University of Groningen

Faculty of Science and Engineering

Bachelor Thesis

An Investigation of Exact Methods for Clearing Barter Exchange Markets

Sietze Houwink

supervisors
prof. dr. G.R. Renardel de Lavalette
B. Jargalsaikhan

July 12, 2017

Abstract

Agents of a barter exchange market seek to exchange items without the aid of money. The goal is to carefully coordinate the transactions so as to maximize overall utility. In this thesis, we propose clearing methods that besides trader agents, also account for donor and receiver agents. New insights will be presented regarding the topics execution time, optimal exchange value and the effect of donor agents. This thesis will present to the reader an extensive introduction to the required tools, clearing methods and implementation details. This thesis builds on the research done by (Abraham et al., 2007) and (Glorie, 2014), who have investigated kidney exchange markets.

Contents

1 Introduction

Agents of a barter exchange market (abbreviated to market in the sequel) seek to exchange items without the aid of money. The goal is to clear the market, that is to carefully coordinate the transactions so as to maximize overall utility.

Prior researches (Abraham et al., 2007) and (Glorie, 2014) have proposed various clearing methods, that have been designed for the purpose of clearing nationwide kidney exchange markets. This thesis is focused on various aspects of clearing more general markets.

The clearing methods proposed by these prior researches focus on markets with trader agents, and occasionally also include donor agents. This thesis considers markets that include trader, donor and receiver agents. Various clearing methods will be proposed that are capable of clearing markets that include these agents.

These prior researches have investigated the execution time as a function of the number of agents, for several instances of kidney exchange markets. This thesis considers both the quantities execution time and the optimal exchange value, as a function of market properties. The considered markets were randomly generated according to a number of agents and a uniform match probability. The execution time measurements were performed for solvers based on various clearing methods. The exchange value measurements were performed for either markets with additional donors, or markets with a maximum allowed number of agents per transaction.

This thesis will present to the reader an extensive introduction to the required tools, clearing methods and implementation details. Chapters 2, 3 and 4 introduce the reader to the required optimization tools, that are linear programming, mixed integer linear programming, and binary integer linear programming. Chapter 5 defines the market, provides representations for markets, and provides an introduction to the clearing problems. Chapters 6 and 7 present the various clearing methods, for both the unrestricted and the restricted clearing problems. Chapter 8 is devoted to the implementation details, experiments and results.

2 Notation

The required tools demand a basic understanding of various linear algebra concepts, that will be introduced throughout the text. This chapter provides an overview of the used notations. The variables and constants in this thesis take on real values, unless explicitly stated otherwise.

A **row vector** consisting of n elements is denoted by

$$\mathbf{a} = (a_1, a_2, \ldots, a_n)$$

A **column vector** consisting of n elements is denoted by

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix} = (a_1, a_2, \ldots, a_n)^T$$

An $m \times n$ **matrix** is denoted by

$$A = \begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_m \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}$$

The **dot product** of two vectors consisting of n elements is denoted by

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

3 Linear Programming

The research is focused on tools that provide exact optimal solutions, as any loss of optimality may be undesirable for certain markets, such as the kidney exchange markets. Linear programming and binary integer linear programming are often the preferred tools, as these provide exact solutions and naturally allow the user to experiment with restrictions that are hard to incorporate by using other methods. This chapter provides an introduction to common formulations, finding optimal solutions and duality. This chapter is inspired by (Cormen, 2009).

3.1 Definitions

This section provides an overview of the relevant linear programming definitions.

Linear Programming

Linear programming is a mathematical technique for optimizing, that is either minimizing or maximizing, a linear function of several variables, subject to a finite set of linear constraints, that are linear equalities, or linear inequalities that are not strict. A **linear program** (LP) is defined as a linear programming problem statement.

Objectives

Let from this point forward, $\mathbf{x}$ denote a column vector consisting of the n variables of an LP, and $\mathbf{c}$ denote a column vector consisting of n constants.

The **linear objective function** is then defined by

$$\mathbf{c} \cdot \mathbf{x}$$

The **objective value**, for some setting of $\mathbf{x}$, is defined by the value of the linear objective function.

Constraints

Let from this point forward $\mathbf{a}$ denote a row vector consisting of n constants, and b denote a single constant.

Let $\triangle$ denote a **linear operator**, that is either $=$, $\leq$ or $\geq$. A **linear constraint** is then defined by

$$\mathbf{a} \cdot \mathbf{x} \triangle b$$

Let from this point forward, $\mathbf{a}_i \cdot \mathbf{x} \triangle b_i$ for $i \in \{1, 2, \ldots, m\}$ denote m linear constraints, $A = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m)^T$ denote an $m \times n$ matrix, and $\mathbf{b} = (b_1, b_2, \ldots, b_m)^T$ denote a column vector consisting of m elements. The m linear constraints can then be formulated in **matrix notation** as

$$A\,\mathbf{x} \triangle \mathbf{b}$$

Solutions

A **feasible solution** to an LP is a setting of $\mathbf{x}$ that satisfies all constraints of the LP.

An LP is called **feasible** if a feasible solution exists, and is called **infeasible** otherwise.

An LP is called **unbounded** if the LP is feasible, but no finite minimum or maximum objective value exists, and is called **bounded** otherwise.

The **optimal solution** to an LP is a feasible solution with a minimum or maximum objective value.

3.2 Formulations

A single LP can be described in a wide variety of forms, though methods that operate on an LP often require the LP to be described in a specific form. The standard form and the slack form are well known forms, and any LP can be written as an equivalent LP in one of these forms. The standard form is conceptually easier, while the slack form reveals more interesting properties. This section provides a definition and conversion notes for both the standard form and the slack form.

3.2.1 Standard Form

An LP in **standard form** is described by

$$\begin{array}{ll} \text{maximize} & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & A\,\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array} \tag{1}$$

Conversion Notes

The following conversion rules allow any LP to be written as an equivalent LP in standard form.

An LP that aims to minimize the objective function, can be written as an LP that aims to maximize the objective function.

$$\text{minimize} \;\; \mathbf{c} \cdot \mathbf{x} \quad \equiv \quad \text{maximize} \;\; (-\mathbf{c}) \cdot \mathbf{x}$$

A linear $\geq$ constraint, can be written as a linear $\leq$ constraint.

$$\mathbf{a} \cdot \mathbf{x} \geq b \quad \equiv \quad (-\mathbf{a}) \cdot \mathbf{x} \leq (-b)$$

A linear $=$ constraint, can be written as a combination of a linear $\leq$ and a linear $\geq$ constraint.

$$\mathbf{a} \cdot \mathbf{x} = b \quad \equiv \quad \mathbf{a} \cdot \mathbf{x} \leq b \;\wedge\; \mathbf{a} \cdot \mathbf{x} \geq b$$

A variable x without a constraint $x \geq 0$, can be replaced by two variables x_1 and x_2 with constraints $x_1 \geq 0$ and $x_2 \geq 0$, by replacing each occurrence of the variable x by $x_1 - x_2$ in the LP (Cormen, 2009, Ch 29.1 - Converting linear programs into standard form).

3.2.2 Slack Form

Consider an LP in standard form, as in equation (1). Let $\mathbf{x}^{non\text{-}basic}$ denote an alias for $\mathbf{x}$, and $\mathbf{x}^{basic}$ denote a set of auxiliary variables, measuring slack. An LP in **slack form** is then described by

$$\begin{aligned} \text{maximize} \quad & c^{opt} + \mathbf{c} \cdot \mathbf{x}^{non\text{-}basic} \\ \text{subject to} \quad & \mathbf{x}^{basic} = \mathbf{b} - A\,\mathbf{x}^{non\text{-}basic} \\ & \mathbf{x}^{non\text{-}basic} \geq 0 \\ & \mathbf{x}^{basic} \geq 0 \end{aligned} \tag{2}$$

The optional constant c^{opt} is introduced to allow conversion between equivalent LPs in slack form.

Conversion Notes

Any LP can be written as an equivalent LP in standard form, from which an LP in slack form can be derived. Any LP can therefore be written as an equivalent LP in slack form.

3.3 Finding Solutions

An LP with n variables describes an n-dimensional space $\mathbb{R}^n$, for which each linear constraint, considering the slack form, defines a half-space of points satisfying this constraint. The intersection of these subspaces defines a not necessarily n-dimensional convex polytope called a **simplex**, which defines all feasible solutions of the LP. It has been proven that an optimal solution to the LP can always be found at a vertex of the simplex.

3.3.1 The Simplex Method

The basic algorithm using the simplex method initiates by finding some vertex of the simplex, then performs a walk on the vertices of the simplex in non-decreasing objective value order. On finding a local maximum, the algorithm may terminate or even continue indefinitely, depending on the pivoting rule. By the convexity of the simplex, and the linearity of the objective function, the local maximum is also the global maximum.

Iteration Procedure

Consider an LP in slack form, as in equation (2). In each iteration of the algorithm, the goal is to alter the setting of $\mathbf{x}_{non\text{-}basic}$ as to increase the objective value

$$c^{opt} + \mathbf{c} \cdot \mathbf{x}^{non\text{-}basic}$$

This is realized by increasing a single term of the summation

$$\mathbf{c} \cdot \mathbf{x}^{non\text{-}basic} = \sum_{i=1}^{n} c_i \cdot x_i^{non\text{-}basic}$$

As a consequence of the constraint $\mathbf{x}_{non\text{-}basic} \geq 0$, the variables of $\mathbf{x}_{non\text{-}basic}$ cannot be decreased indefinitely, yet can be increased indefinitely. A term with positive constant c_i is therefore selected. Let the selected term be denoted by

$$c_\alpha \cdot x_\alpha^{non\text{-}basic}$$

If the LP is bounded, the maximum increase of this term must be finite, moreover there exists a constraint in

$$\mathbf{x}^{basic} = \mathbf{b} - A\,\mathbf{x}^{non\text{-}basic}$$

that in combination with the constraints $\mathbf{x}_{non\text{-}basic} \geq 0$, places a least upper bound on the increase of $c_\alpha \cdot x_\alpha^{non\text{-}basic}$. Let this constraint be denoted by

$$x_\beta^{basic} = b_\beta - \mathbf{a}_\beta\,\mathbf{x}^{non\text{-}basic} \tag{3}$$

The iteration is concluded by performing a pivot operation on the variables $x_\alpha^{non\text{-}basic}$ and x_β^{basic}.

The Pivot Operation

The pivot operation transforms the LP, thereby converting $x_\alpha^{non\text{-}basic}$ to a basic variable and x_β^{basic} to a non-basic variable.

Let equation (3) that is solved for $x_\alpha^{non\text{-}basic}$ be denoted by

$$x_\alpha^{non\text{-}basic} = \ldots \tag{4}$$

The LP is transformed by substituting the constraint in equation (3) by the constraint in equation (4), and by substituting the occurrences of $x_\alpha^{non\text{-}basic}$, in all remaining equations of the LP, by the right hand side of equation (4).

Finding an Initial Vertex

If the origin of the LP is a feasible solution, the algorithm is initiated with the origin as initial vertex, otherwise an additional non-basic variable x_γ is introduced. The LP is feasible only if $x_\gamma = 0$ is the optimal solution of the LP

$$\begin{aligned} \text{maximize} \quad & -x_\gamma \\ \text{subject to} \quad & \mathbf{x}^{basic} = \mathbf{b} - A\,\mathbf{x}^{non\text{-}basic} + x_\gamma \mathbf{1} \\ & \mathbf{x}^{non\text{-}basic} \geq 0 \\ & \mathbf{x}^{basic} \geq 0 \\ & x_\gamma \geq 0 \end{aligned} \tag{5}$$

Let the equation in $\mathbf{x}^{basic} = \mathbf{b} - A\,\mathbf{x}^{non\text{-}basic} + x_\gamma$ with minimum value in $\mathbf{b}$ be denoted by

$$x_\delta^{basic} = b_\delta - \mathbf{a}_\delta\,\mathbf{x}^{non\text{-}basic} + x_\gamma$$

In order to make the origin a feasible solution, a pivot operation is performed on the variables x_γ and x_δ^{basic}. The previously described iteration procedure is now sufficient to solve this LP.

If x_γ is a basic variable in the last iteration, an additional pivot operation is performed that ensures that x_γ is a non-basic variable. The variable x_γ is then removed from the LP, and the original objective function is restored, thereby substituting the basic variables in the objective function by the right hand side of their corresponding constraints.

The optimal solution of the obtained LP is now identical to that of the original LP, and additionally the origin is a feasible solution.

Complexity Analysis

The simplex method algorithm is able to solve the majority of the LPs, that are used in practise, with m constraints by performing m to $3m$ pivot operations. In practice, the performance of the simplex method algorithm is excellent for the vast majority of cases. Some cases have been developed however that need as many as $2^m - 1$ pivot operations in order to solve the LP (Klee and Minty, 1970).

3.3.2 Alternative Methods

A second class of linear programming methods are the interior-point methods. These methods move through the interior of the simplex, rather than along the exterior. Note that the optimal solution to the LP will still be at a vertex of the simplex.

Complexity Analysis

The interior-point methods have a worst-case polynomial time complexity. In practice, the performance of the interior-point method algorithms is comparable to that of the simplex method algorithms. Often, only specific properties of the LP under consideration are decisive to which method has the best performance.

3.4 Duality

Proving optimality of solutions obtained from incremental formulation approaches, as will be explained in more detail in section 7.3, demands a basic understanding of duality. This section provides an introduction to duality and an overview of the relevant theorems.

Let the **primal linear program** (primal LP) be denoted by an LP in standard form, as in equation (1), and let $\mathbf{y}$ denote a column vector of m variables, that is a variable for each constraint of the primal LP. The **dual linear program** (dual LP) is then defined by

$$\begin{aligned} \text{minimize} \quad & \mathbf{b} \cdot \mathbf{y} \\ \text{subject to} \quad & A^T \mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq 0 \end{aligned} \tag{6}$$

Theorems

Let $\mathbf{x}^{feas}$ denote any finite feasible solution to the primal LP, and $\mathbf{y}^{feas}$ denote any finite feasible solution to the dual LP.

The **weak duality theorem** states that

$$\mathbf{c} \cdot \mathbf{x}^{feas} \leq \mathbf{b} \cdot \mathbf{y}^{feas}$$

The **strong duality theorem** states that $\mathbf{x}^{feas}$ and $\mathbf{y}^{feas}$ are finite optimal solutions for their respective LPs if

$$\mathbf{c} \cdot \mathbf{x}^{feas} = \mathbf{b} \cdot \mathbf{y}^{feas}$$

4 Mixed Integer Linear Programming

A solution to an LP may be fractional. This is inconvenient for the purpose of clearing markets, as transactions either take place or not. This chapter provides an introduction to finding integral solutions to an LP.

Some problems, for example the maximum weight matching problem in bipartite graphs, can be solved by an LP that is guaranteed to provide an integral solution. A problem that can be reduced to such a problem can therefore be solved in polynomial time. There is in general no known polynomial time algorithm to find integral solutions to any LP.

4.1 Definitions

A **mixed integer linear program** (MILP) is obtained from an LP by additionally restricting a subset of the variables to be integer.

A **binary integer linear program** (BILP) is obtained from a MILP by additionally restricting all variables to be binary variables. This formulation turns out to be most convenient for the majority of the presented clearing methods in this thesis.

4.2 Branching

The optimal solution to a MILP is obtained by solving the corresponding LP first, that is the MILP without the integrality constraints. If the solution to the LP contains fractional values, some variable x corresponding to a fractional solution v is selected. The master MILP is then branched obtaining two more specific MILPs. The first branch extends the MILP with the constraint

$$x \leq \lfloor v \rfloor$$

while the second branch extends the MILP with the constraint

$$x \geq \lceil v \rceil$$

The process of solving the corresponding LP and branching is repeated recursively, until the solution to the LP contains integral values only. Note that multiple integral solutions may exist. The integral solution with the largest objective value is the optimal solution to the master MILP.

4.3 Bounding

Consider a node of the branch tree. The objective value of the optimal solution to the MILP of the node, is a lower bound for the master MILP. The objective value of the optimal solution to the corresponding LP of the node, is an upper bound for the MILP of the node. If the upper bound of the node is smaller than or equal to the best lower bound, then the subtree of the node can be pruned. If the lower bound of the node is equal to the upper bound of the root node, then the optimal solution is found at that node.

(Glorie, 2014, Ch 3.3.1, Ch 3.3.2 - Bounding)

5 Barter Exchange Markets

The research is focused on markets with the following restrictions. An agent is allowed to provide at most one item to the market, and is allowed to receive at most one item from the market. Note that these restrictions do allow an agent to be interested in more than one provided item of the market. Also note that transactions are not limited to involve exactly two agents, but may involve any number of agents. This chapter provides the definition of a barter exchange market, and introduces graphical representations, structures, and an example market.

5.1 Market Definition

An **agent** of a market M, provides an item to M and/or wants any of a selected subset of the items from M. An agent is categorized as either type donor, receiver or trader.

A **donor** only provides an item to M, while a **receiver** only wants an item from M.

A **trader** primarily wants an item from M, and is able to provide a different item to M in return. A trader will only provide an item if necessary and sufficient for obtaining a wanted item. Consequently, a trader will not provide an item if either a wanted item can be obtained without providing an item, or if a wanted item cannot be obtained while providing an item.

A **barter exchange market** M can be defined by the unweighted digraph G^{dir}. Note that self-loops are forbidden.

$$
\begin{aligned}
&G^{dir} = (V, E)\\
&V = D \cup R \cup T\\
&D = \{d \mid d \text{ is a donor in } M\}\\
&R = \{r \mid r \text{ is a receiver in } M\}\\
&T = \{t \mid t \text{ is a trader in } M\}\\
&E = \{(v_1, v_2) \mid (v_1, v_2) \in (D \cup T) \times (R \cup T),\ v_1 \neq v_2,\\
&\qquad v_1 \text{ provides an item to } M \text{ that } v_2 \text{ wants from } M\}
\end{aligned}
$$

5.2 Example Market

The reader will be assisted throughout the text by the following example market, which has been designed to be as illustrative as possible. This market contains a few but a wide variety of structures (as defined in section 5.5.2), and makes each presented constraint necessary for a correct solution.

$$
\begin{aligned}
&G^{dir\text{-}ex} = (V, E)\\
&V = D \cup R \cup T\\
&D = \{d_1\}\\
&R = \{r_1\}\\
&T = \{t_1, t_2, t_3\}\\
&E = \{(d_1, r_1), (d_1, t_1), (t_1, r_1), (t_1, t_2), (t_2, t_3), (t_3, t_1), (t_3, t_2)\}
\end{aligned}
$$

5.3 Exchanges

An **exchange** X for M is defined by a subset of the possible transactions

$$X \subseteq E$$

such that an item is given away and is received at most once,

$$
\begin{aligned}
((v_1, v_2) \in X \wedge (v_1, v_3) \in X) \rightarrow (v_2 = v_3)\\
((v_1, v_2) \in X \wedge (v_3, v_2) \in X) \rightarrow (v_1 = v_3)
\end{aligned}
$$

and a trader that gives away an item, also receives an item

$$((v_1, v_2) \in X \wedge v_1 \in T) \rightarrow (\exists v_3 \ (v_3, v_1) \in X)$$

The **exchange value** for X is defined by the number of transactions $|X|$.

5.4 Clearing Markets

Clearing a market is equivalent to carefully coordinating the transactions as to maximize overall utility. This section provides definitions for the unrestricted clearing problem and the restricted clearing problem.

The Unrestricted Clearing Problem

Let $\mathcal{X}$ denote the set of all possible exchanges for M

$$\mathcal{X} = \{X \mid X \text{ is an exchange for } M\}$$

The **unrestricted clearing problem** (UCP) involves finding an exchange $X \in \mathcal{X}$ with maximum exchange value. This problem is polynomial time solvable, as this problem can be reduced to a special case of the maximum weight matching problem in bipartite graphs (Glorie, 2014, Ch 3.3.2 - Bounding), as will be explained in section 6.1.2.

The Restricted Clearing Problem

Let $\mathcal{X}$ denote the set of all possible exchanges for M, with cycles of length at most ℓ_1, and chains of length at most ℓ_2.

$$\begin{aligned}\mathcal{X} = \{X \mid\ & X \text{ is an exchange for } M,\\ & \text{cycles in } X \text{ are of length at most } \ell_1,\\ & \text{chains in } X \text{ are of length at most } \ell_2\}\end{aligned}$$

The **restricted clearing problem** (RCP) involves finding an exchange $X \in \mathcal{X}$ with maximum exchange value. This problem is in general NP-complete (Glorie, 2014, Ch 3.2.2).

5.5 Unweighted Digraph

This section provides a graphical representation of the example market, defines structures, and examines exchanges.

5.5.1 Graphical Representation

A graphical representation of $G^{dir\text{-}ex}$ can be found in Figure 1. The edges are labeled for future reference.

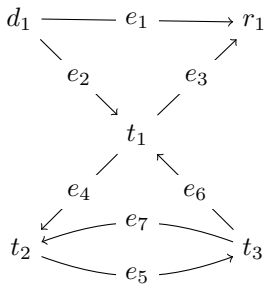


Figure 1: A graphical representation of $G^{dir\text{-}ex}$ and its edge labels.

5.5.2 Structures

Markets exist in which certain exchange structures are undesirable. This section provides an overview of the relevant structures, that are paths, cycles and chains.

Paths

Let a **path** of length $\ell \geq 1$ in G^{dir} be denoted by

$$p = (v_1, v_2, \ldots, v_{\ell+1})$$

such that

$$\begin{array}{ll} \forall i \leq \ell + 1 & v_i \in V \\ \forall i, j \leq \ell + 1 & (i < j \rightarrow v_i \neq v_j) \\ \forall i \leq \ell & (v_i, v_{i+1}) \in E \end{array}$$

For example, (t_3, t_1, r_1) is a path of length 2 in $G^{dir\text{-}ex}$.

Chains

Let a **chain** of length $\ell \geq 1$ in G^{dir} be denoted by

$$ch = (v_1, v_2, \ldots, v_{\ell+1})$$

such that

$$\begin{array}{ll} & v_1 \in D \\ \forall 2 \leq i \leq \ell & v_i \in T \\ & v_{\ell+1} \in R \cup T \\ \forall i, j \leq \ell + 1 & (i < j \rightarrow v_i \neq v_j) \\ \forall i \leq \ell & (v_i, v_{i+1}) \in E \end{array}$$

For example, (d_1, t_1, t_2) is a chain of length 2 in $G^{dir\text{-}ex}$.

Cycles

Let a **cycle** of length $\ell \geq 2$ in G^{dir} be denoted by

$$cy = (v_1, v_2, \ldots, v_\ell, v_1)$$

such that

$$\begin{array}{ll} \forall i \leq \ell & v_i \in T \\ \forall i, j \leq \ell + 1 & (i < j \rightarrow v_i \neq v_j) \\ \forall i \leq \ell & (v_i, v_{i+1}) \in E \\ & (v_\ell, v_1) \in E \end{array}$$

For example, (t_1, t_2, t_3, t_1) is a cycle of length 3 in $G^{dir\text{-}ex}$.

5.5.3 Exchanges

An exchange for M can alternatively be described in terms of the cycles and chains in G^{dir}.

Let from this point forward, C denote the set of all cycles and chains in G^{dir}.

$$C = \{c \mid c \text{ is a cycle in } G^{dir} \text{ or } c \text{ is a chain in } G^{dir}\}$$

Let C^{disj} denote a subset of the cycles and chains in G^{dir}

$$C^{disj} \subseteq C$$

such that the cycles and chains are disjoint

$$\forall c_1, c_2 \in C^{disj} \quad c_1 \neq c_2 \rightarrow (\ nodes(c_1) \cap nodes(c_2) = \emptyset\ )$$

Such a set C^{disj} represents an exchange X for M, defined by

$$X = \{(v_1, v_2) \in edges(c) \mid c \in C^{disj}\}$$

Example Market

A graphical representation of an optimal exchange for the UCP is indicated by the black arrows in Figure 2. A graphical representation of an optimal exchange for the RCP, where the length of a cycle is at most 2 and the length of a chain is at most 1, is indicated by the black arrows in Figure 3.

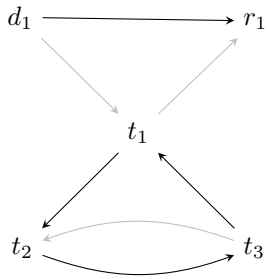


Figure 2: An optimal exchange for the UCP.

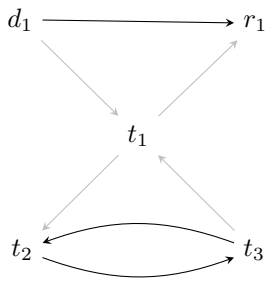


Figure 3: An optimal exchange for the RCP. The length of a cycle is at most 2 and the length of a chain is at most 1.

5.6 Weighted Undirected Bipartite Graph

This section provides a market definition in terms of a weighted undirected bipartite graph, a graphical representation of the example market, and examines exchanges.

5.6.1 Market Definition

Market M can alternatively be represented by the weighted undirected bipartite graph G^{bip}. Nodes indicated by p represent provided items, while nodes indicated by w represent wanted items. The superscripts 0 and 1 indicate the weight of the edges in the respective set.

$$
\begin{aligned}
&G^{bip} = (V^p, V^w, E^0 \cup E^1)\\
&V^p = D^p \cup T^p\\
&V^w = R^w \cup T^w\\
&D^p = D \times \{p\}\\
&T^p = T \times \{p\}\\
&R^w = R \times \{w\}\\
&T^w = T \times \{w\}\\
&E^1 = \{\{(v_1, p), (v_2, w)\} \mid (v_1, v_2) \in E\}\\
&E^0 = \{\{(t, p), (t, w)\} \mid t \in T\}
\end{aligned}
$$

5.6.2 Graphical Representation

A graphical representation of $G^{bip\text{-}ex}$ can be found in Figures 4 and 5. The edges are labeled for future reference.

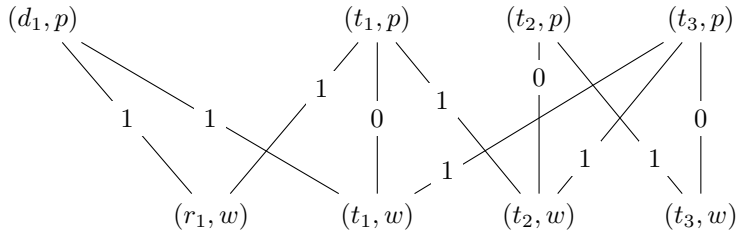


Figure 4: A graphical representation of $G^{bip\text{-}ex}$ and its edge weights.

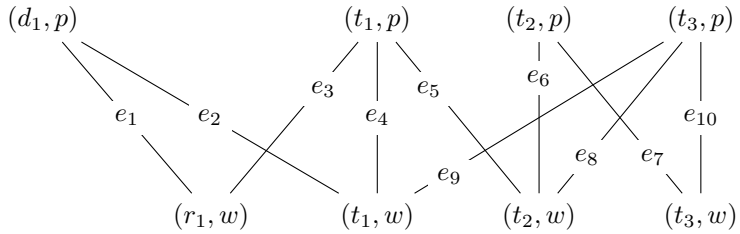


Figure 5: A graphical representation of $G^{bip\text{-}ex}$ and its edge labels.

5.6.3 Exchanges

An exchange for M can alternatively be described in terms of a maximum weight matching of G^{bip}.

A **matching** is a set of edges without common vertices. Let $\mathcal{M}$ denote a matching in G^{bip} such that a trader that gives away an item, also receives an item

$$(v_1 \in T \wedge \exists v_2 \ \{(v_1, p), (v_2, w)\} \in \mathcal{M}) \rightarrow (\exists v_3 \ \{(v_3, p), (v_1, w)\} \in \mathcal{M})$$

Such a matching $\mathcal{M}$ represents an exchange X for M, defined by

$$X = \{(v_1, v_2) \mid \{(v_1, p), (v_2, w)\} \in \mathcal{M}\}$$

Example Market

For the example market, a graphical representation of an optimal exchange for the UCP exchange is indicated by the black arrows in Figure 3.

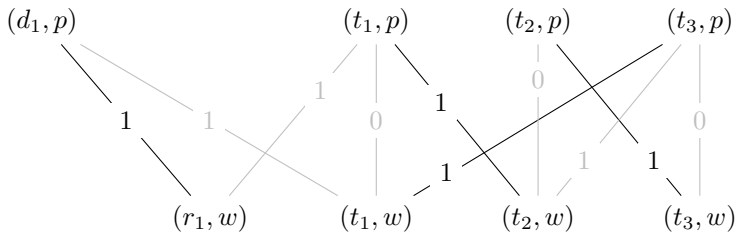


Figure 6: An optimal exchange for the UCP.

6 The Unrestricted Clearing Problem

The UCP can be solved by methods based on either the edge formulation or the cycle and chain formulation. The methods based on the edge formulation select a subset of the edges, while the methods based on the cycle and chain formulation select a subset of the cycles and chains, to be included in the optimal exchange. This chapter provides derivations for the methods, as well as BILP or LP formulations.

6.1 Edge Formulations

This section provides an edge formulation for both the unweighted directed graph G^{dir}, as well as for the weighted undirected bipartite graph G^{bip}. The edge formulation for G^{dir} arises naturally, and allows extensions to formulations solving the RCP, as will be explained in more detail in section 7.1. The edge formulation for G^{bip} is polynomial time solvable, as will be explained in more detail in section 6.1.2.

6.1.1 Unweighted Digraphs

Constraints

Recall that a trader will only provide an item if necessary and sufficient for obtaining a wanted item. Each node in T is therefore constrained to have an indegree not smaller than its outdegree in the exchange.

$$\begin{aligned} & \forall v \in T \quad \mathit{indegree}(v) \geq \mathit{outdegree}(v) \\ \Leftrightarrow \quad & \forall v \in T \quad \mathit{outdegree}(v) - \mathit{indegree}(v) \leq 0 \end{aligned}$$

An agent is allowed to receive at most one item. Each node is therefore constrained to have an indegree of at most 1 in the exchange.

$$\forall v \in T \cup R \quad \mathit{indegree}(v) \leq 1$$

Note that nodes in D are not included, as the indegree for these nodes is 0 by construction.

An agent is allowed to provide at most one item. Each node is therefore constrained to have an outdegree of at most 1 in the exchange.

$$\forall v \in D \quad \mathit{outdegree}(v) \leq 1$$

Note that nodes in T are not included, as the constraints for these nodes will not be binding in combination with the previously defined constraints. Also note that nodes in R are not included, as the outdegree for these nodes is 0 by construction.

BILP Conversion

The number of variables n of the BILP is equal to the number of edges in G^{dir}.

$$n = |E|$$

Let $\mathbf{x}$ be a column vector consisting of n binary variables.

Let $\mathbf{a}_v^{out\text{-}in}$ denote a row vector consisting of n values corresponding to node v. Let the elements of $\mathbf{a}_v^{out\text{-}in}$ be defined by

$$\forall i \leq n \;\; a_{vi}^{out\text{-}in} = \begin{cases} 1 & \text{if } e_i \text{ is outgoing for } v \\ -1 & \text{if } e_i \text{ is incoming for } v \\ 0 & \text{otherwise} \end{cases}$$

Let $\mathbf{a}_v^{in}$ denote a row vector consisting of n values corresponding to node v. Let the elements of $\mathbf{a}_v^{in}$ be defined by

$$\forall i \leq n \;\; a_{vi}^{in} = \begin{cases} 1 & \text{if } e_i \text{ is incoming for } v \\ 0 & \text{otherwise} \end{cases}$$

Let $\mathbf{a}_v^{out}$ denote a row vector consisting of n values corresponding to node v. Let the elements of $\mathbf{a}_v^{out}$ be defined by

$$\forall i \leq n \;\; a_{vi}^{out} = \begin{cases} 1 & \text{if } e_i \text{ is outgoing for } v \\ 0 & \text{otherwise} \end{cases}$$

The constraints then define the BILP given by

$$\begin{array}{lll} \text{maximize} & \mathbf{1} \cdot \mathbf{x} & \\ \text{subject to} & \forall v \in T & \mathbf{a}_v^{out\text{-}in}\, \mathbf{x} \leq 0 \\ & \forall v \in T \cup R & \mathbf{a}_v^{in}\, \mathbf{x} \leq 1 \\ & \forall v \in D & \mathbf{a}_v^{out}\, \mathbf{x} \leq 1 \end{array} \tag{7}$$

(Abraham et al., 2007, inspired by Ch 4.0)

Example Market

The constraint $\forall v \in T \;\; \mathbf{a}_v^{out\text{-}in}\, \mathbf{x} \leq 0$ for t_1 is then given by

$$\begin{array}{c} \\ t_1 \end{array} \begin{array}{c} \begin{array}{ccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \end{array} \\ \left(\begin{array}{ccccccc} 0 & -1 & 1 & 1 & 0 & -1 & 0 \end{array} \right) \end{array} \mathbf{x} \;\leq\; 0$$

6.1.2 Weighted Undirected Bipartite Graphs

Constraints

An agent is allowed to provide at most one item, and receive at most one item. Each node is therefore constrained to be the endpoint for at most one edge in the exchange.

$$\forall v \in D^p \cup T^p \cup R^w \quad degree(v) \leq 1$$

Note that nodes in T^w are not included, as the constraints for these nodes will not be binding in combination with constraints that will be defined now.

Recall that a trader will only provide an item if necessary and sufficient for obtaining a wanted item. If the nodes in T^w are constrained to be the endpoint for exactly one edge in the exchange

$$\forall v \in T^w \quad \text{degree}(v) = 1$$

then only one of the cases in Figure 7 may occur, which are precisely all the cases that are allowed.

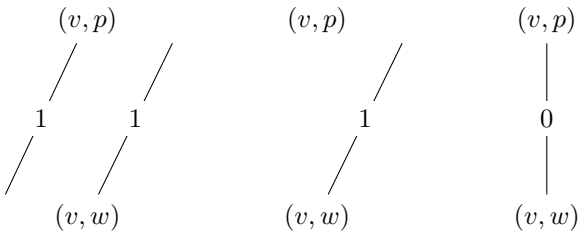


Figure 7: Allowed cases traders.

BILP Conversion

The number of variables n of the BILP is equal to the number of edges in G^{bip}.

$$n = |E|$$

Let $\mathbf{x}$ be a column vector consisting of n binary variables, and let the elements of the column vector $\mathbf{c}$ be defined by

$$\forall i \leq n \quad c_i = weight(e_i)$$

Let $\mathbf{a}_v^{deg}$ denote a row vector consisting of n values corresponding to node v. Let the elements of $\mathbf{a}_v^{deg}$ be defined by

$$\forall i \leq n \quad a_{vi}^{deg} = \begin{cases} 1 & \text{if } v \text{ is an endpoint for } E_i \\ 0 & \text{otherwise} \end{cases}$$

The constraints then define the BILP given by

$$\begin{aligned} \text{maximize} \quad & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} \quad & \forall v \in T^w \qquad \mathbf{a}_v^{deg}\, \mathbf{x} = 1 \\ & \forall v \in D^p \cup T^p \cup R^w \quad \mathbf{a}_v^{deg}\, \mathbf{x} \leq 1 \end{aligned} \tag{8}$$

LP Conversion

The following LP finds a maximum weight matching in G^{bip}.

$$\begin{aligned} \text{maximize} \quad & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} \quad & \forall v \in V^p \cup V^w \quad \mathbf{a}_v^{deg}\, \mathbf{x} \leq 1 \\ & \forall i \leq n \qquad\qquad x_i \in [0,1] \end{aligned} \tag{9}$$

The convex hull of all the matchings in a graph forms a polytope called the matching polytope. It has been proven that the matching polytope always contains all the fractional matchings if and only if the graph is bipartite. For this reason, the LP solutions are integral. The additional equality constraints that are imposed in equation (8) do not influence this property. The LP defined below therefore provides an equivalent optimal solution as the BILP defined in equation (8).

$$\begin{aligned} \text{maximize} \quad & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} \quad & \forall v \in T^w \qquad \mathbf{a}_v^{deg}\, \mathbf{x} = 1 \\ & \forall v \in D^p \cup T^p \cup R^w \quad \mathbf{a}_v^{deg}\, \mathbf{x} \leq 1 \\ & \forall i \leq n \qquad\qquad x_i \in [0,1] \end{aligned} \tag{10}$$

(Glorie, 2014, Ch 3.3.2 - Bounding), (Mahajan, 2010).

Example Market

The constraint $\forall v \in D^p \cup T^p \cup R^w \quad \mathbf{a}_v^{deg}\, \mathbf{x} \leq 1$ for (d_1, p) is then given by

$$\begin{array}{r} \\ (d_1,p) \end{array} \begin{array}{c} \begin{array}{cccccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 & e_{10} \end{array} \\ \left(\begin{array}{cccccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \end{array} \mathbf{x} \leq 1$$

6.2 Cycle and Chain Formulation

Constraints

As the cycles and chains in the exchange must be disjoint, each node is included in at most one cycle or chain in the exchange.

$$\forall v \in V \quad \mathit{included\_count}(v) \leq 1$$

BILP Conversion

The number of variables n of the MILP is equal to the number of cycles and chains in G^{dir}, which may be exponential in the number of agents.

$$n = |C|$$

Let $\mathbf{x}$ be a column vector consisting of n binary variables, and let the elements of the column vector $\mathbf{c}$ be defined by

$$\forall i \leq n \quad c_i = \mathit{length}(c_i)$$

Let $\mathbf{a}_v^{cnt}$ denote a row vector consisting of n values corresponding to node v. Let the elements of $\mathbf{a}_v^{cnt}$ be defined by

$$\forall i \leq n \quad a_{vi}^{cnt} = \begin{cases} 1 & \text{if } v \text{ is contained in } c_i \\ 0 & \text{otherwise} \end{cases}$$

The constraint then defines the BILP given by

$$\begin{array}{ll} \text{maximize} & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & \forall v \in V \quad \mathbf{a}_v^{cnt}\,\mathbf{x} \leq 1 \end{array} \tag{11}$$

(Abraham et al., 2007, Ch 5.0)

Example Market

The set of all cycles and chains C in $G^{dir\text{-}ex}$ is given by

$$C = \{(t_1, t_2, t_3, t_1), (t_2, t_3, t_2), (d_1, r_1), (d_1, t_1), (d_1, t_1, r_1), (d_1, t_1, t_2), (d_1, t_1, t_2, t_3)\}$$

The constraint $\forall v \in V \quad \mathbf{a}_v^{cnt}\,\mathbf{x} \leq 1$ for t_1 is then given by

$$\begin{array}{c} \quad c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5 \quad c_6 \quad c_7 \\ t_1 \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \mathbf{x} \leq 1 \end{array}$$

7 The Restricted Clearing Problem

The RCP can be solved by methods that extend the edge formulation with additional constraints, or restrict the cycle and chain formulation in the to be selected cycles and chains. This chapter provides derivations for the extensions to the BILP formulations, an introduction to optimizations by incremental formulation approaches, and examples that restrict the length of a cycle to at most 2 and the length of a chain to at most 1.

7.1 Edge Formulation

Consider the edge formulation based on unweighted digraphs for the UCP, as in equation (7). The RCP can be solved by additionally including constraints disallowing the existence of illegal structures in an exchange.

A structure, that is a path, a cycle or a chain, of length ℓ can be disallowed by allowing less than ℓ of its edges to be included simultaneously in the exchange.

Approaches based on disallowing paths, and approaches based on disallowing cycles and chains are considered. The approach based on disallowing paths is expected to be faster in cases that the number of paths of length ℓ is less than the number of cycles and chains of length greater than ℓ. The approach based on disallowing cycles and chains is expected to be faster in cases with a few but long cycles and chains. The approach based on disallowing paths does not solve the most general form of the RCP, as the maximum cycle and chain length are coupled.

7.1.1 Disallowing Paths

By disallowing all paths of length ℓ in G^{dir}, the maximum cycle length is constrained to at most $\ell_1 = \ell$, and the maximum chain length is constrained to at most $\ell_2 = \ell - 1$. Note that this method does not allow ℓ_1 and ℓ_2 to be independent of each other.

Constraints

Let P denote the set of all paths of length ℓ in G^{dir}.

$$P = \{p \mid p \text{ is a path of length } \ell \text{ in } G^{dir}\}$$

A path is disallowed by allowing at most $\ell - 1$ of its edges to be included simultaneously in the exchange.

$$\forall p \in P \quad \mathit{included\_edges}(p) \leq \ell - 1$$

BILP Conversion

Consider the BILP in equation 7. Let $\mathbf{a}_p^{incl}$ denote a row vector consisting of n values corresponding to path p. Let the elements of $\mathbf{a}_p^{incl}$ be defined by

$$\forall i \leq n \quad a_{pi}^{incl} = \begin{cases} 1 & \text{if } e_i \text{ is an edge in } p \\ 0 & \text{otherwise} \end{cases}$$

The BILP extended with respect to equation (7) is then given by

$$\begin{array}{lll} \text{maximize} & \mathbf{c} \cdot \mathbf{x} & \\ \text{subject to} & \forall v \in T & \mathbf{a}_v^{out\text{-}in}\,\mathbf{x} \leq 0 \\ & \forall v \in T \cup R & \mathbf{a}_v^{in}\,\mathbf{x} \leq 1 \\ & \forall v \in D & \mathbf{a}_v^{out}\,\mathbf{x} \leq 1 \\ & \forall p \in P & \mathbf{a}_p^{incl}\,\mathbf{x} \leq \ell - 1 \end{array} \tag{12}$$

(Abraham et al., 2007, inspired by Ch 4.0)

Example Market

The set P of all paths of length 2 in $G^{dir\text{-}ex}$ is given by

$$P = \{(d_1, t_1, r_1), (d_1, t_1, t_2), (t_1, t_2, t_3), (t_2, t_3, t_1), (t_3, t_1, r_1), (t_3, t_1, t_2)\}$$

The constraint $\forall p \in P \;\; \mathbf{a}_p^{incl}\,\mathbf{x} \leq \ell - 1$ for path (d_1, t_1, r_1) is then given by

$$\begin{array}{c} \;\; e_1 \;\; e_2 \;\; e_3 \;\; e_4 \;\; e_5 \;\; e_6 \;\; e_7 \\ (d_1, t_1, r_1) \;\; \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \mathbf{x} \;\leq\; 2 - 1 = 1 \end{array}$$

7.1.2 Disallowing Cycles and Chains

By disallowing the existence of cycles of length greater than ℓ_1 and chains of length greater than ℓ_2, the maximum cycle length is constrained to at most ℓ_1, and the maximum chain length is constrained to at most ℓ_2.

Constraints

Let $C^{>}$ denote the set of all cycles of length greater than ℓ_1, and all chains of length greater than ℓ_2 in G^{dir}.

$$\begin{aligned} C^{>} = \{c \mid\; & c \text{ is a cycle of length greater than } \ell_1, \text{ or} \\ & c \text{ is a chain of length greater than } \ell_2, \text{ in } G^{dir}\} \end{aligned}$$

A cycle or chain $c \in C^{>}$ is disallowed by allowing at most $\mathit{length}(c) - 1$ of its edges to be included simultaneously in the exchange.

$$\forall c \in C^{>} \quad \mathit{included\_edges}(c) \leq \mathit{length}(c) - 1$$

BILP Conversion

Let $\mathbf{a}_c^{incl}$ denote a row vector consisting of n values corresponding to cycle or chain c. Let the elements of $\mathbf{a}_c^{incl}$ be defined by

$$\forall i \leq n \quad a_{ci}^{incl} = \begin{cases} 1 & \text{if } e_i \text{ is an edge in } c \\ 0 & \text{otherwise} \end{cases}$$

The BILP, extended with respect to equation (7), is then given by

$$\begin{array}{lll} \text{maximize} & \mathbf{c} \cdot \mathbf{x} & \\ \text{subject to} & \forall v \in T & \mathbf{a}_v^{out\text{-}in}\, \mathbf{x} \leq 0 \\ & \forall v \in T \cup R & \mathbf{a}_v^{in}\, \mathbf{x} \leq 1 \\ & \forall v \in D & \mathbf{a}_v^{out}\, \mathbf{x} \leq 1 \\ & \forall c \in C^{>} & \mathbf{a}_c^{incl}\, \mathbf{x} \leq \mathit{length}(c) - 1 \end{array} \tag{13}$$

(Abraham et al., 2007, inspired by Ch 4.2)

Example Market

The set $C^{>}$ of all cycles of length greater than 2 and chains of length greater than 1 in $G^{dir\text{-}ex}$ is given by

$$C^{>} = \{(t_1, t_2, t_3, t_1), (d_1, t_1, r_1), (d_1, t_1, t_2), (d_1, t_1, t_2, t_3)\}$$

The constraint $\forall c \in C^{>} \quad \mathbf{a}_c^{incl}\, \mathbf{x} \leq \mathit{length}(c) - 1$ for cycle (t_1, t_2, t_3, t_1) is then given by

$$\begin{array}{r} \\ (t_1, t_2, t_3, t_1) \end{array} \begin{array}{c} \begin{array}{ccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \end{array} \\ \left(\begin{array}{ccccccc} 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array} \right) \end{array} \mathbf{x} \leq 3 - 1 = 2$$

7.2 Cycle and Chain Formulation

Consider the cycle and chain formulation for the UCP, as in equation (11). The RCP can be solved by replacing C by the more restricted $C^{\leq}$.

Constraints

Let $C^{\leq}$ denote the set of all cycles of length at most ℓ_1, and all chains of length at most ℓ_2 in G^{dir}.

$$\begin{aligned} C^{\leq} = \{c \mid\ & c \text{ is a cycle of length at most } \ell_1, \text{ or} \\ & c \text{ is a chain of length at most } \ell_2, \text{ in } G^{dir}\} \end{aligned}$$

BILP Conversion

The BILP is then equal to the BILP of equation (11).

$$\begin{aligned} &\text{maximize} && \mathbf{c} \cdot \mathbf{x} \\ &\text{subject to} && \forall v \in V \quad \mathbf{a}_v^{cnt}\, \mathbf{x} \leq 1 \end{aligned} \tag{14}$$

(Abraham et al., 2007, Ch 5.0)

Example Market

The set $C^{\leq}$ of all cycles of length at most 2 and chains of length at most 1 in $G^{dir\text{-}ex}$ is given by

$$C^{\leq} = \{(t_2, t_3, t_2), (d_1, r_1), (d_1, t_1)\}$$

The constraint $\forall v \in V \quad \mathbf{a}_v^{cnt}\, \mathbf{x} \leq 1$ for t_1 is then given by

$$\begin{array}{c} \\ t_1 \end{array} \begin{array}{c} \begin{array}{ccc} c_1 & c_2 & c_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \end{array} \mathbf{x} \;\leq\; 1$$

7.3 Incremental Formulation Approaches

In general, for the RCP, only a small fraction of the constraints will be binding, or only a small fraction of the binary variables turn out to have non-zero value. Instead of formulating the BILP in its entirety, an incremental formulation approach may be used. This chapter provides possible incremental formulation approaches for various methods. The experiments will not consider these approaches, as the utility of these approaches is dependent on the class of markets that is considered.

The BILP is initiated with a subset of the constraints, or variables, that is columns. Then iteratively the BILP is solved, and constraints that turn out to be binding, or columns that turn out to have potential of improving the objective value, are added. For a BILP, the process of adding constraints or columns must be performed at each node in the branch tree.

Initiation

A good initiation provides the BILP with a subset of constraints or columns, such that the BILP is solvable in reasonable time, and the objective value of its optimal solution is not too far from the objective value of the optimal solution of the BILP in its entirety. Initiation is optional, but may result in a significant speedup.

Extending the Formulation

A good algorithm for determining the constraints or columns that are to be added in each iteration, finds in reasonable time a constraint that is (most) binding, or a column that has the potential to increase the objective (the most). In each iteration, more than one constraint or column may be added, but this may not always be beneficial.

Simply traversing all excluded constraints or columns in order to determine the best choice may not be possible in reasonable time. In these cases a heuristic must be used, that balances between the utility of the found constraint or column, and the computational cost.

(Abraham et al., 2007, Ch 5.2.0)

7.3.1 Edge Formulation

Consider the methods based on the edge formulation of the RCP. The corresponding formulations may contain a large number of constraints. The focus for these formulations is therefore on constraint generation.

Initiation

A simple approach that can be used for both edge formulation methods in providing an initial set of constraints, is selecting at random a set of constraints.

An approach for the method based on disallowing paths, is by only disallowing paths of length $\ell - 1$ that do not have an edge closing the cycle from its head to its tail. This approach focuses the search away from cycles that cannot be in the final solution.

An approach for the method based on disallowing cycles and chains, is by only disallowing cycles and chains, that contain just a few more edges than the constraints allow, as longer cycles and chains are often disallowed implicitly by the other constraints.

Finding Violated Constraints

One approach in determining a violated constraint is to solve the LP corresponding to the BILP of the current iteration. The optimal solution to the LP may contain fractional solutions for the binary variables corresponding to the edges. The edges corresponding to non-zero values define a graph, that can be searched for cycles or chains violating the constraints.

An approach for the method based on disallowing paths is to eliminate all found violating paths in this graph. An approach for the method based on disallowing cycles and chains is to eliminate all found violating cycles and chains in this graph.

(Abraham et al., 2007, Ch 5.2)

An Alternative Branching Scheme

The number of cycles and chains may increase exponentially with respect to the number of nodes, while the number of edges may only increase quadratically with the number of nodes. The optimal solution of the LP corresponding to the BILP, of the current iteration, may contain fractional values. The value of an edge is defined as the sum of these fractional solutions for cycles or chains that contain this edge.

An approach in branching, is to branch on the edge with fractional value closest to 0.5, creating a branch in which this edge is banned, and a branch is which this edge is enforced, in the exchange.

Another approach in branching, is to select the node with the highest number of fractional edges, and distribute these edges evenly according to their value in two sets. In each branch, the corresponding set of edges is banned.

(Glorie, 2014, Ch 3.3.3)

7.3.2 Cycle and Chain Formulation

Consider the method based on the cycle and chain formulation of the RCP. The corresponding formulation may contain a large number of variables. The focus for this formulation is therefore on column generation.

Initiation

An approach for providing an initial set of columns is to iteratively start at a random node in G^{dir}, and try to cover that node and other uncovered nodes in a cycle or chain.

Finding Columns to be Included

An approach in determining a column that has the potential to increase the objective function, is to solve the dual LP corresponding to the BILP, of the current iteration. The columns in the dual LP correspond to the nodes in the market digraph. The price of a cycle or chain is equal to its length, minus the dual-value sum of the nodes that are contained in this cycle or chain.

(Abraham et al., 2007, Ch 5.2.0)

The Pricing Problem

Finding a cycle or chain with positive price is not trivial if simply traversing the excluded cycles and chains is not an option. One approach is to search G^{dir}, with the heuristic that nodes are to be explored in non-decreasing order of their dual-value.

(Abraham et al., 2007, Ch 5.2.1)

7.3.3 Constraint and Column Management

As new constraints or columns are added, existing constraints may not be binding anymore, or existing columns may not have potential to influence the objective value anymore.

By the observation that only a small fraction of the constraints will be binding, or only a small fraction of the binary variables turn out to have non-zero value, one might choose to delete constraints that are not binding or columns with a low price. Columns on which the algorithm has branched, or that have a non-zero value in the solution of the corresponding LP, cannot be deleted. One must be careful however, as deleting the wrong constraints or columns might result in a significant execution time penalty.

(Abraham et al., 2007, Ch 5.2.4)

8 Experiments

This chapter provides an overview and the results of the performed measurements.

8.1 Considered Formulations

An overview of the used methods, abbreviations and reference equations is provided in the following table.

Formulation	Abbreviation	Equation
UCP ...		
edge formulation ...		
unweighted digraphs	unrestricted edge digraph	(7)
weighted undirected bipartite graphs	unrestricted edge bipartite graph	(10)
cycle and chain formulation	unrestricted cycle and chain	(11)
RCP ...		
edge formulation ...		
disallowing paths	restricted edge paths	(12)
disallowing cycles and chains	restricted edge cycles and chains	(13)
cycle and chain formulation	restricted cycle and chain	(14)

8.2 Implementation

Building an implementation of the various solvers is not a trivial task. This section provides an overview of experiences and attention points regarding implementation. Appendix A provides the source code for the most important functionality.

(BI)LP Solver

Mathworks provides the Optimization Toolbox for Matlab. The available functions include solvers for either LPs or MILPs. The user is allowed to specify a large variety of optimization settings. We have performed measurements altering a single optimization setting with respect to the defaults at a time, for a variety of markets and formulations. The setting that disabled LP preprocessing lowered the execution time by about 20%, and turned out to be the only setting that significantly lowered the execution time. Subsequent research has turned out that these solvers were too unstable, as changing a parameter even slightly in the market generation process could influence the execution time of the solvers by several orders of magnitude.

Prior research has often used the IBM ILOG CPLEX Optimization Studio, while programming in languages like C(++). This software package also contains a plugin for Matlab, that is written in the Matlab programming language. The available functions include solvers for either LPs, MILPs and even BILPs. These solvers turned out to be a lot more stable with respect to different market generation parameters, but were slightly slower than the solvers that were equipped with the Matlab Optimization Toolbox, and did not allow essential optimization options.

Sparsity

The RCP formulations, for considerably large or dense markets, have the potential of being extremely memory demanding when described by full matrices. Sparse matrices solve this problem, but its use requires special care to be efficient. Elements of sparse matrices should not be set using direct indexing, but should rather be created given all indices in advance.

Overhead

Rather simple built-in functionality often turned out to be significantly slower than a custom implementation. Even provided functionality that perform an identical operation on each element of a structure, that therefore may be assumed to be faster than a naive iterative approach, turned out to be slow. Most of the overhead can be attributed to input validation phases that were not relevant to the application at hand. Matlab additionally generates overhead in function calls and the use of dynamic structures like stacks.

Structure Generation

For large or dense markets, the number of structures that need to be enumerated may be considerable. Implementations of naive methods such as an adapted depth first search, or even special methods turn out to be relatively slow due to the mentioned overhead in function calls and stacks, both of which are often required in these methods. It turns out that an adapted version of bread first search, that each iteration operates on the entire queue using matrix operations, is able to offer the required performance.

Correctness

Appendix A provides the markets that were used to verify correctness. A part of these markets were designed by someone that did not have prior knowledge of methods for clearing markets. The results were compared for different solvers and different restrictions. Confidence in the correctness of the implemented solvers is increased by the observation that there is a small chance that two different solvers produce an identical wrong solution.

Graphs

Matlab provides functionality for working with several kinds of graphs. These implementations turned out to be rather fast, though special care needs to be taken to extract almost all of the required information at once to reduce overhead.

8.3 System

All experiments were performed using the following system

System	MacBook Pro (Mid 2015)
Processor	2,8 GHz (Intel Core i7)
Memory	16 GB (1600 MHz DDR3)
Operating System	macOS Sierra (10.12.4)
Language	Matlab R2016b (9.1.0.441655, 64-bit)
(BI)LP Solver	IBM ILOG CPLEX Optimization Studio (V12.7.1, Matlab)

8.4 Considered Markets

A market may contain a number of properties, such as non-uniform match probabilities, that influence a quantity under consideration. The investigation of a quantity under consideration as a function of a single market property may yield insignificant relations for a too wide class of markets. The investigation is therefore focused on a specific class of markets, for which the properties that are not under investigation are uniform.

The market properties that will be investigated are the number of agents, and the match probability that is uniform over all agents. Randomly generating markets according to these properties should improve uniformity over other market properties. To reduce the number of variables, the investigated markets regarding execution time will consist of traders only, as donors and receivers are not expected to have a great influence in this.

8.5 Measurements

A measurement, for a class of markets with an identical number of agents and match probability, consists of an evaluation of the quantity under consideration for 10 randomly selected markets in this class. The distribution defined by these evaluations tends to be normally distributed, with occasionally a significant outlier. A statistic obtained from these evaluations must therefore be resistant to outliers. The statistics mean and standard deviation failed to provide the required robustness. The statistics median and mad (median average distance) are therefore adopted as the preferred statistics.

8.6 Execution Time

The execution time, of a function returning an optimal exchange for a given market, is determined by the time spent generating the formulation and the time spent solving the formulation. The investigation will be focused on the **core execution time**, that is the time spent solving the formulation. This choice has been made as little transparency in the Matlab framework implementation, as well as the use of presumable suboptimal structure generation, may affect the time spent generating the formulation significantly.

8.6.1 Contour

The expectation is that the core execution time is a monotonically increasing function of both the number of agents and the match probability. The focus of this section is the investigation of the 1.0 seconds core execution time contour line of this function, determined for each of the considered formulations.

Note that for any number of agents, the core execution time can be made arbitrarily small, by making the match probability arbitrarily small. The contour consequently exists for an unrestricted domain. The domain to be investigated must therefore be bounded by either a maximum number of agents, or by a minimum match probability. The latter turned out to be the more convenient choice, therefore the investigation is restricted to the domain

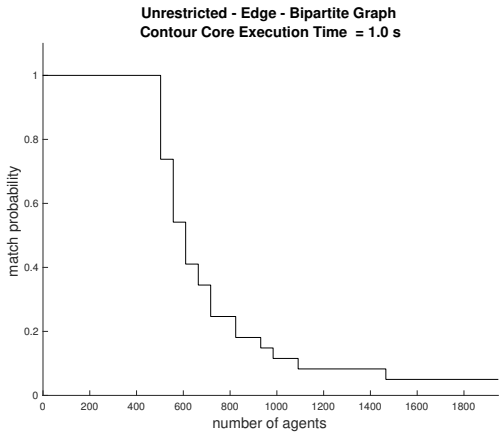
$$0.05 \leq \textit{match probability} \leq 1.0$$

The contour is determined by an algorithm that walks along the contour, iteratively increasing the number of agents, if the median core execution time of the previous measurement is smaller than 1.0, or decreasing the match probability, otherwise. The initial match probability is 1.0 which decreases in 30 linearly spaced steps to 0.05, after which the algorithm terminates. The range of the number of agents is manually adjusted to increase in 30 linearly spaced steps, in the curved region. The obtained contours can be found in Figure 8.

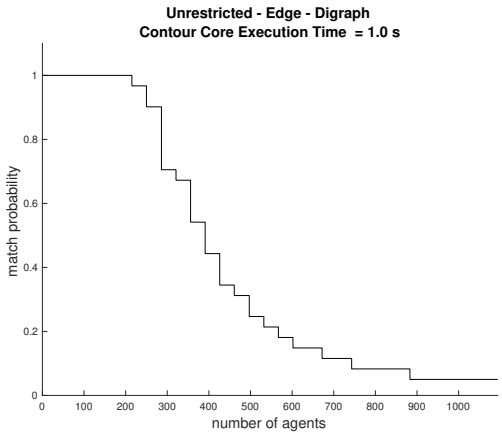
Consider the contours for the unrestricted edge formulations in Figures 8(a) and 8(b). The bipartite graph formulation outperforms the digraph formulation significantly. The contours of both formulations seem to be comparable, except that the early descent of the bipartite graph formulation seems steeper.

Consider the contours for the restricted formulations in Figures 8(c), 8(d) and 8(e). All measurements were performed restricting the maximum cycle length to be at most 2. The cycle and chain formulation outperforms the edge formulations significantly, and the edge paths formulation outperforms the edge cycles and chains formulation significantly. Concerning the latter comparison, note that the edge cycles and chains formulation is less restricted in solving the RCP, and will perform better as the maximum allowed cycle length increases.

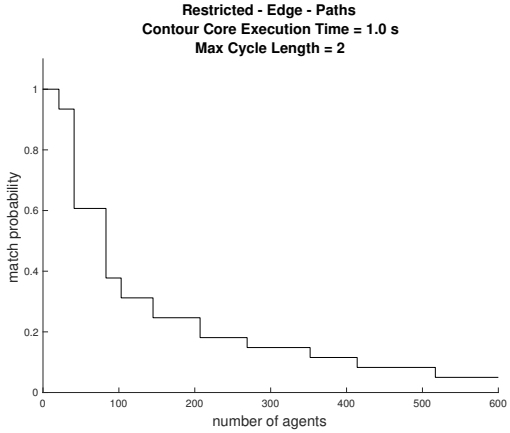
The investigation will be focused on the unrestricted edge bipartite graph formulation and the restricted cycle and chain formulation, as these formulations outperform formulations solving similar problems.



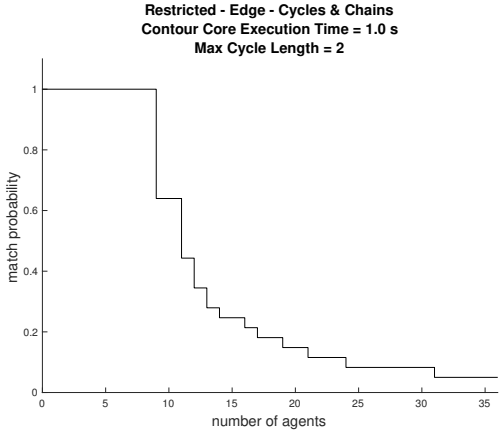
(a)



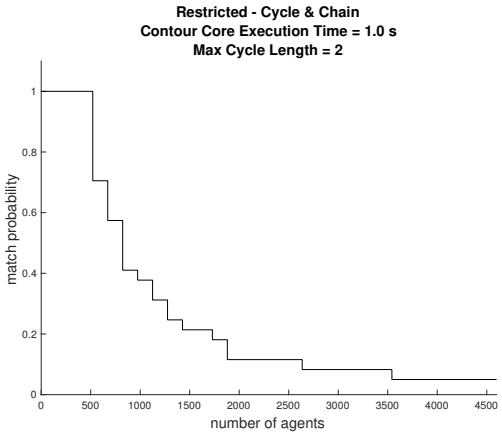
(b)



(c)



(d)



(e)

Figure 8

A core execution time measurement smaller than 0.1 seconds is considered to be influenced by noise, and a core execution time measurement greater than 1.0 seconds is considered to take too long to measure. The forthcoming measurements must therefore be performed in the region between the 0.1 and 1.0 seconds core execution time contours. These contours and chosen measurement regions can be found in Figure 9. The 0.1 and 1.0 seconds core execution time contours are shown by dotted lines, while the measurement domains are shown as solid lines.

8.6.2 Match Probability

To investigate the core execution time as a function of the match probability, the number of agents should be kept constant. The measurement domains are indicated by the vertical lines in Figure 9. The obtained results can be found in Figure 10.

Consider the plots for the unrestricted formulations in Figures 10(a), 10(b) and 10(c). For a constant 450 agents and 615 agents, the increase seems to be close to linear. For a constant 779 agents the relation seems to experience a sudden significant increase. This phenomenon remains unexplained.

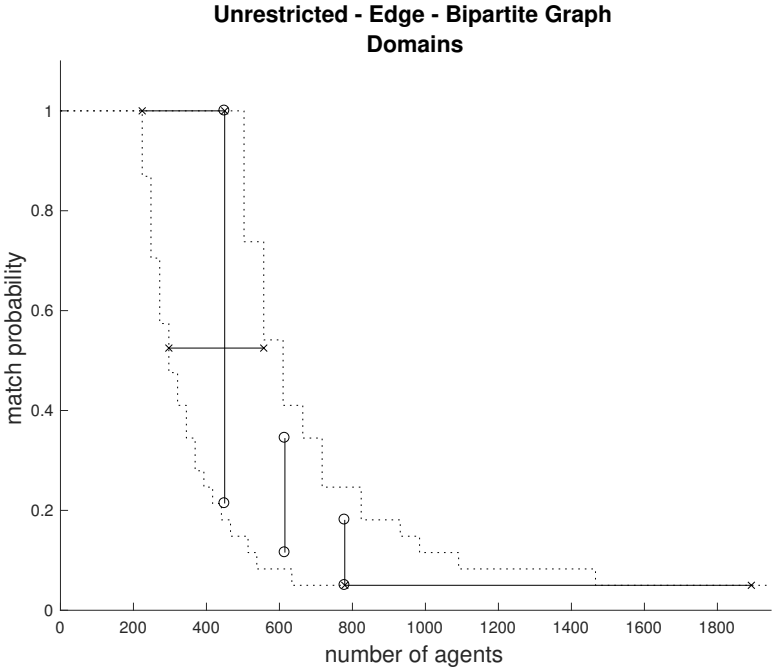
Consider the plots for the restricted formulations in Figures 10(d), 10(e) and 10(f). Over the full range, the increase seems to be linear. The increase as well as the deviations seems to be highly unstable. This phenomenon remains unexplained as well.

8.6.3 Number of Agents

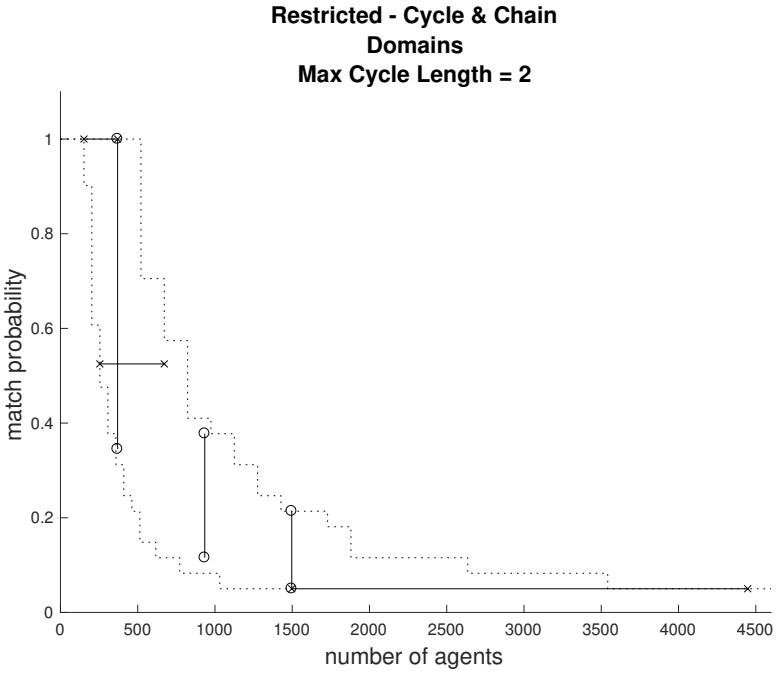
To investigate the core execution time as a function of the number of agents, the match probability should be kept constant. The measurement domains are indicated by the horizontal lines in Figure 9. The measurements can be found in Figure 11.

Consider the plots for the unrestricted formulations in Figures 11(a), 11(b) and 11(c). Taking the domain of the horizontal axis into account, the increase seems to be most likely a second or third degree polynomial. The plot for match probability equal to 1.0 is outstanding, as it seems that markets with a certain number of agents have a higher execution time than others. This phenomenon remains unexplained.

Consider the plots for the restricted formulations in Figures 11(d), 11(e) and 11(f). All measurements were performed restricting the maximum cycle length to be at most 2. The increase is most likely second or third degree polynomial, or exponential. The decline visible for the plot with match probability 0.05 is outstanding and has occurred also in unreported measurements. Also here when the match probability is 1.0, the same conclusions can be made as for the unrestricted variants.

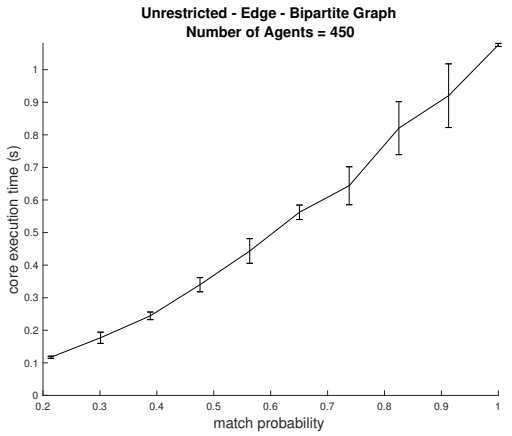

Unrestricted - Edge - Bipartite Graph
Domains
match probability
number of agents
0
0.2
0.4
0.6
0.8
1
0
200
400
600
800
1000
1200
1400
1600
1800

(a)

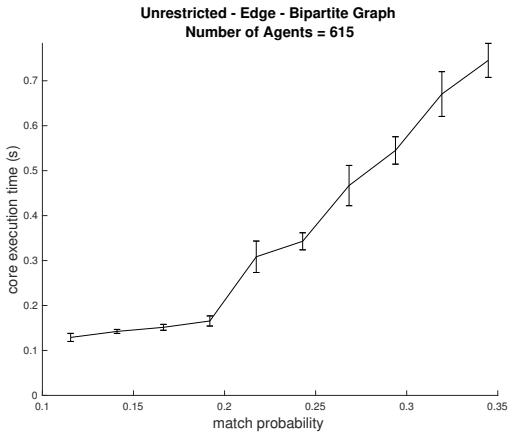

Restricted - Cycle & Chain
Domains
Max Cycle Length = 2
match probability
number of agents
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000
3500
4000
4500

(b)

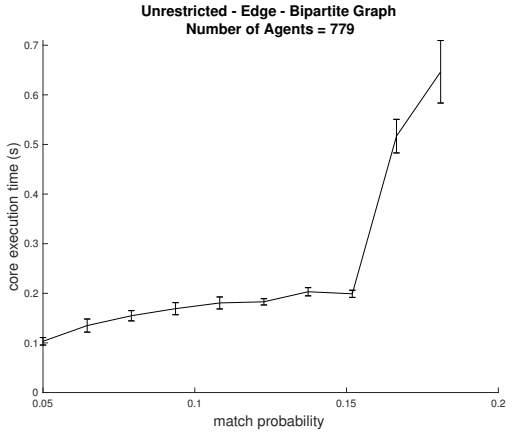
Figure 9



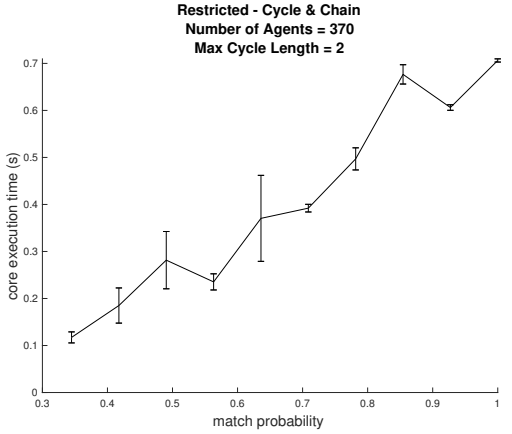
(a)



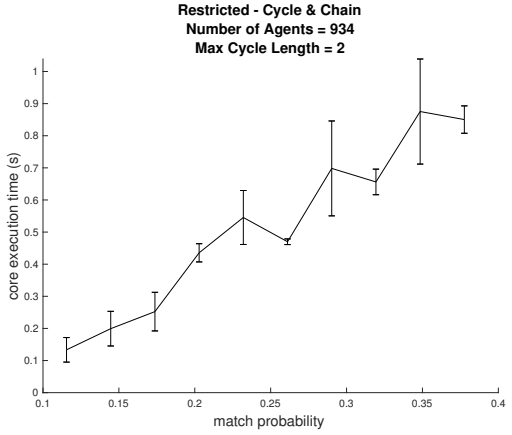
(b)



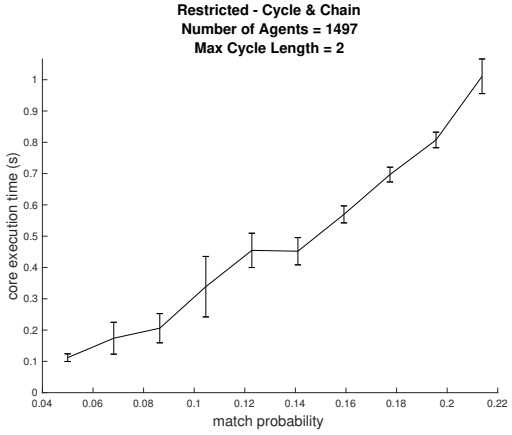
(c)



(d)

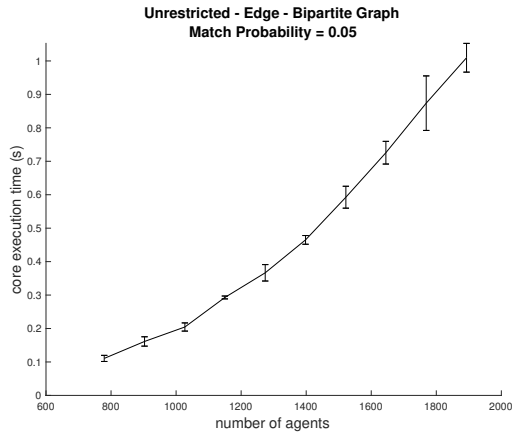


(e)

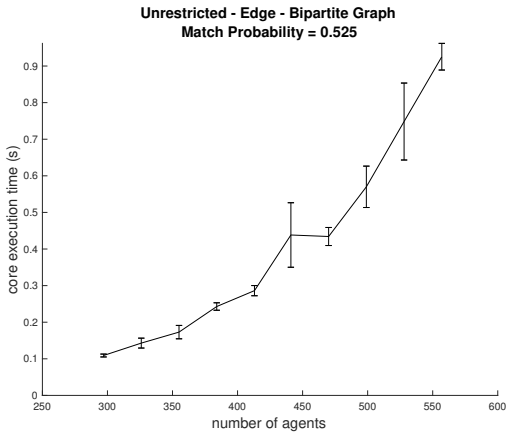


(f)

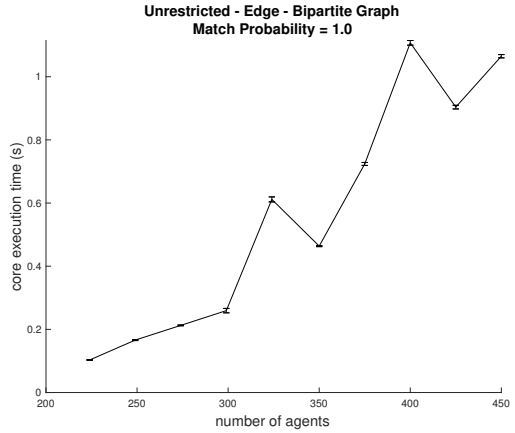
Figure 10



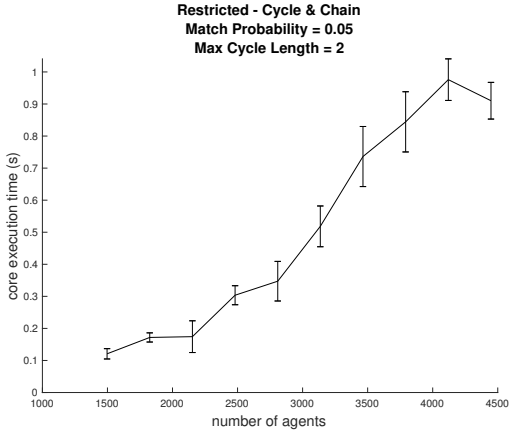
(a)



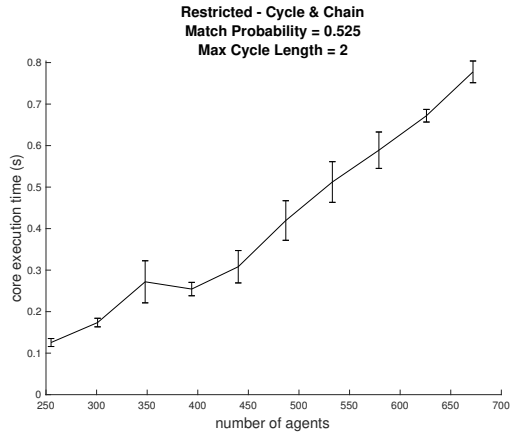
(b)



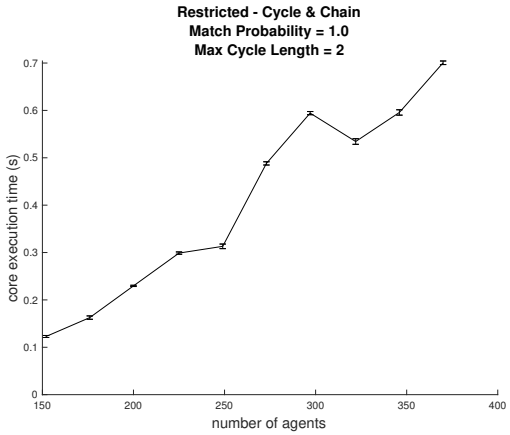
(c)



(d)



(e)



(f)

Figure 11

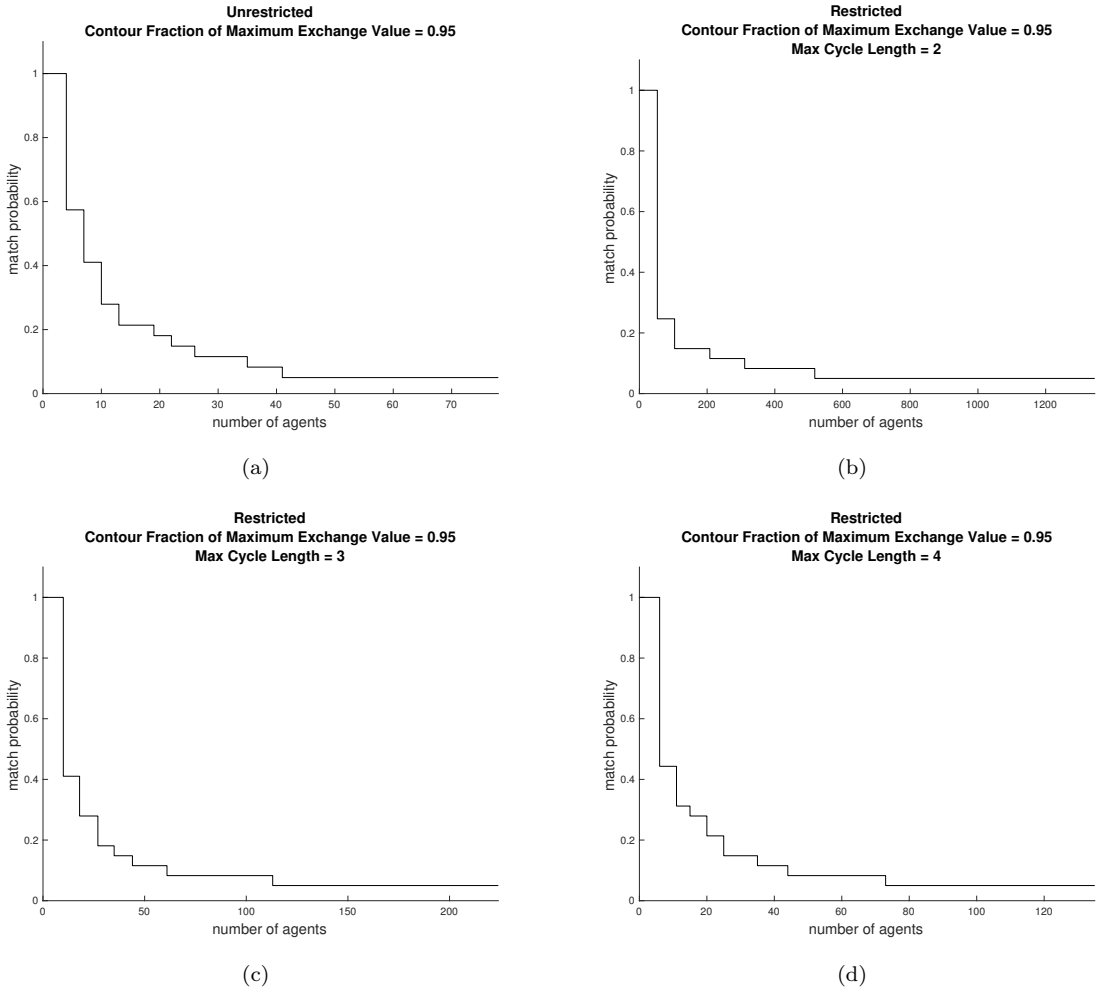
8.7 Exchange Value

8.7.1 Contour

The expectation is that the exchange value is a monotonically increasing function of both the number of agents and the match probability. The focus of this section is the investigation of the 0.95 fraction of the maximum exchange value contour line of this function, for a selected number of maximum cycle lengths.

Following the same reasoning as in the section concerning the execution time, the investigation is restricted to the domain

$$0.05 \leq \textit{match probability} \leq 1.0$$



(a) (b) (c) (d)

Figure 12

The contour line is obtained in a similar way as in the section concerning the execution time. The obtained results can be found in Figure 12.

Observe that the maximum cycle length has a significant influence on, with equal match probabilities, the number of agents needed for 0.95 fraction of the maximum exchange value. Also note that for an increasing maximum cycle length, the contour quickly converges to the contour for the unrestricted cycle length.

The investigation will now be focused on the unrestricted cycle length.

Fractions smaller than 0.05 or greater than 0.95 of the maximum exchange value are considered to be influenced by the specific random generation. The forthcoming measurements must therefore be performed in the region between the 0.05 and 0.95 fraction of the maximum exchange value. These contours and chosen measurement regions can be found in Figure 13.

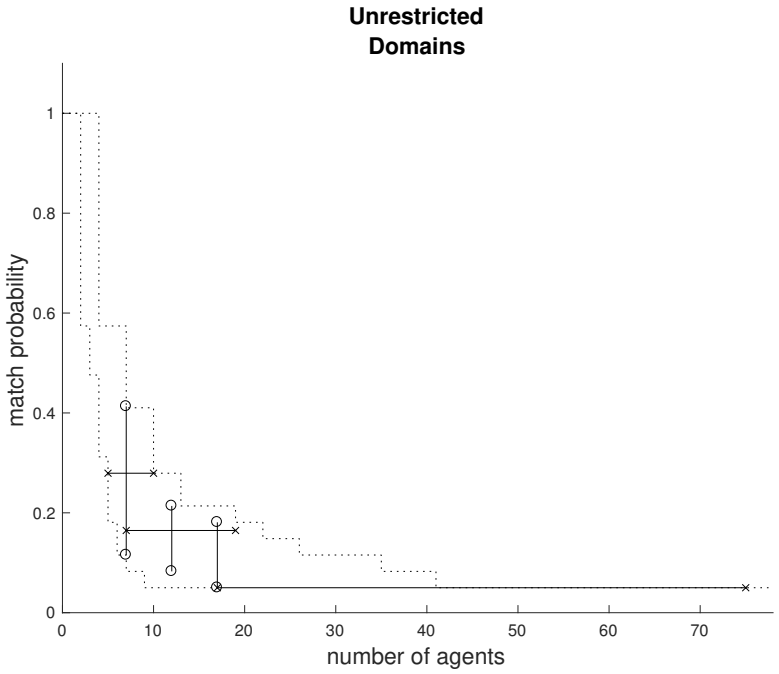
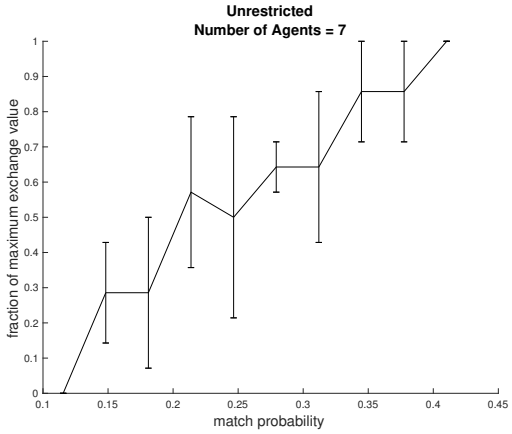


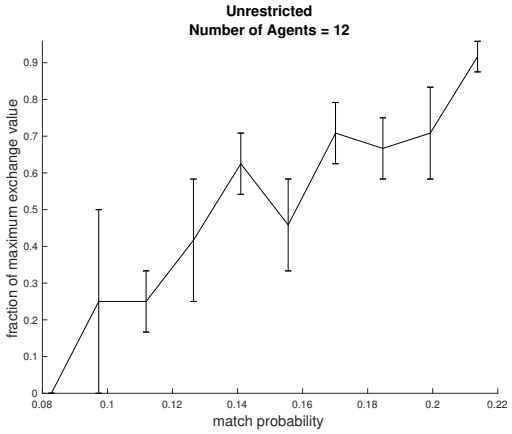
Figure 13

8.7.2 Match Probability

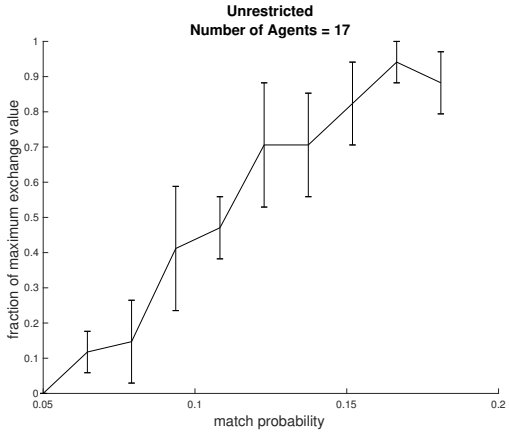
To investigate the exchange value as a function of the match probability, the number of agents should be kept constant. The measurement domains are indicated by the vertical lines in Figure 13. The measurements can be found in Figure 14. Although there are significant deviations, all plots seem to conclude linear growth.



(a)



(b)

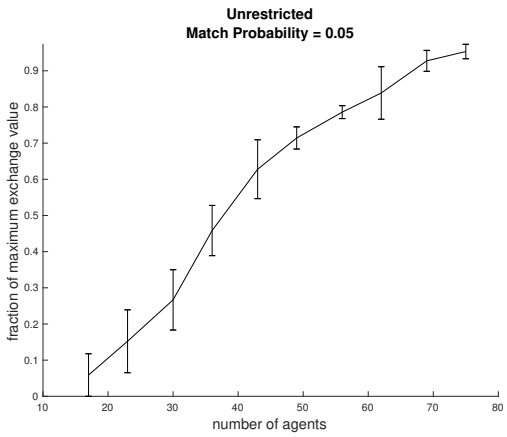


(c)

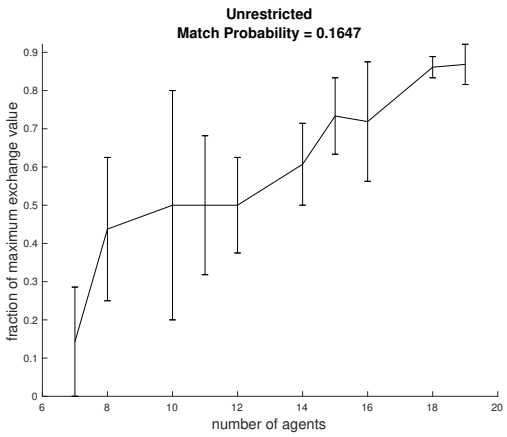
Figure 14

8.7.3 Number of Agents

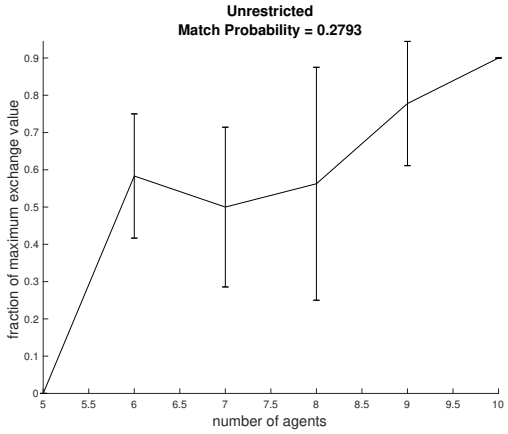
To investigate the exchange value as a function of the number of agents, the match probability should be kept constant. The measurement domains are indicated by the horizontal lines in Figure 13. The measurements can be found in Figure 15. The relations seem to be linear in some ranges, but seem to experience nods in the full range.



(a)



(b)



(c)

Figure 15

8.8 Donor Influence

This section investigates the optimal exchange value, as a function of a number of additional donors to an existing market. The investigation is restricted to the UCP, and an existing market with 1000 traders.

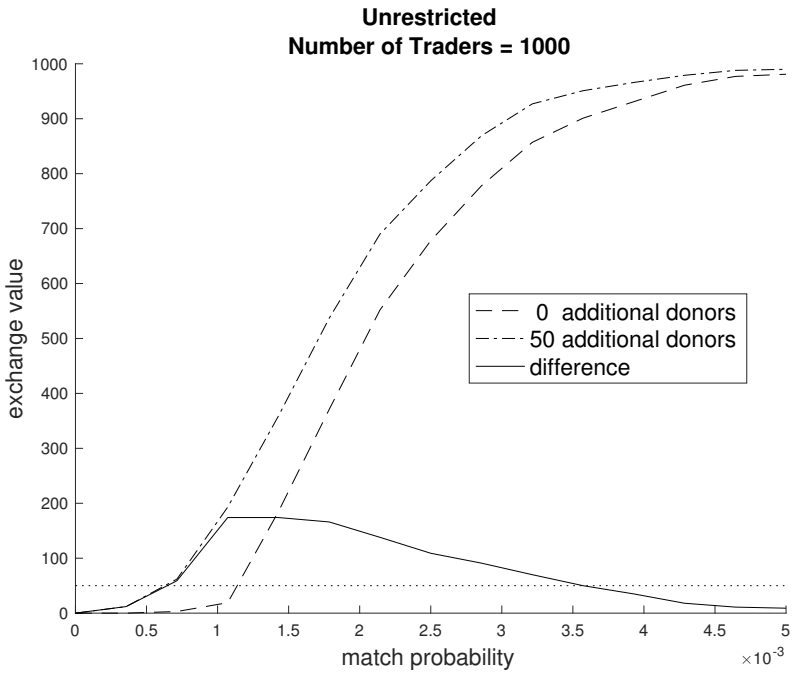


Figure 16

A sufficient match probability, to achieve the maximum exchange value of 1000, is 0.005. The market is extended with only 50 additional donors, as to minimize possible side effects. Figure 16 compares the existing market with the extended market, for match probabilities between 0 and 0.005. Note the range in which the difference in optimal exchange value exceeds the number of additional donors, to a maximum of 174 for an interpolated match probability of 0.00125. Further research indicates that the location of this maximum is relatively independent of the additional number of donors.

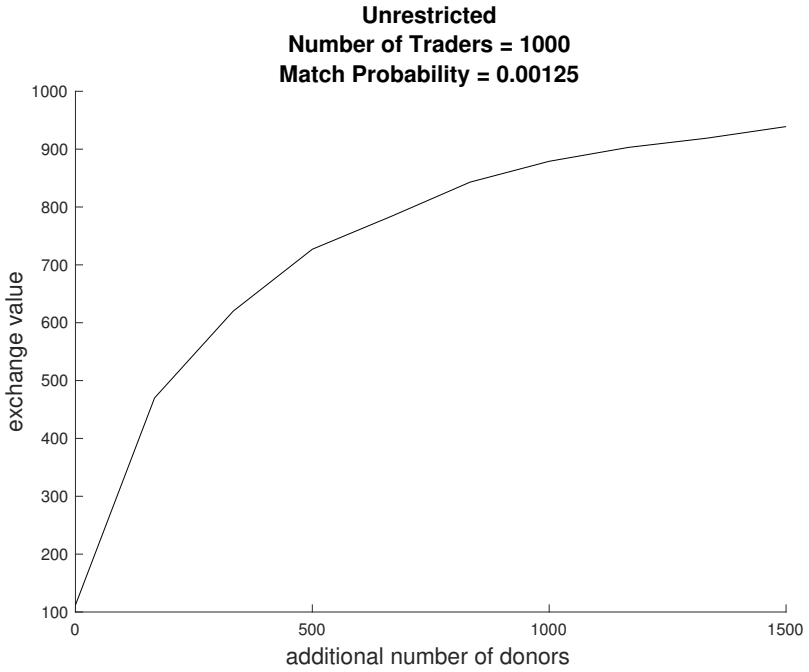


Figure 17

Figure 17 shows the optimal exchange value as a function of the number of additional donors, for the optimal match probability of 0.00125. As up to 50 additional donors did not reveal a clear relation, up to 1500 additional donors were added. The trend is declining and approaches the maximum exchange value of 1000. Note that more than 1000 additional donors does not guarantee an exchange value of 1000.

9 Conclusion

This thesis has proposed clearing methods that besides trader agents, also account for donor and receiver agents. There are a lot of things to consider in the implementation of the solvers, such as the correct use of sparse matrices, structure generation, and general overhead. The experiments have shown that for the UCP, the method based on bipartite graph edge formulation performs best. For the RCP, the method based on the cycle and chain formulation performs best. There is still some uncertainty in how quantities like the execution time and optimal exchange value increase as a function of market properties like the number of agents and the match probability. For certain instances, the solvers may experience execution times that are orders of magnitude different from the median. Regarding the donor influence, this research has discovered that the donor influence shows a maximum over the match probability, and additional donors are of decreasing importance when more donors are added.

10 Future Research

The considered markets in this research have a uniform match probability, which is in general not true for all markets. This research provides a basis, and can be extended towards other classes of markets.

The considered methods in this research solve the clearing problems to optimality, and have been based on (BI)LP methods. A market can be too large to be solved with the presented methods, and may accept some loss of optimality. Approximate algorithms may therefore be also be considered in future research. Future research may also be concerned with algorithms specifically designed for clearing barter exchange markets.

The performance of the presented methods can be increased by the use of parallel execution, especially in the structure generation and formulation solver regions.

Future research may consider incremental formulation approaches and general purpose optimizations, as well as more market properties such as a non-uniform match probability.

References

Abraham, D. J., Blum, A., and Sandholm, T. (2007). Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges. In *Proceedings of the 8th ACM conference on Electronic commerce*, pages 295–304. ACM.

Cormen, T. H. (2009). *Introduction to algorithms*. MIT press.

Glorie, K. (2014). *Clearing Barter Exchange Markets: Kidney Exchange and Beyond*. PhD thesis, Erasmus Research Institute of Management (ERIM).

Klee, V. and Minty, G. J. (1970). How good is the simplex algorithm. Technical report, University of Washington Seattle Department of Mathematics.

Mahajan, M. (2010). Matchings in graphs. `TheInstituteofMathematicalSciences.http://www.imsc.res.in/~meena/matching/lecture5.pdf`.

A Source Code

This appendix provides source code for the most important functionality. Section A.1 contains source code for the function `restricted_cycle_chain_solver`, which is representative for the cycle and chain formulation solvers. Section A.2 contains source code for the function `restricted_edge_cycles_chains_solver`, which is representative for the edge formulation solvers. Section A.3 contains source code for the function `get_cycles`, which is representative for the structure generators. The full source code can be obtained from `https://github.com/sietzehouwink/bachelor-project`.

A.1 restricted_cycle_chain_solver.m

```
function [exchange_digraph, exchange_value, timed_out, core_exec_time] = restricted_cycle_chain_solver(market_digraph, max_nr_edges_cycle, max_nr_edges_chain, timeout_find_cycles_chains, timeout_solver, cplex_options)
    % Get structures.
    [structures, timed_out] = get_cycles_chains(market_digraph, 0, max_nr_edges_cycle, 0, max_nr_edges_chain, timeout_find_cycles_chains);
    if timed_out
        [exchange_digraph, exchange_value, core_exec_time] = set_timed_out();
        return;
    end

    % Get BILP.
    [weight_vector, inequality_matrix, inequality_vector] = get_BILP(market_digraph, structures);

    % Solve BILP.
    [setting_structures, exchange_value, timed_out, core_exec_time] = BILP_solver(weight_vector, inequality_matrix, inequality_vector, [], [], timeout_solver, cplex_options);
    if timed_out
        [exchange_digraph, exchange_value, core_exec_time] = set_timed_out();
        return;
    end

    % Get exchange digraph.
    exchange_digraph = get_exchange_digraph(market_digraph, structures, setting_structures);
end

function [weight_vector, inequality_matrix, inequality_vector] = get_BILP(market_digraph, structures)
    weight_vector = structures_to_nr_edges(structures);
    [inequality_matrix, inequality_vector] = get_node_containment_constraints(market_digraph, structures);
end

function [inequality_matrix, inequality_vector] = get_node_containment_constraints(market_digraph, cycles_chains)
    indices_nodes = vertcat(cycles_chains{:});
    nr_nodes_cycles_chains = structures_to_nr_nodes(cycles_chains);
    indices_cycles_chains = repelem(1:length(cycles_chains), nr_nodes_cycles_chains);
    inequality_matrix = sparse(indices_nodes, indices_cycles_chains, 1, numnodes(market_digraph), length(cycles_chains));
    inequality_matrix(inequality_matrix > 1) = 1; % Duplicates in sparse generation automatically sum.
    inequality_vector = ones(numnodes(market_digraph), 1);
end
```

```
36
37 function [nr_nodes_structures] = structures_to_nr_nodes(structures)
38     nr_nodes_structures = zeros(length(structures), 1);
39     for idx = 1:length(structures)
40         structure = structures{idx};
41         nr_nodes_structures(idx) = length(structure);
42     end
43 end
44
45 function [exchange_digraph] = get_exchange_digraph(market_digraph, cycles_chains,
       setting_cycles_chains)
46     exchange_cycles_chains = cycles_chains(logical(setting_cycles_chains));
47     [start_nodes, end_nodes] = structures_to_edges(exchange_cycles_chains);
48     adjacency_matrix = sparse(start_nodes, end_nodes, 1, numnodes(market_digraph), numnodes(
           market_digraph));
49     adjacency_matrix(adjacency_matrix > 1) = 1;  % Duplicates in sparse generation
           automatically sum.
50     exchange_digraph = digraph(adjacency_matrix);
51     exchange_digraph.Nodes = market_digraph.Nodes;
52 end
```

A.2 restricted_edge_cycles_chains_solver.m

```
1  function [exchange_digraph, exchange_value, timed_out, core_exec_time] =
       restricted_edge_cycles_chains_solver(market_digraph, max_nr_edges_cycle,
       max_nr_edges_chain, timeout_find_cycles_chains, timeout_solver, cplex_options)
2      % Get BILP.
3      [weight_vector, inequality_matrix, inequality_vector, timed_out] = get_BILP(
           market_digraph, max_nr_edges_cycle, max_nr_edges_chain, timeout_find_cycles_chains);
4      if timed_out
5          exchange_digraph = NaN; exchange_value = NaN; core_exec_time = NaN;
6          return;
7      end
8
9      % Solve BILP and act on timeout.
10     [setting_edges, exchange_value, timed_out, core_exec_time] = BILP_solver(weight_vector,
           inequality_matrix, inequality_vector, [], [], timeout_solver, cplex_options);
11     if timed_out
12         exchange_digraph = NaN; exchange_value = NaN; core_exec_time = NaN;
13         return;
14     end
15
16     % Convert to exchange digraph.
17     exchange_digraph = get_exchange_digraph(market_digraph, setting_edges);
18 end
19
20 function [weight_vector, inequality_matrix, inequality_vector, timed_out] = get_BILP(
       market_digraph, max_nr_edges_cycle, max_nr_edges_chain, timeout_find_cycles_chains)
21     % Get cycle and chain constraints.
22     [inequality_matrix_1, inequality_vector_1, timed_out] =
           get_disallowed_cycles_chains_constraints(market_digraph, max_nr_edges_cycle,
           max_nr_edges_chain, timeout_find_cycles_chains);
23     if timed_out
24         weight_vector = NaN; inequality_matrix = NaN; inequality_vector = NaN;
25         return;
26     end
27
28     % Get other constraints.
29     [inequality_matrix_2, inequality_vector_2] = get_unrestricted_edge_digraph_constraints(
           market_digraph);
30
31     % Construct results.
```

```
32      weight_vector = market_digraph.Edges.Weight;
33      inequality_matrix = [inequality_matrix_1; inequality_matrix_2];
34      inequality_vector = [inequality_vector_1; inequality_vector_2];
35  end
36
37  function [inequality_matrix, inequality_vector, timed_out] =
        get_disallowed_cycles_chains_constraints(market_digraph, max_nr_edges_cycle,
        max_nr_edges_chain, timeout)
38      % Get cycles and chains.
39      [cycles_chains, timed_out] = get_cycles_chains(market_digraph, max_nr_edges_cycle+1, Inf
            , max_nr_edges_chain+1, Inf, timeout);
40      if timed_out
41          inequality_matrix = NaN; inequality_vector = NaN;
42          return;
43      end
44
45      % Construct results using sparsity.
46      nr_edges_cycles_chains = structures_to_nr_edges(cycles_chains);
47      indices_cycles_chains = repelem(1:length(cycles_chains), nr_edges_cycles_chains);
48      [start_nodes, end_nodes] = structures_to_edges(cycles_chains);
49      indices_edges = findedge(market_digraph, start_nodes, end_nodes);
50      inequality_matrix = sparse(indices_cycles_chains, indices_edges, 1, length(cycles_chains
            ), numedges(market_digraph));
51      inequality_vector = nr_edges_cycles_chains - 1;
52  end
53
54  function [exchange_digraph] = get_exchange_digraph(market_digraph, setting_edges)
55      exchange_edges = market_digraph.Edges(logical(setting_edges), :);
56      exchange_digraph = digraph(exchange_edges, market_digraph.Nodes);
57  end
```

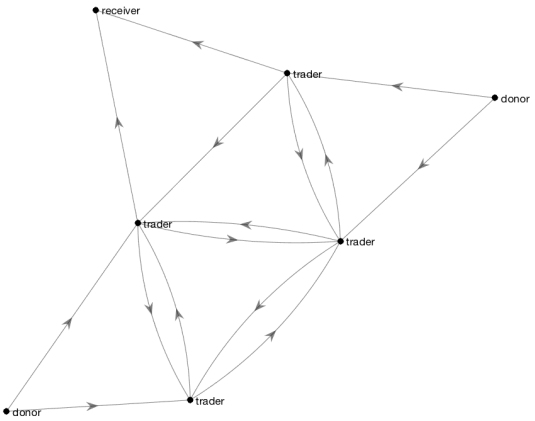
A.3 get_cycles.m

```
1   function [cycles, timed_out] = get_cycles(market_digraph, source_nodes, min_nr_edges,
        max_nr_edges, timeout)
2       [min_nr_edges, max_nr_edges] = tighten_bounds(market_digraph, min_nr_edges, max_nr_edges
            );
3       if min_nr_edges > max_nr_edges
4           cycles = {}; timed_out = false;
5           return;
6       end
7       [cycles, timed_out] = get_cycles_tight_bounds(market_digraph, source_nodes, min_nr_edges
            , max_nr_edges, timeout);
8   end
9
10  % Iteratively: find all cycles that includ source node, then remove source node.
11  function [cycles, timed_out] = get_cycles_tight_bounds(market_digraph, source_nodes,
        min_nr_edges, max_nr_edges, timeout)
12      timer = tic;
13      adjacency_matrix = full(adjacency(market_digraph));
14      cycles = cell(length(source_nodes), 1);
15      for idx = 1:length(source_nodes)
16          source_node = source_nodes(idx);
17          cycles{idx} = get_cycles_source_node(source_node, adjacency_matrix, min_nr_edges,
                max_nr_edges, timeout-toc(timer));
18          adjacency_matrix = remove_node(adjacency_matrix, source_node);
19          if toc(timer) > timeout
20              cycles = NaN; timed_out = true;
21              return;
22          end
23      end
24      cycles = vertcat(cycles{:});
```

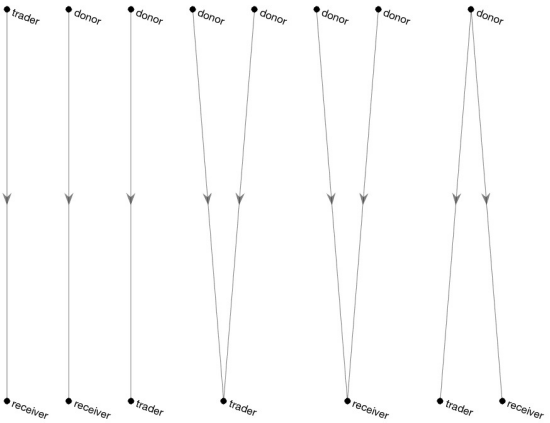
```
25      timed_out = false;
26  end
27
28  function [adjacency_matrix] = remove_node(adjacency_matrix, node)
29      adjacency_matrix(node, :) = 0;
30      adjacency_matrix(:, node) = 0;
31  end
32
33  function [min_nr_edges, max_nr_edges] = tighten_bounds(market_digraph, min_nr_edges,
        max_nr_edges)
34      min_nr_edges = max(min_nr_edges, 2);
35      max_nr_edges = min(max_nr_edges, numnodes(market_digraph));
36  end
37
38  % Iteratively: add one edge to paths, close cycles.
39  function [cycles] = get_cycles_source_node(source_node, adjacency_matrix, min_nr_edges,
        max_nr_edges, timeout)
40      timer = tic;
41      cycles = cell(max_nr_edges-1, 1);
42      paths = (source_node);
43      for nr_edges = 1:max_nr_edges-1
44          paths = add_edge_to_paths(paths, adjacency_matrix);
45          if isempty(paths)
46              break;
47          elseif nr_edges+1 >= min_nr_edges
48              cycles{nr_edges} = close_cycles(source_node, paths, adjacency_matrix);
49          end
50          if toc(timer) > timeout
51              cycles = NaN;
52              return;
53          end
54      end
55      cycles = vertcat(cycles{:});
56  end
57
58  % Finds cycles obtained from adding one edge between path(end) and path(1).
59  function [cycles] = close_cycles(source_node, paths, adjacency_matrix)
60      last_paths = paths(:, end);
61      predecessors_start = find(adjacency_matrix(:,source_node));
62      closing_paths = paths(ismember(last_paths,predecessors_start), :);
63      if isempty(closing_paths)
64          cycles = {};
65          return;
66      end
67      cycles = [closing_paths repelem(source_node,size(closing_paths,1),1)];
68      cycles = matrix_rows_to_cells(cycles);
69  end
```

A Test Cases

This appendix provides illustrations for some of the test cases. Illustrations for the remaining test cases can be obtained from `https://github.com/sietzehouwink/bachelor-project`.



(a)



(b)

Figure 18