), anonymous, bound by different laws of different countries, or if litigation and escrow services are expensive. For example, current electronic commerce escrow companies, such as i-Escrow, Inc., and Trade-Direct, Inc., charge 5%-6% of the contract price as an escrow fee.

Most work on automated negotiation has used some form of *ex ante* rationality as the agents' decision criterion [5, 6]. This assumes that the deals that the agents make are enforced. If possible, it would be desirable to also require *interim* rationality, i.e., that the contracts are self-enforcing [1, 10, 2].

Recently, a game-theoretic self-enforcing method for carrying out exchanges was presented [7, 8]. The exchange is divided into chunks, and each exchange party delivers its next chunk only after the other party has completed the previous chunk. The method mainly targets setting where only negligible cost is incurred by dividing and delivering the goods and payments in chunks. This is the case, for example, in exchanging data (software, books, music, video-on-demand, etc.) or computational services (e.g., Disney subcontracting to a rendering farm, or idle CPU cycles being traded among anonymous parties [4]). Unfortunately, the technique may also facilitate illegal trades of drugs, arms, porn, etc., where enforcement of the exchange is not possible via litigation.

In order not to require the user to plan and execute the exchange strategy of delivering one chunk at a time, we suggest that she use an exchange manager agent. The user inputs information about the exchange setting to her agent. The agent then plans a safe exchange, and carries it out so the user does not even have to know that the exchange is carried out in chunks instead of in one step. The agent should do the chunking and chunk sequencing so that at every point of the exchange, both exchange parties gain more by completing the exchange than by vanishing. This paper operationalizes the theory by presenting a design and implementation of such a safe exchange planner. For a variety of exchange types, we present chunking and chunk sequencing algorithms that provably find safe exchanges that optimize different simplicity criteria. We also present interface designs that minimize the amount of information that the user has to input to her agent in each exchange type.

2 Our model of an exchange

In our model, the exchange occurs between two agents: a supplier and a demander. The supplier is delivering n distinguishable items ($n \in \mathbf{Z}^+$), with λ_i, $i \in \{1..n\}$, units per item. The units of any given item are indistinguishable from each other. We denote a state of delivery by $x = (x_1, \ldots, x_n)$. The initial delivery state is $(0, \ldots, 0)$, and a completed delivery is $(\lambda_1, \ldots, \lambda_n)$. For countable goods (such as software licenses, stocks, and barrels of oil), the set of delivery states is $S = \{0, 1, \ldots, \lambda_1\} \times \{0, 1, \ldots, \lambda_2\} \times \ldots \times \{0, 1, \ldots, \lambda_n\}$ and

[1] Supported by NSF under CAREER Award IRI-9703122, Grant IRI-9610122, and Grant IIS-9800994.

for uncountable goods (such as consulting time, mutual funds, and electricity), it is $S = [0, \lambda_1] \times [0, \lambda_2] \times \ldots \times [0, \lambda_n]$. The demander is paying for these goods. Initially the payment is 0, the complete payment of the contract is p^{contr}, and the state of payment is $p \in [0, p^{contr}]$. Put together, the state of the exchange is $(x_1, ..., x_n, p)$.

Whenever the exchange ends—possibly prematurely because an agent defected by vanishing—the supplier's utility is $u_s(x,p) = p - v_s(x)$ if she did not defect and $u_s(x,p) = p - v_s(x) - c_s^{def}$ if she did. The nondecreasing function $v_s(x) \in \Re^+$ is the supplier's cost of generating and delivering the goods x.[2] The constant $c_s^{def} \geq 0$ is the supplier's exogenous disutility of defecting. Such disutility can stem from inherent honesty, risk of being litigated, reputation effects that may lead to loss of future business, etc.

The demander's utility is $u_d(x,p) = v_d(x) - p$ if she did not defect and $u_s(x,p) = v_d(x) - p - c_d^{def}$ if she did. The nondecreasing function $v_d(x) \in \Re^+$ is the demander's valuation of the goods x, and $c_d^{def} \geq 0$ is the demander's exogenous disutility of defecting.

We want to ensure that at any state of the exchange, (x,p), that is reached, the supplier's utility from completing the exchange (assuming that the demander does not defect) is no less than his utility from defecting:

$$p^{contr} - v_s(\lambda_1, \ldots, \lambda_n) \geq p - v_s(x) - c_s^{def}$$

So, we want to maintain $p \leq p^{max}(x)$ throughout the exchange, where

$$p^{max}(x) \stackrel{\text{def}}{=} p^{contr} - v_s(\lambda_1, \ldots, \lambda_n) + v_s(x) + c_s^{def} \quad (1)$$

We also want to ensure that at any state of the exchange that is reached, the demander's utility from completing the exchange (assuming that the supplier does not defect) is no less than his utility from defecting:[3]

$$v_d(\lambda_1, \ldots, \lambda_n) - p^{contr} \geq v_d(x) - p - c_d^{def}$$

So, we want to maintain $p \geq p^{min}(x)$ throughout the exchange, where

$$p^{min}(x) \stackrel{\text{def}}{=} p^{contr} - v_d(\lambda_1, \ldots, \lambda_n) + v_d(x) - c_d^{def} \quad (2)$$

It was recently shown that the exchange strategies S_s and S_d (defined below) for the agents—where each agent sends the other as much as it can while honoring (1) and (2)—lead to completion of the exchange in subgame perfect Nash equilibrium (SPNE) if for every two consecutive delivery states[4]

$$x, x' \in S, \quad p^{max}(x) \geq p^{min}(x') \quad (3)$$

(otherwise no exchange strategies can proceed safely without violating (1) or (2)) [7, 8]. We call delivery states x and x' *consecutive* if $\exists k \in [1,n]$ such that $x'_k = x_k + \varepsilon$ and $x'_l = x_l$ for $l \neq k$, where $\varepsilon = 1$ if the good k is countable, and $\varepsilon \geq \epsilon$ if the good k is uncountable (ϵ is a positive constant).

Say that the delivery order of the different units of the different items has been fixed (we will present algorithms for determining a good order later). Then the state of the exchange, $(x_1, \ldots, x_n, p)$, can be captured uniquely by (ξ, p) where $\xi = \sum_{i=1}^n x_i \in X$, and $X = \{0, \ldots, \sum_{i=1}^n \lambda_i\}$ for countable goods and $X = [0, \sum_{i=1}^n \lambda_i]$ for uncountables. Therefore, with a slight abuse of notation, we allow $p^{max}(\cdot)$ and $p^{min}(\cdot)$ to be called with either an argument x or an argument ξ. Now we are ready to state the exchange strategies:

Definition. 2.1 (Supplier's strategy S_s) *At any point (ξ^t, p^t) of the exchange, if $p^{min}(\xi^t) \leq p^t \leq p^{max}(\xi^t)$, deliver an amount such that cumulative delivery $\xi^{t+1} = \max\{\xi \in X : p^{min}(\xi) \leq p^t\}$.[5] Else exit.*

Definition. 2.2 (Demander's strategy S_d) *At any point (ξ^t, p^t) of the exchange, if $p^{min}(\xi^t) \leq p^t \leq p^{max}(\xi^t)$, pay an amount such that cumulative payment $p^{t+1} = p^{max}(\xi^t)$. Else exit.*

It was previously shown that for any given order of delivering the different units of different items, no other strategies can lead to completion of the exchange in SPNE in fewer steps (a step can include both a delivery and a payment) than strategies S_s and S_d do [7]. We now show that, alternatively, with slight modifications to the strategies, the exchange can be completed in SPNE in the minimal number of payments, minimal number of deliveries, or minimal sum of payments and deliveries. To minimize the number of payments, we simply try to force the first step of the exchange to include a delivery only. The exchange will then alternate between payments and deliveries. To minimize the number of deliveries, we simply try to force the first step of the exchange to include a payment only. The exchange will then alternate between deliveries and payments. To minimize the sum of payments and deliveries, we try both ways of forcing and see which leads to the smaller sum.

Theorem 2.1 *Assume that the exchange can be completed in SPNE using the given order of delivering the*

[2] We assume that this cost (and any other input to the safe exchange planner) does not depend on the number of chunks into which the exchange is split.

[3] If the agents do not know each other's value functions, they can use bounds they know to compute $p^{max}(x)$ and $p^{min}(x)$. The supplier is safe using an upper bound for $p^{min}(x)$, i.e. a lower bound for $v_d(\lambda_1, \ldots, \lambda_n)$ and an upper bound for $v_d(x)$. The demander is safe using a lower bound for $p^{max}(x)$. Although the agents are safe using such bounds, even possible exchanges are disabled if the bounds are too far off.

[4] Two strategies are in SPNE if neither agent wants to deviate from its strategy in any subgame of the game tree, given that the other party does not deviate [9].

[5] For uncountable goods, this max may not be defined, so we use $max(\xi^t, sup\{\xi \in X : p^{min}(\xi) \leq p^t\} - \underline{\epsilon})$ instead, where $\underline{\epsilon} < \epsilon$.

units of the different items. If at $(\xi^0, p^0) = (0,0)$, the agents apply strategies that immediately lead to $(\xi^1, p^1) = (\max\{\xi \in X : p^{min}(\xi) \leq p^0\}, p^0)$, and S_s and S_d from then on, the exchange is completed in SPNE in the minimal number of payments. If at $(\xi^0, p^0) = (0,0)$, the agents apply strategies that immediately lead to $(\xi^1, p^1) = (\xi^0, p^{max}(\xi^0))$, the exchange is completed in SPNE in the minimal number of deliveries. To complete the exchange in SPNE with a minimal sum of payments and deliveries, it suffices to try both of these variants and select the one that leads to the smaller sum.

We call the items of an exchange *independent* if $v_s(x_1, x_2, \ldots, x_n) = \sum_{i \in \{1..n\}} v_s^i(x_i)$ and $v_d(x_1, x_2, \ldots, x_n) = \sum_{i \in \{1..n\}} v_d^i(x_i)$. Otherwise the items are *dependent.* We call the units of items *independent* if $\forall i \in \{1..n\}$, $v_s(x_1, \ldots, x_i, \ldots, x_n) = v_s(x_1, \ldots, 0, \ldots, x_n) + x_i \cdot \frac{\Delta v_s}{\Delta x_i}$ and $v_d(x_1, \ldots, x_i, \ldots, x_n) = v_d(x_1, \ldots, 0, \ldots, x_n) + x_i \cdot \frac{\Delta v_d}{\Delta x_i}$, where $\frac{\Delta v_s}{\Delta x_i}$ and $\frac{\Delta v_d}{\Delta x_i}$ are constants that depend only on i. Otherwise the units are *dependent.*

We call an exchange *safe* if it can be completed in SPNE. Exchanges differ based on whether there is one or multiple items to exchange, whether there is one or multiple units of each item, whether the items are dependent or independent, whether the units are dependent or independent, and whether the goods are countable or uncountable. In the rest of the paper, we present chunking and chunk sequencing algorithms for the different settings. We also present interface designs that minimize the amount of information that the user has to input in each setting. For every exchange type, the user starts by inputting the contract price, p^{contr}, and the defection disutilities, c_s^{def} and c_d^{def}. However, the rest of the input depends on the exchange type. Section 3 covers independent items and independent units. Section 4 handles independent items and dependent units. Section 5 covers dependent items and dependent units. Section 6 handles dependent items and independent units. Finally, Section 7 presents conclusions.

3 Independent item(s) with independent unit(s)

In this section we discuss an exchange involving one or many independent items, where the units of each item are independent. If there is only one item to be delivered ($n = 1$), we simply need to decide, for every step of the exchange, how many units of that item are to be delivered and how much is to be paid, i.e., we need to decide the chunking of the exchange. In the case of multiple items, we first need to decide the order in which the items are to be delivered, and then decide, for every step of the exchange, how many units of that item are to be delivered and how much is to be paid, i.e., the chunking. Since both settings require chunking, we present our chunking algorithm in the next subsection. The second subsection covers the setting with a single item. The third subsection discusses multiple items.

3.1 Chunking algorithm

Say that there are λ units to be delivered. In the setting with one item, these are all units of the same item ($\lambda = \lambda_1$). In the case of multiple items, these are sequenced units of different items ($\lambda = \sum_{i=1}^{n} \lambda_i$). Since the units are sequenced before the algorithm is executed, the state of the exchange can be compressed to be (ξ, p) instead of $(x_1, ..., x_n, p)$, where $\xi = \sum_{i=1}^{n} x_i$.

Here p^{max} is represented by line segments $\{ls_j^{max}\}_{j \in \{1..r_{max}\}}$, and p^{min} is represented by line segments $\{ls_i^{min}\}_{i \in \{1..r_{min}\}}$. We also define

$$\alpha(ls_j^{max}) = \xi\text{-coordinate of the left tip of } ls_j^{max}$$
$$\omega(ls_j^{max}) = \xi\text{-coordinate of the right tip of } ls_j^{max}$$

and $\forall \xi \in [\alpha(ls_j^{max}), \omega(ls_j^{max})]$ (or $\{\alpha(ls_j^{max}) \ldots \omega(ls_j^{max})\}$ in the case of countable goods), $ls_j^{max}(\xi) = p$, and $(ls_j^{max})^{-1}(p) = \xi$. Furthermore, $\forall j, \omega(ls_j^{max}) = \alpha(ls_{j+1}^{max})$ (analogously for ls_i^{min}).

Let $COUNTABLE$ be a Boolean vector of length r_{min} so that $\forall i \in \{1..r_{min}\}$, $COUNTABLE[i]$ is true if the goods on the interval ls_i^{min} are countable, and false if they are uncountable. Finally, let $F(\xi, i) = \xi$ if $COUNTABLE[i] = false$, and $F(\xi, i) = \lfloor \xi \rfloor$ if $COUNTABLE[i] = true$.

If no exchange can proceed beyond line segment ls_i^{min}, and the good involved in that segment is countable, the algorithm below stops and reports "NO SOLUTION". However, if the good involved in that segment is uncountable, it may be that the exchange keeps approaching some delivery amount with an infinite sequence of progressively decreasing steps. In that case, the algorithm below would never stop. Therefore, before running the algorithm below, a separate check is executed for every line segment that involves an uncountable good. Specifically, $\forall i$ s.t. $COUNTABLE[i] = false$, we run an algorithm that looks for ξ and j such that $ls_j^{max}(\xi) = ls_i^{min}(\xi)$ with $\alpha(ls_j^{max}) < \xi$. A safe exchange is impossible if $ls_j^{max}(\alpha(ls_j^{max})) < ls_j^{max}(\xi)$ or if $ls_i^{min}(\omega(ls_i^{min})) > ls_i^{min}(\xi)$. In that case, execution of the algorithm below is not attempted.

Algorithm 3.1 DETERMINE-CHUNKING
$(\{ls_i^{max}\}_{i \in \{1, r_{max}\}}, \{ls_i^{min}\}_{i \in \{1, r_{min}\}}, COUNTABLE[])$

1. $\xi^0 = 0$; $p^0 = 0$; $i = 1$; $j = 1$ // initialize
2. while ($i \leq r_{min}$ AND $ls_i^{min}(\omega(ls_i^{min})) < p^0$) do
   $i = i + 1$
3. if $i > r_{min}$ then $\xi^1 = \lambda$ else $\xi^1 = F((ls_i^{min})^{-1}(0), i)$
4. while ($\omega(ls_j^{max}) < 0$) do
   $j = j + 1$ // here $j \leq r_{max}$ always
5. $p^1 = min(ls_j^{max}(0), p^{contr})$
6. $t = 2$
7. while ($\xi^{t-1} < \lambda$ OR $p^{t-1} < p^{contr}$) do // main loop
   while ($i \leq r_{min}$ AND $ls_i^{min}(\omega(ls_i^{min})) < p^{t-1}$) do
   $i = i + 1$

if $i > r_{min}$ then $\xi^t = \lambda$
else $\xi^t = F((ls_i^{min})^{-1}(p^{t-1}), i)$
while $(\omega(ls_j^{max})) < \xi^{t-1})$ do
$j = j+1$
$p^t = min(ls_j^{max}(\xi^{t-1}), p^{contr})$
if $COUNTABLE[i] = true$ AND $(\xi^t = \xi^{t-1} = \xi^{t-2})$, return "NO SOLUTION"
$t = t+1$

8. return $(\xi^1, \xi^2, \ldots, \xi^\tau)$ and $(p^1, p^2, \ldots, p^\tau)$

Theorem 3.1 *For a given order of delivering the units, for countable or uncountable goods, for nondecreasing p^{max} and p^{min}, DETERMINE-CHUNKING finds a safe exchange plan that minimizes the number of exchange steps. By forcing the exchange to start with a payment only (by setting $\xi^1 = 0$ in step 3), the algorithm finds a safe exchange with minimal number of deliveries. By forcing the exchange to start with a delivery only (by setting $p^1 = 0$ in step 5) the algorithm finds a safe exchange with a minimal number of payments. To pick a safe exchange that minimizes the sum of deliveries and payments, pick that solution given by the two variants of the algorithm that minimizes the sum. For all of these variants, the algorithm runs in $O(\tau \max(r_{min}, r_{max}))$ time and $O(\max(\tau, r_{min}, r_{max}))$ space, where τ is the number of chunks in the output.*

3.2 Single item, independent unit(s)

Exchanges of a single item with independent units (countable or uncountable) are the simplest. The input interface is a table where the user fills two cells (Table 1). Since the units are independent, $\frac{\Delta v_s(x)}{\Delta x_1}$ and $\frac{\Delta v_d(x)}{\Delta x_1}$ do not depend on x_1 by definition. Thus p^{max} and p^{min} are lines. The optimal safe exchange is computed by running DETERMINE-CHUNKING on this input.

item name	supplier's cost per unit	demander's value per unit
item	$\frac{\Delta v_s(x)}{\Delta x_1}$	$\frac{\Delta v_d(x)}{\Delta x_1}$

Table 1: *Input form for a single independent item with independent units.*

3.3 Many independent items, independent units

For exchanges with multiple independent items (countable or uncountable) with independent units, the user enters how much one unit of each item increases v_s and v_d (Table 2). Therefore, we have the following line segments (the quantity before the comma is the increase in x_i, the one after the comma is the increase in p): $\{(\lambda_i, \lambda_i \frac{\Delta v_s}{\Delta x_i})\}_{i \in \{1..n\}}$ and $\{(\lambda_i, \lambda_i \frac{\Delta v_d}{\Delta x_i})\}_{i \in \{1..n\}}$. To simplify the notation, define $\{(\lambda_i, \Delta p_i^{max})\}_{i \in \{1..n\}} = \{(\lambda_i, \lambda_i \frac{\Delta v_s}{\Delta x_i})\}_{i \in \{1..n\}}$ and $\{(\lambda_i, \Delta p_i^{min})\}_{i \in \{1..n\}} = \{(\lambda_i, \lambda_i \frac{\Delta v_d}{\Delta x_i})\}_{i \in \{1..n\}}$. For both countable and uncountable goods, the algorithm below finds a safe ordering of $\{(\lambda_i, \Delta p_i^{max})\}_{i \in \{1..n\}}$ (resp. p^{min}) if one exists.

item name	supplier's cost per unit	demander's value per unit
$item_1$	$\frac{\Delta v_s}{\Delta x_1}$	$\frac{\Delta v_d}{\Delta x_1}$
$item_2$	$\frac{\Delta v_s}{\Delta x_2}$	$\frac{\Delta v_d}{\Delta x_2}$
...	...	...
$item_n$	$\frac{\Delta v_s}{\Delta x_n}$	$\frac{\Delta v_d}{\Delta x_n}$

Table 2: *Input form for independent items, independent units.*

Algorithm 3.2
SEQUENCE-SEGMENTS($\{(\lambda_i, \Delta p_i^{max})\}_{i \in \{1..n\}}$, $\{(\lambda_i, \Delta p_i^{min})\}_{i \in \{1..n\}}$, p^{contr}, c_s^{def}, c_d^{def}, *COUNTABLE[]* **)**

1. $p_{init}^{max} = p^{contr} + c_s^{def}$; $p_{init}^{min} = p^{contr} - c_d^{def}$
2. $\forall i \in \{1..n\}$ do /* Sets bounds for p at $x = 0$ */
$p_{init}^{max} = p_{init}^{max} - \Delta p_i^{max}$; $p_{init}^{min} = p_{init}^{min} - \Delta p_i^{min}$
3. if $p_{init}^{max} < 0$ OR $p_{init}^{min} > 0$, return "NO SOLUTION"
4. Divide $\{1..n\}$ into two sets POS and NEG s.t. $POS = \{i \in \{1..n\} : \Delta p_i^{max} - \Delta p_i^{min} \geq 0\}$ and $NEG = \{i \in \{1..n\} : \Delta p_i^{max} - \Delta p_i^{min} < 0\}$
5. $p^{max} = p_{init}^{max}$; $p^{min} = p_{init}^{min}$; $n_p = |POS|$; $n_n = |NEG|$
6. for $t = 1$ to n_p
$FEASIBLES = \{i \in POS : (COUNTABLE[i] = true$ AND $p^{min} + \frac{\Delta p_i^{min}}{\lambda_i} \leq p^{max})$ OR $(COUNTABLE[i] = false$ AND $(p^{max} > p^{min}$ OR $\Delta p_i^{min} = 0))\}$
if $FEASIBLES = \emptyset$, return "NO SOLUTION"
$i^* = \arg\max_{i \in FEASIBLES} \Delta p_i^{max} - \Delta p_i^{min}$
$segment[t] = i^*$
$p^{max} = p^{max} + \Delta p_{i^*}^{max}$; $p^{min} = p^{min} + \Delta p_{i^*}^{min}$.
$POS = POS - \{i^*\}$
7. $p^{max} = p^{contr} + c_s^{def}$; $p^{min} = p^{contr} - c_d^{def}$
8. for $t = n_n + n_p$ down to $n_p + 1$
$FEASIBLES = \{i \in NEG : (COUNTABLE[i] = true$ AND $p^{min} \leq p^{max} - \frac{\Delta p_i^{max}}{\lambda_i})$ OR $(COUNTABLE[i] = false$ AND $(p^{max} > p^{min}$ OR $\Delta p_i^{max} = 0))\}$
if $FEASIBLES = \emptyset$, return "NO SOLUTION"
$i^* = \arg\max_{i \in FEASIBLES} \Delta p_i^{min} - \Delta p_i^{max}$
$segment[t] = i^*$
$p^{max} = p^{max} - \Delta p_{i^*}^{max}$; $p^{min} = p^{min} - \Delta p_{i^*}^{min}$
$NEG = NEG - \{i^*\}$
9. Return the vector "*segment*". First segment to be placed is in $segment[1]$.

Theorem 3.2 *For the case of countable goods and for the case of uncountable goods, SEQUENCE-SEGMENTS finds a safe ordering if one exists, and returns "NO SOLUTION" otherwise. It always terminates in $O(n^2)$ time.*[6]

[6] SEQUENCE-SEGMENTS is an improvement over the SEQ-CHUNKS algorithm (presented earlier [7, 8]). First, it is quadratic in the number of line segments instead of in the

SEQUENCE-SEGMENTS determines an ordering for the items. All units of one item are delivered before delivering any units of the next (this does not compromise safety or minimality of the exchange plan). After SEQUENCE-SEGMENTS has determined an ordering, we run DETERMINE-CHUNKING to determine the chunking. It outputs the exchange plan.

3.4 Example: Purchasing stocks

In this example the agents exchange multiple independent units of multiple independent items (stocks: MSFT, HWP, DELL, EBAY, and PG). The stocks have different values for the supplier and the demander (e.g., because the supplier and the demander have different portfolios, and the value of a stock in a portfolio depends on how correlated the stock is with the rest of the portfolio statistically). Note that v_s can be greater than v_d for some items—but not all of them—to allow safe exchange. Say the agents have agreed to a contract price \$25864.60, and the defection disutilities are $c_s^{def} = \$1000$ and $c_d^{def} = \$1500$. The input to the algorithm and the output are shown in Figure 1.

4 Independent item(s), dependent units

For exchanges of dependent units of a single item, the user inputs the number of units to be exchanged, λ_1. The user also inputs $v_s(x)$ and $v_d(x)$. In our implementation, the user draws piecewise linear, nondecreasing functions $v_s(x)$ and $v_d(x)$. These define piecewise linear, nondecreasing functions $p^{max}(x)$ and $p^{min}(x)$ according to Equations (1) and (2). These pieces are then fed to DETERMINE-CHUNKING which plans a minimal safe exchange if one exists.

For exchanges of dependent units of multiple independent items, the user inputs the number of units to be exchanged for each item, $\lambda_i, i \in \{1..n\}$. The user also draws piecewise linear, nondecreasing functions $v_{s,i}(x_i)$ and $v_{d,i}(x_i)$ for all $i \in \{1..n\}$. For countable goods, these value function can be aggregated: $v_s(x_1, \ldots, x_n) = \sum_{i=1}^{n} v_{s,i}(x_i)$ and $v_d(x_1, \ldots, x_n) = \sum_{i=1}^{n} v_{d,i}(x_i)$. This information (which does not capitalize on the fact that the items are independent) is then fed into the algorithm that plans a minimal safe exchange with multiple dependent items and dependent units (CHUNK&SEQUENCE, described in Section 5).[7]

item name	number of units to exchange	$\frac{\Delta v_s}{\Delta x_i}$	$\frac{\Delta v_d}{\Delta x_i}$
MSFT	60	\$90.81	\$98
HWP	30	\$102.63	\$109.9
DELL	10	\$41.81	\$42.5
EBAY	100	\$120.25	\$127.25
PG	40	\$85.1	\$88.44

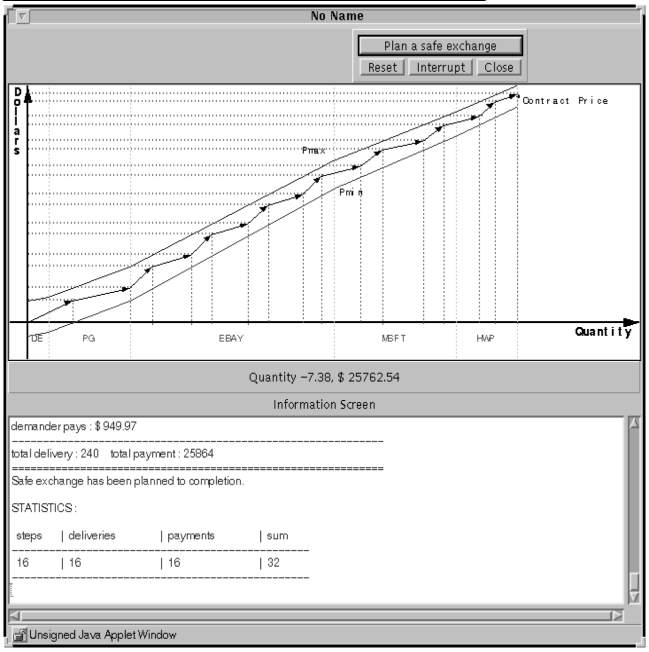

Figure 1: **Top:** *Input form for independent items, independent units.* **Bottom:** *Output. This optimal plan is 16 steps long. The items (DELL, PG, EBAY, MSFT, HWP) were sequenced by SEQUENCE-SEGMENTS and then the chunking was determined using DETERMINE-CHUNKING.*

5 Many dependent items, dependent units

Exchanges of several dependent items with dependent units are the most complex class of exchanges. In this section we describe a fully expressive input method, and an algorithm that is guaranteed to find a safe optimal exchange in this setting.

5.1 Table for user to input v_s and v_d

If the number of items is greater than two, v_s and v_d can no longer be represented graphically. However, the user can input them in table form as long as the units are countable. In the rest of this section we therefore restrict our discussion to countable goods. In the input table, the user fills in a value for v_s and for v_d for each possible $(x_1, x_2, \ldots, x_n) \in \{0, 1, \ldots, \lambda_1\} \times \{0, 1, \ldots, \lambda_2\} \times \ldots \times \{0, 1, \ldots, \lambda_n\}$. This means filling $2 \cdot \prod_{i=1}^{n}(\lambda_i + 1)$ cells in a table.

Example A. Consider a software system consisting of two packages, a main package and a plug-in. The former takes four floppy disks (units) whereas the latter takes only one. They depend on each other in the sense of

number of units to be exchanged. This improves the speed significantly in settings where segments are multiple units long. Second, in the case of uncountable goods, the number of smallest possible deliverables is infinite, so SEQ-CHUNKS cannot handle that case, while SEQUENCE-SEGMENTS works in that setting as well.

[7]For uncountable goods this aggregation would result in a table with an uncountably infinite number of rows. Therefore, a different method is required. In the safe exchange planner implementation that we offer on the web, we use a fast dynamic programming algorithm for this case (it also handles countable goods). It capitalizes on the independence of the items. However, we omit the algorithm due to space limitations, and due to the fact that we have not yet proven that the algorithm finds a safe exchange every time one exists.

the value functions v_s and v_d, as the user has expressed this in the following input table:

plug-in	package	v_s	v_d
0	0	0	0
0	1	5	3
0	2	9	6
0	3	13	10
0	4	17	14
1	0	10	6
1	1	13	10
1	2	16	15
1	3	18	20
1	4	19	26

Table 3: *Single table input form (STIF) for example A.*

If $v_s(x') = v_s(x)$ for $x, x' \in S$ such that $\forall i \in \{1..n\}, x' \geq x$, and $x' > x$ for some $i \in \{1..n\}$, then the user need not enter a value in the cell for $v_s(x')$ (analogously for $v_d(\cdot)$). The following simple algorithm fills these missing values automatically:

$$v_s(x_1, x_2, \ldots, x_n) = \max_{i \in [1,n]} v_s^i$$

where $v_s^i = v_s(x_1, x_2, \ldots, x_{i-1}, x_i - 1, x_{i+1}, \ldots, x_n)$.

Before we run the safe exchange planning algorithm, we check that the input satisfies our assumption that v_s and v_d are nondecreasing. For dependent items with dependent units, we check this using a local notion of nondecreasing function:

$$\forall i \in \{1, n\}, \quad v_s(x_1, x_2, \ldots, x_n) \geq v_s^i$$

5.2 Algorithm for planning an optimal safe exchange

As before, let S be the set of possible states of delivery: $S = \{0, 1, \ldots, \lambda_1\} \times \{0, 1, \ldots, \lambda_2\} \times \ldots \times \{0, 1, \ldots, \lambda_n\}$. So, any vector $x \in S$ is a delivery state $x = (x_1, x_2, \ldots, x_n)$. Let $r(x)$ be $true$ if x can be reached safely, and $false$ otherwise. Let $\eta(x)$ be the number of steps needed to reach x. Let $c(x) = (y^1, y^2, \ldots)$ be the minimal ordered list of delivery states, $y^i \in S$, to reach x such that the exchange can safely move from one of these delivery states to the next in a single step, i.e., for any $y^i, y^{i+1} \in c(x)$, $p^{max}(y^i) \geq p^{min}(y^{i+1})$. Let $d(x) = (p^1, p^2, \ldots)$ be the ordered list of safe payment states, $p^i \in \Re^+$, corresponding to the delivery states in $c(x)$.

Let $x^i = (x_1, x_2, \ldots, x_{i-1}, x_i - 1, x_{i+1}, \ldots, x_n)$. Let $r^i(x)$ be $true$ if x can be reached safely via x^i, and $false$ otherwise. Let $\eta^i(x)$ be the number of chunks used to reach x via x^i. Let $c^i(x) = (y^1, y^2, \ldots)$ be the minimal ordered list of delivery states to reach x via x^i such that the exchange can safely move from one of these delivery states to the next in a single step. Let $d^i(x) = (p^1, p^2, \ldots)$ be the ordered list of safe payment states, $p^i \in \Re^+$, corresponding to the delivery states in $c^i(x)$.

Figure 2 visualizes the problem instance of example A. There are four cases to consider when reaching one vertex from another, e.g. $(1,1)$ from $(0,1)$:

- $(1,1)$ can be reached safely via $(0,1)$ in the same step that reached $(0,1)$. So, $(1,1)$ replaces $(0,1)$ as the last element of $c(1,1)$.

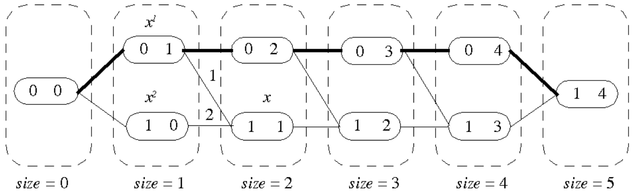

Figure 2: *Graph representation of the delivery states, $x = (x_1, x_2)$, in example A. The states are grouped according to size $= \sum_{i=1}^{n} x_i$.*

- $(1,1)$ can be reached safely in one step via $(0,1)$, but not in the same step that reached $(0,1)$. So, $(1,1)$ is appended to $c(1,1)$.
- $(1,1)$ can be reached safely via $(0,1)$ in two steps (one is a payment and the other is a delivery), neither of which is the same as the step that reached $(0,1)$. So, $(0,1)$ is appended to $c(1,1)$, and then $(1,1)$ is appended to $c(1,1)$.
- $(1,1)$ cannot be reached safely via $(0,1)$.

The dynamic programming algorithm below is based on considering these cases. The functions $last(\cdot)$, $last_{-1}(\cdot)$, give the last and the second to last element of a list respectively. The operators $\oplus$ and $\ominus$ add and subtract an element at the end of an ordered list.

Algorithm 5.1 CHUNK&SEQUENCE(n, p^{contr}, $p^{max}(\cdot)$, $p^{min}(\cdot)$, $\lambda_1, \lambda_2, \ldots, \lambda_n$)

1. if $p^{max}(0, \ldots, 0) < 0$ OR $p^{min}(0, \ldots, 0) > 0$ OR $p^{max}(\lambda_1, \ldots, \lambda_n) < p^{contr}$ OR $p^{min}(\lambda_1, \ldots, \lambda_n) > p^{contr}$, return "NO SOLUTION"

2. $r(0, 0, \ldots, 0) = true$
$\eta(0, 0, \ldots, 0) = 1$
$c(0, 0, \ldots, 0) = \{(0, 0, \ldots, 0)\}$
$d(0, 0, \ldots, 0) = \{0\}$

3. for $size = 1$ to $\sum_{i=1}^{n} \lambda_i$ do
  a. $THIS\_SIZE\_REACHABLE = false$
  b. for $x \in \{y \in S : \sum_{k=1}^{n} y_k = size\}$ do
    1. for $i \in [1, n]$ do
      i. if $r(x^i) = true$ then {//try the 3 possibilities
      if $|d(x^i)| \geq 2$ AND $p^{min}(x) \leq last_{-1}(d(x^i))$
      then // extend current chunk
        $r^i(x) = true$
        $\eta^i(x) = \eta(x^i)$
        $c^i(x) = c(x^i) \ominus last(c(x^i)) \oplus x$
        $d^i(x) = d(x^i)$
      elseif $p^{min}(x) \leq last(d(x^i))$ then
      // pay and deliver in a new chunk
        $r^i(x) = true$
        $\eta^i(x) = \eta(x^i) + 1$
        $c^i(x) = c(x^i) \oplus x$
        $d^i(x) =$
        $d(x^i) \oplus \min(p^{max}(x^i), p^{contr})$
      elseif $p^{min}(x) \leq \min(p^{max}(x^i), p^{contr})$
      then // pay, then deliver (2 chunks)
        $r^i(x) = true$
        $\eta^i(x) = \eta(x^i) + 2$

$$c^i(x) = c(x^i) \oplus x^i \oplus x$$
$$d^i(x) = d(x^i) \oplus \min(p^{max}(x^i), p^{contr}) \oplus \min(p^{max}(x^i), p^{contr})$$
else $r^i(x) = false$
}

ii. else $r^i(x) = false$

2. if $\exists i \in \{1..n\}$ such that $r^i(x) = true$ then
// find the minimal path
$r(x) = true$
$THIS\_SIZE\_REACHABLE = true$
$\eta(x) = \min_{i\in\{1..n\}} \eta^i(x)$
$c(x) = c^{\arg\min_{i\in\{1..n\}}(\eta^i(x))}(x)$
$d(x) = d^{\arg\min_{i\in\{1..n\}}(\eta^i(x))}(x)$

3. else $r(x) = false$

c. if $THIS\_SIZE\_REACHABLE = false$, return "NO SOLUTION"

4. if $last(d(\lambda_1, \lambda_2, \dots, \lambda_n)) < p^{contr}$ then
$c(\lambda_1, \lambda_2, \dots, \lambda_n) = c(\lambda_1, \lambda_2, \dots, \lambda_n) \oplus (\lambda_1, \lambda_2, \dots, \lambda_n)$
$d(\lambda_1, \lambda_2, \dots, \lambda_n) = d(\lambda_1, \lambda_2, \dots, \lambda_n) \oplus p^{contr}$

5. return $c(\lambda_1, \lambda_2, \dots, \lambda_n)$ and $d(\lambda_1, \lambda_2, \dots, \lambda_n)$.

Theorem 5.1 *Assume that a safe sequence exists. CHUNK&SEQUENCE finds a safe sequence that minimizes the number of steps. By changing the initialization to $d(0, 0, \dots, 0) = \{p^{min}(0, 0, \dots, 0) - \epsilon\}$, for any $\epsilon > 0$, CHUNK&SEQUENCE finds a safe sequence that minimizes the number of deliveries. By setting $p^{max}(0, 0, \dots, 0) = 0$ within the algorithm, CHUNK&SEQUENCE finds a safe sequence that minimizes the number of payments. To minimize the sum of payments and deliveries, pick that solution given by these two algorithm variants that minimizes the sum. If a safe sequence does not exist, all of these variants report "NO SOLUTION". The algorithm runs in $O((\sum_{i=1}^{n} \lambda_i)|S|n) = O((\sum_{i=1}^{n} \lambda_i)(\prod_{i=1}^{n}(\lambda_i + 1))n)$ time and $O(S) = O(\prod_{i=1}^{n}(\lambda_i + 1))$ space.*

The time and space are polynomial in the size of the input ($2 \cdot \prod_{i=1}^{n}(\lambda_i + 1)$). If there is only one unit of each item ($\lambda_i = 1$), CHUNK&SEQUENCE runs in $O(n^2 2^n)$ time and $O(2^n)$ space, so even in that simple setting, the algorithm is exponential in the number of items. In general, for a fixed number of items, CHUNK&SEQUENCE is polynomial in the number of units.

In Figure 2, the thick line represents the path that CHUNK&SEQUENCE chooses. Given the input (Table 3), CHUNK&SEQUENCE returns $c(1, 4) = \{(0, 0), (0, 2), (0, 3), (0, 4), (0, 4), (1, 4)\}$, and $d(1, 4) = \{0, 5, 14, 18, 22, 22\}$. So, in the first step, the supplier delivers 2 units of item 2 and the demander pays \$5. In the second step, the supplier delivers 1 unit of item 2 and the demander pays \$9. In the third step, the supplier delivers 1 unit of item 2 and the demander pays \$4. In the fourth step, the supplier does not deliver, but the demander pays \$4. In the fifth step, the supplier delivers 1 unit of item 1 and the demander does not pay anything. So, the exchange is completed safely in five steps, containing four deliveries and four payments.

6 Many dependent items, independent units

The previous section discussed exchanges of several dependent items whose units are dependent. Exchanges of several dependent items whose units are independent could be handled with the same input form, same algorithm (CHUNK&SEQUENCE) and same output form. We follow this approach except that we capitalize on the independence of the units in the input form. The user inputs how much each additional unit of a given item increases v_s and v_d, given how many units of each other item has been delivered so far.

Example B. Consider a company purchasing a software system consisting of two items: package 1 and package 2. The employees mainly use package 2 and rarely package 1, so the company purchases four licenses (units) of package 2 and one license of package 1. Both packages can work alone but are dependent in the sense of the value functions as shown in Table 4.

To use CHUNK&SEQUENCE to plan an optimal safe exchange, the MTIF needs to be converted into STIF form. The value functions v_s and v_d of the STIF are calculated from the MTIF as follows:

$$v_s(x_1, x_2, \dots, x_n) = \sum_{i=1}^{n} \frac{\Delta v_s}{\Delta x_i}(x_1, \dots, x_i, 0, \dots, 0)x_i$$

and analogously for v_d.

For the MTIF to be consistent, every sequence $(item_{i_1}, item_{i_2}, \dots, item_{i_n})$ must give the same value for $v_s(x_1, x_2, \dots, x_n)$ and for $v_d(x_1, x_2, \dots, x_n)$. This gives additional constraints on the values in the MTIF. For example, in our software license scenario, the following constraints pertain to $v_s(1, 3)$, corresponding to different orders of delivering the items (how many units of package 2 are delivered before the one unit of package 1):

$$\begin{array}{lll}
\text{case 1} & v_s(1,3) & = \frac{\Delta v_s(0,0)}{\Delta x_1} + 3\frac{\Delta v_s(1,0)}{\Delta x_2} \\
\text{case 2} & v_s(1,3) & = 3\frac{\Delta v_s(0,0)}{\Delta x_2} + \frac{\Delta v_s(0,3)}{\Delta x_1} \\
\text{case 3} & v_s(1,3) & = \frac{\Delta v_s(0,0)}{\Delta x_2} + \frac{\Delta v_s(0,1)}{\Delta x_1} + 2\frac{\Delta v_s(1,0)}{\Delta x_2} \\
\text{case 4} & v_s(1,3) & = 2\frac{\Delta v_s(0,0)}{\Delta x_2} + \frac{\Delta v_s(0,2)}{\Delta x_1} + \frac{\Delta v_s(1,0)}{\Delta x_2}
\end{array}$$

So, in the MTIF, the user has too many degrees of freedom (cells to fill in the table). One way to tackle this would be to have the user fill the entire table, and check for consistency afterwards. Instead, we have the user only fill in the minimal number of cells. This precludes the user from submitting inconsistent preferences while maintaining full expressive power within the space of consistent preferences (we do not impose any particular functional form). The following algorithm collects this minimal information from the user. We present the algorithm for collecting the information for v_s. The collection for v_d is analogous. To simplify the notation we

package 1			package 2		
units of package 2 delivered so far	supplier's cost per unit	demander's value per unit	units of package 1 delivered so far	supplier's cost per unit	demander's value per unit
0	$\frac{\Delta v_s(0,0)}{\Delta x_1}$	$\frac{\Delta v_d(0,0)}{\Delta x_1}$	0	$\frac{\Delta v_s(0,0)}{\Delta x_2}$	$\frac{\Delta v_d(0,0)}{\Delta x_2}$
1	$\frac{\Delta v_s(0,1)}{\Delta x_1}$	$\frac{\Delta v_d(0,1)}{\Delta x_1}$	1	$\frac{\Delta v_s(1,0)}{\Delta x_2}$	$\frac{\Delta v_d(1,0)}{\Delta x_2}$
2	$\frac{\Delta v_s(0,2)}{\Delta x_1}$	$\frac{\Delta v_d(0,2)}{\Delta x_1}$			
3	$\frac{\Delta v_s(0,3)}{\Delta x_1}$	$\frac{\Delta v_d(0,3)}{\Delta x_1}$			
4	$\frac{\Delta v_s(0,4)}{\Delta x_1}$	$\frac{\Delta v_d(0,4)}{\Delta x_1}$			

Table 4: *(Unminimized) multiple tables input form (MTIF) for example B.*

package 1			package 2		
units of package 2 delivered so far	supplier's cost per unit	demander's value per unit	units of package 1 delivered so far	supplier's cost per unit	demander's value per unit
0	$\frac{\Delta v_s(0,0)}{\Delta x_1}$	$\frac{\Delta v_d(0,0)}{\Delta x_1}$	0	$\frac{\Delta v_s(0,0)}{\Delta x_2}$	$\frac{\Delta v_d(0,0)}{\Delta x_2}$
			1	$\frac{\Delta v_s(1,0)}{\Delta x_2}$	$\frac{\Delta v_d(1,0)}{\Delta x_2}$

Table 5: *Minimized MTIF for our software license purchasing scenario (example B).*

define $\Delta_{i,j} v_s = \frac{\Delta v_s}{\Delta x_i}(0,\ldots,0,1,0,\ldots,0)$ with 1 in the j^{th} position, and $\Delta_{i,0} v_s = \frac{\Delta v_s}{\Delta x_i}(0,\ldots,0)$.

Before the algorithm is executed, the items can be resorted and then renumbered from 1 to n. The value of a unit of an item is conditioned on the number of units of each earlier item in this list. In our implementation, by default, the new order is chosen so that $\lambda_1 < \lambda_2 < \ldots < \lambda_n$, but the user can choose the order because it may be more natural for her to express the contingencies in some order than in others.

Algorithm 6.1 MINIMIZE-MTIF$((\lambda_1, \lambda_2, \ldots, \lambda_n), n)$

1. user enters $\Delta_{1,0} v_s$

2. $\forall k \in \{2, n\}$ do

- user enters $\Delta_{k,0} v_s$, $\Delta_{k,1} v_s$, ..., $\Delta_{k,k-1} v_s$
- $\forall (x_1, x_2, \ldots, x_{k-1}) \in \{0, \lambda_1\} \times \ldots \times \{0, \lambda_{k-1}\}$, $\frac{\Delta v_s}{\Delta x_k}(x_1, x_2, \ldots, x_{k-1}, 0, \ldots, 0) = \sum_{h=1}^{k-1} x_h \Delta_{k,h} v_s - \left(\sum_{h=1}^{k-1} x_h - 1\right) \Delta_{k,0} v_s$

3. $\forall (x_1, x_2, \ldots, x_n) \in \{0, \lambda_1\} \times \{0, \lambda_2\} \times \ldots \times \{0, \lambda_n\}$, $v_s(x_1, x_2, \ldots, x_n) = \sum_{h=1}^{n} \frac{\Delta v_s}{\Delta x_h}(x_1, x_2, \ldots, x_h, 0, \ldots, 0) x_h$

The number of cells that the user ends up filling in this minimal MTIF (see Table 5) is $2 \cdot \frac{n(n+1)}{2} = n^2 + n$. This is significantly less than the number of cells to fill in the corresponding STIF ($2 \cdot \prod_{i=1}^{n} (\lambda_i + 1) \geq 2^{n+1}$) because we capitalize on the independence of items. Once the cells have been filled, CHUNK&SEQUENCE plans a safe minimal exchange.

7 Conclusions

Nondelivery is a major problem in exchanges, especially in electronic commerce. By splitting the exchange into chunks, and by appropriately sequencing the chunks, exchanges can sometimes be structured to be self-enforcing. We presented the design of a safe exchange planning and executing agent. We presented algorithms that provably find a safe exchange plan (chunking and chunk sequence) if one exists, and require the minimal amount of input from the user. The algorithms and the input interface depend on the setting: whether there is one or multiple items to exchange, whether the items are dependent or independent, whether the units are dependent or independent, and whether the goods are countable or uncountable. All of the algorithms and input interfaces of the paper have been implemented, and a safe exchange planning service, *eExchangeHouse*, is offered on the web as part of *eMediator*, our next generation electronic commerce server (`http://ecommerce.cs.wustl.edu/emediator/`).

References

[1] Sviatoslav Brainov. Deviation-proof plans in open multiagent environments. *ECAI*, p. 274–278, 1994.

[2] James Friedman and Peter Hammerstein. To trade, or not to trade; that is the question. In Reinhard Selten, ed., *Game equilibrium models I: Evolution and Game Dynamics*, p. 257–275. Springer, 1991.

[3] National Consumers League. New NCL survey shows consumers are both excited and confused about shopping online, 1999. www. natlconsumersleague. org/ BeEWisepr.html, Oct. 20. By Opinion Research Corp.

[4] Ori Regev and Noam Nisan. The POPCORN market - an online market for computational resources. *Intl. Conf. on Information and Computation Economies*, p. 148–157, Charleston, SC, 1998.

[5] Jeffrey S Rosenschein and Gilad Zlotkin. *Rules of Encounter*. MIT Press, 1994.

[6] Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. *AAAI*, p. 256–262, Washington, D.C., 1993.

[7] Tuomas Sandholm. *Negotiation among Self-Interested Computationally Limited Agents*. PhD thesis, Univ. of Massachusetts, Amherst, 1996. www.cs.wustl.edu/ ~sandholm/ dissertation.ps.

[8] Tuomas Sandholm and Victor Lesser. Equilibrium analysis of the possibilities of unenforced exchange in multiagent systems. *IJCAI*, p. 694–701, 1995.

[9] R Selten. Spieltheoretische behandlung eines oligopolmodells mit nachfrageträgheit. *Zeitschrift für die gesamte Staatswissenschaft*, 12:301–324, 1965.

[10] L G Telser. A theory of self-enforcing agreements. *Journal of Business*, 53(1):27–44, 1980.