Nash Equilibria, Gale Strings, and Perfect Matchings

Julian Merschen

London School of Economics and Political Science

PhD

March 2012

Abstract

This thesis concerns the problem 2-NASH of finding a Nash equilibrium of a bimatrix game, for the special class of so-called "hard-to-solve" bimatrix games. The term "hard-to-solve" relates to the exponential running time of the famous and often used Lemke–Howson algorithm for this class of games. The games are constructed with the help of dual cyclic polytopes, where the algorithm can be expressed combinatorially via labeled bitstrings defined by the "Gale evenness condition" that characterise the vertices of these polytopes.

We define the combinatorial problem "Another completely labeled Gale string" whose solutions define the Nash equilibria of any game defined by cyclic polytopes, including the games where the Lemke–Howson algorithm takes exponential time. We show that "Another completely labeled Gale string" is solvable in polynomial time by a reduction to the "Perfect matching" problem in Euler graphs. We adapt the Lemke–Howson algorithm to pivot from one perfect matching to another and show that again for a certain class of graphs this leads to exponential behaviour. Furthermore, we prove that completely labeled Gale strings and perfect matchings in Euler graphs come in pairs and that the Lemke–Howson algorithm connects two strings or matchings of opposite signs.

The equivalence between Nash Equilibria of bimatrix games derived from cyclic polytopes, completely labeled Gale strings, and perfect matchings in Euler Graphs implies that counting Nash equilibria is #P-complete. Although one Nash equilibrium can be computed in polynomial time, we have not succeeded in finding an algorithm that computes a Nash equilibrium of opposite sign. However, we solve this problem for certain special cases, for example planar graphs. We illustrate the difficulties concerning a general polynomial-time algorithm for this problem by means of negative results that demonstrate why a number of approaches towards such an algorithm are unlikely to be successful.

Declaration

I certify that the thesis I presented for examination for the MPhil/PhD degree of the London School of Economics and Political Science is largely based on joint work with Bernhard von Stengel. The results in Section 3.3 are published in Casetti, Merschen and von Stengel (2010). Theorem 5.3 in Section 5.2 is joint work with Ron Peretz.

Contents

List of Figures

Acknowledgements

Foremost I would like to thank my PhD supervisor Bernhard von Stengel. I am deeply indebted to him for outstanding academic guidance. Bernhard has always been there for me with excellent supervision. Working together with him has taught me greatly and it has been a real pleasure to learn from and with him. I would also like to thank him for introducing me to the academic community and exposing me to all the conferences, especially to the Equilibrium Computation conferences in Dagstuhl, Germany. It is there where I had the chance to meet many of the authors whose results are cited in this thesis. I could not have wished for a better supervisor in any dimension.

I would also like to thank my co-author Marta Casetti. It has been a pleasure working together with her through out the years, filled with fruitful discussions and learning experiences. I thank Ron Peretz for the stimulating discussions that led to a new result. Also, I like to thank Rahul Savani for the valuable discussions and constructive advice he gave me from the beginning of my PhD.

I would like to thank all the members of the Mathematics Department, in particular Jackie Everid, Simon Jolly, and David Scott for always being there with support and help in any situation in life. Furthermore, I would like to thank Robert Simon for his continuous help and guidance over the last years. Also, I would like to thank Graham Brightwell, Jan van den Heuvel and Josef Skokan for constructive feedback and help.

I thank the London School of Economics and Political Science and its Mathematics Department for providing me with such an inspirational setting for research and personal development. My educational journey has been greatly shaped by this institution. I also thank the UK Engineering and Physical Sciences Research Council and the Heilbronn Institute at the University of Bristol for their generous financial support.

Finally I would like to thank my family for their unconditional support, love and encouragement. You have been truly inspirational.

Chapter 1

Introduction

In Section 1.1, we present the required material needed to understand this introductory chapter followed by the contribution of this thesis in Section 1.2. The structure of the thesis is described in Section 1.3. In Section 1.4, we elaborate on related work.

1.1 Background

This thesis concerns the problem 2-Nash of finding a Nash equilibrium of a bimatrix game, for the special class of so-called "hard-to-solve" bimatrix games. Bimatrix games are a common model to analyse the strategic interaction between two players. A solution to the games can be computed with the Lemke-Howson algorithm which determines the outcome by pivoting. It achieves this usually quickly, except for the class of hard-to-solve bimartix games, where it needs exponentially many steps in the size of the game.

A bimatrix game (A,B) is defined by two players where the $m \times n$ integer matrices A and B are the payoff matrices of the two players. Each player has a finite set of *pure* strategies, which are the rows and columns of the matrices for the players. To play the game, each player independently and simultaneously chooses a pure strategy, and their combined choice, a *strategy profile*, specifies the payoff for the two players given as an entry in their payoff matrix. A *mixed strategy* of a player is a randomization over his pure strategy set. The goal of each player is to maximize his own expected payoff. This maximization problem depends on the strategic action of the other player. A *Nash equilibrium* is a possibly mixed strategy profile such that no player has the incentive to deviate

unilaterally. Nash's famous theorem (1951) asserts that every bimatrix game has such an equilibrium in mixed strategies.

While economists are more interested in whether an equilibrium exists, the field of complexity theory, a part of theoretical computer science, asks the following question. Given a problem, such as 2-NASH, how quickly can a solution be found, thus, what is the complexity of solving the problem? The complexity of a problem is measured as a function of the size of the encoding of the input, for 2-NASH the number of bits needed to encode the payoff matrices. A problem is solved "efficiently" if the number of steps needed to arrive at a solution requires only "polynomially" many steps, making the problem "computationally tractable". Chen and Deng (2006) and Daskalakis, Goldberg and Papadimitriou (2006) showed that 2-NASH is PPAD-complete. PPAD is a complexity class which is comprised of problems that are known to have a solution by means of a "parity argument". However, at present it is not believed that a polynomial-time algorithm exists for 2-NASH.

A standard method to find a Nash equilibrium for a bimatrix game is due to Lemke and Howson (1965), referred to as the LH-algorithm. The LH-algorithm is a pivoting algorithm that starts at a completely labeled vertex of a polytope, which describes the best response conditions of a bimatrix game (see Shapley (1974) for the introduction of labels and von Stengel (2002) for the polytope description). An initial label is dropped and a new label is picked up. After that the old duplicate label is dropped. This process is repeated until the initially dropped label is found again, resulting in another completely labeled vertex, equivalent to another Nash equilibrium.

The class of what we call hard-to-solve bimatrix games has been described by Savani and von Stengel (2006) and is comprised of games that are not polynomially solvable by standard algorithms that compute Nash equilibria. The importance of this subclass of games is comparable to the exponential counterexamples for another famous pivoting algorithm, the simplex method by Dantzig (1963). This algorithm is usually efficient on typical instances. Klee and Minty (1972) describe a set of problems where the simplex algorithm takes exponentially many pivots to arrive at a solution. Similarly, the class of hard-to-solve games is the first class for which the LH-algorithm and support enumeration algorithms take exponentially many steps and expected exponential time, respectively; see Savani (2006) and Savani and von Stengel (2006).

1.2 The contribution of this thesis

Certain polytopes are useful to construct bimatrix games with interesting properties, for example games that have a larger number of equilibria (von Stengel (1999)), or games where it takes a long time to find an equilibrium with the LH-algorithm; see Savani and von Stengel (2006).

Equilibria of bimatrix games can be described by means of "best response polytopes"; see von Stengel (2002) for a survey. The problem 2-NASH is then equivalent to a combinatorial polytope problem in two respects. First, only the combinatorial structure of the polytope is important, i.e. given a vertex of a polytope, it only matters which facets a vertex lies on. Second, the facets of the polytope have labels, which correspond to unplayed or best response pure strategies (Shapley (1974)). In equilibrium, each pure strategy must be present as such a label. This is because a pure strategy that is not a best response must have probability zero in equilibrium. In other words, Nash equilibria are exactly the completely labeled vertices.

The correspondence between bimatrix games and labeled polytopes goes both ways. It is possible to define a bimatrix game starting from any labeled polytope given by inequalities, under the assumption that it has one completely labeled vertex. This special vertex corresponds to the so-called "artificial equilibrium". For details see Section 3.2 which provides details on polytopes and unit vector games; note that here the labeled polytope directly defines the payoff matrix B of player 2 while the columns of A are given by unit vectors which correspond to the labeling of the polytope. In particular, one can use the so-called dual cyclic polytopes, which have a known combinatorial structure in any dimension d.

The combinatorial structure of a dual cyclic polytope is completely described by its dimension d and the number n of its facets. We always assume that d is even, which simplifies the description. Any vertex is defined by the d facets it lies on, which are a subset of the set of all n facets. Any vertex of this polytope can be fully described combinatorially by bitstrings of length n such that exactly d bits are set to one ("1") and the rest are zero ("0") and such that they fulfill the Gale evenness condition (Gale (1963)). This condition requires that a *run* of consecutive one's has even length. Such a run may extend over the ends of the bitstring where the two ends of the run can be "glued"

together to fulfill the evenness condition. We denote the set of these strings by $G(d,n)$; see Section 3.1.

For constructing a game, one also needs labels for the n facets of the polytope. Each label receives a number from the set $\{1,\ldots,d\}$. There has to be a completely labeled vertex corresponding to the artificial equilibrium. For the dual cyclic polytope, we normally assume that this is the vertex encoded by the bitstring 1^d0^{n-d}, that is, d ones followed by $n-d$ zeros, which belongs to $G(d,n)$. The first d facets are therefore assumed to have labels $1,\ldots,d$ and only the remaining $n-d$ labels are chosen freely. We call a game defined by $G(d,n)$ and a string of labels as described a *Gale game*.

Games where the LH-algorithm requires exponentially many steps can be derived from the construction by Morris (1994). In this case, $n = 2d$, and the remaining d labels have a simple pattern (see Section 4.4). An important questions is to which extent the complexity of finding a Nash equilibrium of a bimatrix game is already captured by labeled cyclic polytopes? These games do give rise to exponentially long LH-paths. In addition, cyclic polytopes labeled with Morris's construction can be used to generate hard instances for support enumeration algorithms (Savani and von Stengel (2006), Section 4). However, a first main result of this thesis shows that for any Gale game a Nash equilibrium can be found in polynomial time; see Section 3.3. Interestingly, this result strongly suggests that the structure of Gale games is not "rich" enough to capture the full complexity of 2-NASH, unless P = PPAD.

A *perfect matching* of a graph is a set of edges such that each vertex is adjacent to exactly one of these edges. The result that a Nash equilibrium in Gale games is found in polynomial time follows from a reduction to PERFECT MATCHING, shown to be polynomially decidable by Edmonds (1965). From the label string l we build a graph with d vertices. Each vertex corresponds to exactly one label. Two vertices are connected by an edge if and only if their corresponding labels are neighbours in l. A perfect matching in the graph exactly captures completely labeled Gale strings. The artificial equilibrium is thus one perfect matching of the graph, given by the set of edges $\{(1,2),\ldots,(d-1,d)\}$. A second matching, corresponding to a "real" Nash equilibrium, is found in polynomial time; see Section 3.3 for details. Thus, finding a second perfect matching in certain Euler graphs (these are graphs where each vertex has an even number of incident edges) is

equivalent to computing a Nash equilibrium in Gale games. This is a new link between game and graph theory.

Already Morris (1994) and Savani and von Stengel (2006) translated the LH-algorithm to the setup of Gale strings, here called the LHG-algorithm (the "G" stands for Gale). We adapt the LHG-algorithm to find another perfect matching when considering an Euler graph and an initial perfect matching. We simply call it the pivoting algorithm; see Section 4.2. The Euler property follows from the label string l, which is an Euler tour of the corresponding graph.

The graphs derived from the Morris labels, here called a Morris graph G_d with d vertices, again give rise to long pivot paths. However, there is a polynomial-time algorithm to find another perfect matching for general Euler graphs. Note that the outcome of this algorithm does not have to be identical to the one from the pivoting algorithm. The pivoting algorithm relies on a pairing of the edges, initially given by the label string or Euler tour of the graph. This determines the next step. We show that for the class of Morris graphs there exists a pairing different to the one given by the original label string. Using this pairing the pivoting algorithm finds another matching in linear time in the number of edges; see Section 6.1.

The pivoting algorithm coincides with the "exchange algorithm" for Euler complexes or oiks (Edmonds (2007)), of which Euler graphs are special case; see Section 3.5. Also, Morris graphs are the only known example of Euler complexes where dropping any starting vertex in the exchange algorithm leads to exponentially long paths.

Motivated by Shapley's result (1974), which asserts that the endpoints of the path computed by the LH-algorithm have a different sign, i.e. the Nash equilibria, we introduce a *sign*, either positive or negative, for any completely labeled Gale string (see Section 3.4). The sign of the Gale string transfers to the sign of a perfect matching in a digraph; see Section 3.6. The pivoting algorithm keeps the convention that endpoints, i.e. the perfect matchings, have different signs. It is the only algorithm known to us which finds a different signed matching without exhaustively checking all perfect matchings.

In hope to finding a polynomial-time argument to compute a different signed matching we ask the question whether the Euler tour, or more generally the pairing of the edges, has to be considered as an assumption. LaPaugh and Papadimitriou (1994) show that

finding an even length path between two vertices of a directed graph is NP-complete. For general directed graphs, where the orientation is not necessarily consistent with the Euler tour, the decision problem which decides whether another signed perfect matching exists, called EXISTENCE OF SIGNED PERFECT MATCHING, is NP-complete. This is proven by a reduction from the even length path in digraph problem.

For two special classes of graphs we give polynomial-time arguments to find a signed matching. If the Euler graph is *bipartite* the pivoting algorithm computes a perfect matching of a different sign in linear time in the number of vertices of the graph (see Section 4.3). For *planar* graphs the argument does not follow from the pivoting algorithm. However, the planarity of a graph implies that the number of perfect matchings can be computed efficiently. Combined with the Euler property we can determine and successively delete edges that are part of both positive and negative signed perfect matchings. This allows us to construct a perfect matching from the deleted and remaining graph, resulting in a polynomial time algorithm, see Section 5.2. The planar result is joint work with Ron Peretz.

Counting the number of perfect matchings, even for a bipartite graph, is #P-complete; see Valiant (1979). Using the equivalences between perfect matchings and Nash equilibria in Gale games we prove that counting the number of Nash equilibria is #P-complete. This was first shown by Conitzer and Sandholm (2003) by a completely different reduction from a counting problem for Boolean formulas in conjunctive normal form.

1.3 The structure of this thesis

In this section we outline the structure of the thesis, and point out where some of our results have already been published.

Chapter 2 gives the needed background for the thesis. Section 2.1 provides bimatrix games and the notion of a Nash equilibrium. Section 2.2 describes theoretical concepts in computational complexity theory. We give the definition of a decision and function problem and explain how to classify them in the complexity classes P, NP, PPAD, and #P. The last section of the chapter points out the connection between completely labeled polytopes and Nash equilibria, and explains the Lemke-Howson algorithm.

In Chapter 3 we show the equivalence between Nash equilibria of Gale games, completely labeled facets of dual cyclic polytopes, Gale strings and perfect matchings. The last equivalence was published in Casetti, Merschen and von Stengel (2010) and shows that each of these problems is decidable in polynomial time. In Section 3.1 we introduce the decision problem COMPLETELY LABELED GALE STRING and ANOTHER COMPLETELY LABELED GALE STRING and explain the connection to labeled polytopes and Nash equilibria, published in Casetti, Merschen and von Stengel (2010). In Section 3.3 we give the polynomial-time reduction from COMPLETELY LABELED GALE STRING to PERFECT MATCHING and show that ANOTHER COMPLETELY LABELED GALE STRING is polynomial-time solvable as well. In Section 3.4 we define a sign for a Gale string and give a memoryless pivoting algorithm to compute ANOTHER COMPLETELY LABELED GALE STRING. We show that the number of Gale strings with a positive sign equals the number with a negative sign, a new result. In Section 3.5 we describe Euler complexes, or oiks, defined by Edmonds (2007). In Section 3.6 of the chapter we describe the notion of signed perfect matchings which relate to the Pfaffian of an adjacency matrix of a digraph. We also give an alternative, algebraic proof to show the parity of positive and negative signed matchings.

In Chapter 4 we focus on finding a perfect matching of different sign. In Section 4.1 we describe Edmonds's algorithm in detail and introduce the function problem ANOTHER SIGNED PERFECT MATCHING. We demonstrate some properties of the symmetric difference of two matchings of opposite sign. In Section 4.2 we adapt the LH-algorithm to the graph setting and generalise it such that any pairing of edges is sufficient for the algorithm's correctness. In Section 4.3 we show that the pivoting algorithm finds a perfect matching of different sign in linear time if the graph is bipartite. In Section 4.4 we show that the pivoting algorithm can take exponential time to find a solution when the input is a Morris graph.

In Chapter 5 we show that if we can efficiently count the number of perfect matchings in an Euler graph then this yields a polynomial-time argument for ANOTHER SIGNED PERFECT MATCHING. In Section 5.1 we show that counting the number of perfect matchings in general Euler graphs is #P-complete and discuss the relationship to Polya's scheme. In Section 5.2 we describe how to obtain a Pfaffian orientation for a planar graph. This orientation lets us count the number of perfect matchings in polynomial time. This

means that for planar Euler graphs the problem ANOTHER SIGNED PERFECT MATCHING is in FP. This result is joint work with Ron Peretz. In Section 5.3 we give a different proof to Conitzer and Sandholm (2003) showing that counting Nash equilibria is #*P*-complete.

In Chapter 6 we describe different approaches that could lead to a polynomial-time algorithm to solve ANOTHER SIGNED PERFECT MATCHING for general graphs. In Section 6.1 we prove that we need to consider the Euler tour, or more generally, some pairing of the edges, in order to hope to find a polynomial-time algorithm under the assumption that $P \neq NP$. Here, we also give a specific pairing of the edges such that the pivoting algorithm takes linear time for a Morris graph. In Section 6.2 we described the solution set of the pivoting algorithm and conjecture that under certain conditions a contraction argument leads to a polynomial time algorithm. In Section 6.3 we extend Edmonds's algorithm to incorporate the Euler tour.

In Chapter 7 we discuss recent complexity results for the LH-algorithm. A conclusion of our results is given as well as open problems. A list of symbols and references is included at the end of this thesis.

1.4 Related work

Zero-sum games are bimatrix games where the positive payoff to one player is the loss of the other, so these are games with directly opposed strategic interests of the players. For zero-sum games the set of Nash equilibria is convex and can be computed by a linear program (LP). These can be efficiently solved with interior point methods; see Ye (1997) for an excellent introduction. Instead of pivoting from vertex to vertex of a polytope, interior point algorithms take steps in the interior of the polytope towards a solution of the LP. For games with large payoff matrices, such as poker, interior point algorithms are not feasible due to memory requirements; see Gilpin, Peña, and T. Sandholm (2008) for an algorithm that is just as time efficient but does not need the additional memory.

The complexity of computing a Nash equilibrium for non zero-sum-games, even for two players, was shown to be PPAD complete (Chen and Deng (2006)). Asking more detailed questions about the solution of bimatrix games, such as what is the Nash equilibrium with the highest social welfare, or whether there exists a Nash equilibrium where,

say player 1, has an expected payoff of at least k, are NP-hard to compute (Conitzer and Sandholm (2003)).

Edmonds and Sanita (2010) investigate combinatorially defined "manifolds". These are collections of triples of points called triangles or "rooms", such that any "wall" of a room (obtained by omitting a point from the room) belongs to exactly one other room. They study *room partitions* which are partitions of the set of all points into rooms. They show that room partitions come in pairs by means of a pivoting algorithm. In addition, they show that this algorithm to find a second room partition can take exponential time.

The manifolds by Edmonds and Sanita (2010) can also be defined for rooms with any fixed number d of points. Based on this, they introduce an abstract equilibrium theory and show that if a game can be described by simplicial manifolds then equilibria, just as room partitions, come in pairs as endpoints of the pivoting or "exchange" algorithm. Scarf's (1967) setup to compute the core of an n person game falls under Edmonds and Sanita's characterisation. This establishes an interesting link between cooperative and non-cooperative game theory. The solution of both fields can be found by pivoting algorithms, alternatively by pivoting in two simplicial manifolds.

In graph theory, finding paths and cycles with certain properties in a graph has been studied for a long time. Some of these paths and cycles can be found in polynomial time. However, certain questions, for example finding a longest simple path, or a path of certain length, or an even-length path connecting two vertices in a directed graph, are NP-complete. The last problem is related to the question: Given two adjacent vertices, does there exist an even-length directed cycle between the two vertices? The local problem is NP-complete, but interestingly the global problem of deciding whether a directed graph has such a cycle, without specifying an edge of such a cycle, is tractable; see McCuaig, Robertson, Seymour, and Thomas (1997).

Chapter 2

Bimatrix Games and the Complexity of Computing a Nash Equilibrium

In Section 2.1 an introduction to bimatrix games and Nash equilibria is given. We define the necessary complexity classes and computational theory in Section 2.2. Here we also discuss the complexity results of η-NASH. In Section 2.3 we explain the connection between Nash equilibria in bimatrix games and labeled polytopes. We describe the Lemke-Howson algorithm. This algorithm finds one Nash equilibrium by pivoting in the labeled polytope derived from the bimatrix game.

2.1 Bimatrix games and Nash equilibria

A bimatrix game (A,B) is given by $m \times n$ payoff matrices for player 1 and player 2. A *pure strategy* for player 1 is a row, a pure strategy for player 2 is a column. A *mixed strategy*, $x \in \mathbb{R}^m$ for player 1 or $y \in \mathbb{R}^n$ player 2, is a probability distribution over his pure strategies. For a matrix A its transpose is $A^\top$ and a_{ij} refers to an entry in row i and column j. All vectors are written in column form, so a row vector, typically for a mixed strategy x of player 1, is written as its transpose $x^\top$. A mixed strategy x of player 1 is a best response against any mixed strategy y of player 2 if $x^\top Ay \geq x'^\top Ay$ holds for all mixed strategies x', where inequalities hold componentwise. A best response (or best reply) for player 2 is defined analogously. The goal of each player is to maximise their own payoff. A Nash equilibrium is a *strategy profile* (x,y) such that x is a best response to y and vice versa. In

other words, a Nash equilibrium is a stable outcome of the game where no player benefits from deviating unilaterally. Nash's famous theorem (Nash (1951)) asserts that every game with finitely many players and pure strategies possesses at least one equilibrium in mixed strategies.

The problem of computing a Nash equilibrium in a bimatrix game is referred to as 2-NASH, while for η-player games it is given by η-NASH. The *support* of a strategy is the set of rows or columns played with positive probability. A pure strategy thus has support size one. A bimatrix game is called *nondegenerate* if no mixed strategy of support size k has more than k best pure responses. Throughout this thesis we assume that games are nondegenerate. Any game can be made nondegenerate by a suitable "lexicographic" perturbation of A and B; for example see Savani (2006).

An example of a bimatrix game is the well known "battle of the sexes" game defined by matrices $A = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$ and $B = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$. The game has three Nash equilibria. Two pure strategy equilibria, with support 1, at $(x,y) = \left((1,0)^\top, (1,0)^\top\right)$ and $\left((0,1)^\top, (0,1)^\top\right)$ with payoffs pairs $(2,1)$ and $(1,2)$, respectively. The third Nash equilibrium is fully mixed with probability vectors $\left((2/3,1/3)^\top, (1/3,2/3)^\top\right)$ and an expected payoff of $2/3$ for both players.

2.2 Complexity classes and the complexity of finding Nash equilibria

A *decision problem* is described by an *input* and a *question*, where the answer to the question is either "YES" or "NO". A *function problem* is defined similarly but instead of a questions its *output* can encode more information than simply "YES" or "NO". For example, the output could be a Hamiltonian tour of a graph or "NO" in case the graph does not admit one.

Let x be an instance of a problem and let $|x|$ be the length of the encoding, by which we mean the number of bits needed to encode x. A decision problem can be identified with the set of inputs x where the answer is "YES". A decision problem P_1 *reduces* to another problem P_2 in polynomial time, $P_1 \leq_p P_2$, if there exists a polynomial-time reduction from

P_1 to P_2; see Garey and Johnson (1979) or Papadimitriou (1994a) for an introduction to the topic. Formally, P_1 is *polynomial-time reducible* to P_2, if there exists a function $f : \{0,1\}^* \to \{0,1\}^*$ and a Turing machine $\mathcal{M}$ such that for all $x \in \{0,1\}^*$:

1. $x \in P_1 \Leftrightarrow f(x) \in P_2$,
2. $\mathcal{M}$ computes $f(x)$,
3. $\mathcal{M}$ stops after $p(|x|)$ steps, where p is some polynomial.

This means that P_1 is reduced to P_2 by taking the input x and then computing $f(x)$ in polynomial time, and then deciding whether $f(x)$ is a YES instance of P_2, which answers whether x is a YES instance of P_1.

The complexity class P, the class of polynomially decidable problems, is defined as follows. Intuitively, a decision problem is in P, if there exists an algorithm that requires polynomially many steps in the input size of the problem to arrive at the answer of the question. Formally, a decision problem P_1 is in P if there exists a Turing machine $\mathcal{M}$ that outputs either "YES" or "NO" in polynomial time, that is, if for all $x \in \{0,1\}^*$, $\mathcal{M}$ stops after at most $p(|x|)$ steps, where p is some polynomial. We often use the word *efficient* or *tractable* if an algorithm takes at most polynomially many steps. Analogously, the class FP contains the set of function problems that can be solved in polynomial time. Formally, FP is the class of binary relations $R(x,y)$ for which a polynomial-time algorithm exists such that this algorithm, for a given x, computes a y for which $R(x,y)$ holds (Megiddo and Papadimitriou (1991)).

A decision problem is in the class NP, nondeterministic polynomial-time problems, if one can "guess" a solution, a so called *certificate* y, of the instance, and "check" whether the certificate satisfies the question of the problem in polynomial time. For example, one guesses a tour of a graph and checks whether it is a Hamiltonian tour and then outputs either "YES" or "NO". We formally define the class NP using deterministic Turing machines. A decision problem P_1 is in NP if there exists a Turing machine $\mathcal{M}$ and polynomials p_1, p_2 such that

1. for all $x \in P_1$ there exists a certificate $y \in \{0,1\}^*$ which satisfies $|y| \leq p_1(|x|)$ and $\mathcal{M}$ accepts the combined input xy, stopping after at most $p_2(|x| + |y|)$ steps,
2. for all $x \notin P_1$ there does not exist a $y \in \{0,1\}^*$ such that $\mathcal{M}$ accepts the combined input xy.

Equivalently, a problem is in NP if it can be decided by a nondeterministic Turing machine in polynomial time, see Papadimitriou (1994a). A problem P_2 is NP-complete if it is in NP and for all P_1 in NP, $P_1 \leq_p P_2$. A problem P is in the class co-NP if its complement is in NP.

The class FNP allows for the function form of decision problems that are in NP. That is, it asks to output a specific solution to a YES instance or "NO", if the instance has no such solution. Formally, FNP is the class of binary relations $R(x,y)$ for which a polynomial-time algorithm exists such that this algorithm, given x and y, where $|y| \leq p(|x|)$, decides whether $R(x,y)$ holds (Megiddo and Papadimitriou (1991)). If an instance of a function problem is known to always have a positive answer then the problem is called *total*. The class TFNP (total function nondeterministic polynomial) consists of exactly these problems. For problems in TFNP, we additionally require that for every instance x, there exists a y such that $R(x,y)$ holds. Interestingly, unless NP = co-NP, the class TFNP does not have any complete function problems (Megiddo and Papadimitriou (1991)). The intuition is as follows. Suppose there exists a FNP-complete problem P_1 in TFNP. Then take a problem in FNP, for example the function form of the Boolean satisfiability problem, FSAT. By a polynomial-time reduction from FSAT to P_1 any unsatisfiable Boolean formula has a certificate of unsatisfiability; guaranteed because P_1 is total. In other words we can recognise both satisfiable and unsatisfiable Boolean formulae via the reduction to P_1. This shows NP = co-NP. Papadimitriou (1994b) introduced subclasses of TFNP characterised by a "method of proof" to show totality. We describe two of these classes in detail below.

The complexity class PPA, PPAD, and PPADS, subclasses of TFNP (Papadimitriou (1994b)), comprise function problems that are known to have a solution via an undirected and directed "polynomial parity argument". It is well known that if a graph has a node of odd degree, then by parity argument it must have another. This is the argument to prove totality. In this context PPA is defined as the class of problems, given a graph and a node with odd degree, find another such node. Also, if a directed graph has an *unbalanced* node, that is a vertex with different in-degree and out-degree, then by parity argument for directed graphs it must have another. PPAD is defined as the class of problems, given a directed graph and an unbalanced node, find another unbalanced node. The class PPADS, where the extra "S" stands for "signed", restricts the solution space to only sinks, see

Yannakakis (2009) for a recent survey. The term "polynomial" in the classes PPA and PPAD refers to the fact that the neighbors of a node in the graph are implicitly given as the output of some polynomial-time algorithm; the resulting graph itself may be exponential.

Formally, PPAD consists of problems that reduce to the problem END OF THE LINE, given by two polynomial-sized Boolean circuits σ and π with k input and k output bits; see Daskalakis, Goldberg and Papadimitriou (2009). This pair σ, π defines an implicit digraph with k-bitstrings as vertices and arcs (u,v) whenever $\sigma(u) = v$ and $\pi(v) = u$. If $\sigma(\pi(u)) \neq u$ then u is a source and if $\pi(\sigma(v)) \neq v$ then v is a sink of this digraph. It is assumed that 0^k is a source. The sought output is any sink, or source other than 0^k. See Figure 2.1 for an example.

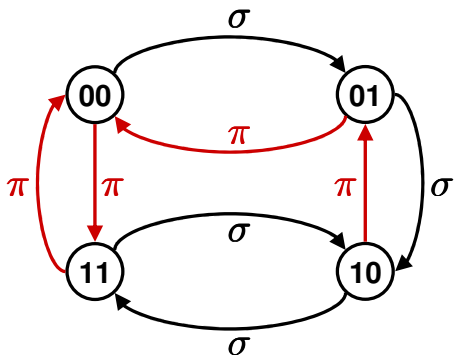


Figure 2.1 The polynomial-sized Boolean circuits σ and π for $k = 2$. The successor circuit, σ, is given by the black edges and the predecessor circuit, π, by the red edges. The implicit digraph is given by the vertices 00, 01 and 10 connected with a directed edge in that order, where 00 and 10 are the source and sink, respectively.

The output of END OF THE LINE exists because the digraph is a collection of directed paths and cycles, with at least one path which starts at 0^k. This output can be any sink or source and does not have to be the sink at the end of the directed path starting at 0^k. This particular sink is computed in the problem OTHER END OF THIS LINE, which is PSPACE-complete (Papadimitriou (1994b)). We discuss this in more detail in Section 7.1. A problem is called PPAD-complete if the problem is a member of the class PPAD and if END OF THE LINE, or any other problem in PPAD, reduces to it in polynomial time. The main difference to the class of NP is that the existence of a solution is guaranteed in PPAD; however, finding a solution might take exponential time. It is believed that PPAD-complete problems are not solvable in polynomial time.

Other interesting computational problems such as the fixed point problems BROUWER, KAKUTANI (Papadimitriou (1994b)) and SPERNER (Chen and Deng (2006)) as well as other equilibrium concepts, e.g. EXCHANGE ECONOMY (Papadimitriou (1994b)) have been shown to be PPAD-complete. Also see Kintali, Poplawski, Rajaraman, Sundaram and Teng (2009) for an overview of PPAD-complete fractional stability problems.

While the existence of a Nash equilibrium has been known for 60 years and the classes PPA and PPAD at latest since Papadimitriou (1994b), the complexity of 2-NASH has only recently been solved by Chen and Deng (2006). Chen and Deng prove that 2-NASH is PPAD-complete and improve the construction of Daskalakis and Papadimitriou (2005) and Daskalakis, Goldberg and Papadimitriou (2005) (see (2009) for an expository article) who showed that η-NASH for $\eta \geq 3$ is PPAD-complete. See Goldberg (2011) for a detailed survey of PPAD and computing Nash equilibria.

Their proof relies on a string of reductions between BROUWER, SPERNER and NASH. However, these results only imply that finding an approximate Nash equilibrium is PPAD-complete, where the accuracy of approximation is inversely proportional to the encoding of the game. This is especially of importance when considering games with more than two players, because the equilibria can have irrational probability vectors even if all the entries in the matrices consist of rational numbers (Nash (1951)). The Nash equilibria of bimatrix games have rational solutions and can thus be exactly determined, by solving a system of linear equations for the support of the equilibrium.

To compute a Nash equilibrium in practice, an often used and, for most cases, well-performing algorithm is the famous Lemke-Howson algorithm (see Lemke and Howson (1964)). Although over 40 years old, it is still one of the standard algorithms for computing Nash equilibria for bimatrix games. It is implemented, for example, in the game theory software Gambit (McKelvey, McLennan and Turocy (2010)). We give an overview of the Lemke-Howson algorithm in the next section and discuss recently shown complexity results of this algorithm in Section 7.1.

2.3 Best reply polytopes, Nash equilibria, and the Lemke-Howson algorithm

We want to describe Nash equilibria geometrically using polytopes. For this we need the following notation taken from von Stengel (1999) and (2002). For an introduction to polytopes see Ziegler (1995). We treat vectors u, v in $\mathbb{R}^d$ as column vectors, so $u^\top v$ is their scalar product. By $\mathbf{0}$ we denote a vector of all 0's, of suitable dimension, by $\mathbf{1}$ a vector of all 1's. A unit vector, which has a 1 in its i-th component and 0 otherwise, is denoted by e_i. Inequalities like $u \geq \mathbf{0}$ hold for all components.

An *affine combination* of points $z_1, \ldots, z_k$ in some Euclidean space is of the form $\sum_{i=1}^k z_i \lambda_i$ where $\lambda_1, \ldots, \lambda_k$ are reals with $\sum_{i=1}^k \lambda_i = 1$. It is called a *convex combination* if $\lambda_i \geq 0$ for all i. A set of points is *convex* if it is closed under forming convex combinations. Given points are *affinely independent* if none of these points is an affine combination of the others. A convex set has *dimension* d if and only if it has $d+1$, but no more, affinely independent points.

A *polyhedron* P in $\mathbb{R}^d$ is a set $\{z \in \mathbb{R}^d \mid Cz \leq q\}$ for some matrix C and vector q. It is called *full-dimensional* if it has dimension d. It is called a *polytope* if it is bounded. A *face* of P is a set $\{z \in P \mid c^\top z = q_0\}$ for some $c \in \mathbb{R}^d$, $q_0 \in \mathbb{R}$ so that the inequality $c^\top z \leq q_0$ holds for all z in P. A *vertex* of P is the unique element of a 0-dimensional face of P. An *edge* of P is a one-dimensional face of P. A *facet* of a d-dimensional polyhedron P is a face of dimension $d-1$. It can be shown that any nonempty face F of P can be obtained by turning some of the inequalities defining P into equalities, which are then called *binding* inequalities. That is, $F = \{z \in P \mid c_i z = q_i,\ i \in I\}$, where $c_i z \leq q_i$ for $i \in I$ are some of the rows in $Cz \leq q$. A facet is characterised by a single binding inequality which is *irredundant*, that is, the inequality cannot be omitted without changing the polyhedron (Ziegler (1995), Chapter 2). A d-dimensional polyhedron P is called *simple* if no point belongs to more than d facets of P, which is true if there are no special dependencies between the facet-defining inequalities.

Above, a polytope was defined by the intersection of finitely many closed *halfspaces* in some $\mathbb{R}^d$, where the intersection is required to be bounded. It is often easier to alternatively define a polytope as the convex hull of a finite set of points in some $\mathbb{R}^d$. Then a

d-dimensional *simplicial polytope* P is the convex hull of a set of at least $d+1$ points v in $\mathbb{R}^d$ in general position, that is, no $d+1$ of them are on a common hyperplane.

The *polar* or *dual* of a polytope P that is the convex hull of the d-vectors $c_1, \ldots, c_N$ is given by

$$P^\Delta = \{x \in \mathbb{R}^d \mid c_i^\top x \leq 1,\ 1 \leq i \leq N\}, \tag{2.1}$$

provided P (and then also P^Δ) has $\mathbf{0}$ in its interior, which can always be achieved by translating P. Any face of P^Δ of dimension $d-k$ is defined by k binding inequalities in (2.1), and corresponds to a face of dimension $k-1$ of P, given by the convex hull of the corresponding k vertices of P. When drawing a polytope we usually use a *Schlegel diagram* for higher dimension polytopes. It is a projection of a polytope from dimension d to $d-1$ through a point outside of the polytope onto a single facet (Ziegler (1995), Chapter 5).

Nash equilibria of bimatrix games and polytopes are related as follows (for an in depth discussion see Savani (2006) and von Stengel (2007)). 2-NASH has as its input two integer $m \times n$ matrices A and B that define the bimatrix game. The desired output is one Nash equilibrium. Computing a Nash equilibrium of a bimatrix game is equivalent to the problem of finding a completely labeled facet of a labeled polytope, as described below.

First, we explain how the best response polytopes, one for each player, are constructed. Add constants to the payoffs such that all entries of A and $B^\top$ are nonnegative and have no 0 columns.[1] The best-reply polytopes P and Q, for player 1 and player 2, respectively, are defined by

$$\begin{aligned} P &= \{x \in R^m \mid \ \ x \geq \mathbf{0},\ B^\top x \leq \mathbf{1}\} \\ Q &= \{y \in R^n \mid \ \ y \geq \mathbf{0}, \quad Ay \leq \mathbf{1}\}, \end{aligned} \tag{2.2}$$

where each inequality receives a label from M or N, the disjoint pure strategy sets $M = \{1, \ldots, m\}$ and $N = \{m+1, \ldots, m+n\}$. The labels are allocated to the inequalities as follows (we explain this for Q). A point of Q has label $k \in M \cup N$ if the kth inequality in the definition of Q is binding. If this is the ith matrix inequality $\sum_{j \in N} a_{ij} y_j = 1$ then $k = i \in M$. This occurs when the pure strategy i is a best response to the mixed strategy y. On the other hand, the kth inequality receives a label from $k = j \in N$ if the binding inequality is $y_j = 0$. This case corresponds to a non-played pure strategy j of player 2.

[1] This can be done such that the equilibria of the game do not change.

In the definition of P and Q in (2.2), x and y do not correspond to probability vectors. They have been normalised such that the expected payoffs of the players are now at most 1, given by the matrix inequalities in (2.2). The nonnegativity conditions imposed on A and B imply that P and Q are polytopes. The polytopes P and Q in (2.2) are defined by a finite number of half-spaces, written as a set of linear inequalities; or in matrix form $B^\top x \leq \mathbf{1}$ for P.

Any vertex pair (x,y) in $P \times Q$ has the labels of the binding (or tight) inequalities. If all labels of $M \cup N$ are present then (x,y) is called *completely labeled*. By normalising x and y to probability vectors a completely labeled vertex (x,y) is a Nash equilibrium of the game. The completely labeled pair of vertices $(\mathbf{0},\mathbf{0})$, is not a Nash equilibrium and is referred to as the *artificial equilibrium*.

We assume the game is nondegenerate, which means that no point in P has more than m labels and no point in Q has more than n labels. So P and Q are *simple* polytopes. Additionally, we require that no inequality that defines P or Q is redundant. The latter requirement is needed because even if the polytopes are simple, the game can be degenerate. If an inequality which is sometimes binding, but otherwise can be omitted for, say, Q, is written down twice then this means that A has two identical rows. This makes the game degenerate. This case occurs when a player has a *weakly dominant* strategy. We thus require that the polytopes are nondegenerate from a geometric viewpoint. That is, if the inequalities defining them are "generic", then these polytopes are simple. Clearly, in a nondegenerate game a completely labeled (x,y) has each label exactly once. However, we do not need an algorithm to decide if the game is nondegenerate. Degenerate games can be resolved by symbolic lexicographic perturbation; a well known technique from linear programming (see von Stengel (2007)).

By the description of Nash equilibria and best reply polytopes the problem of 2-NASH looks like a "labeled polytope" problem. The special attribute is that the polytope is a product polytope, $P \times Q$, in dimension $m+n$ and defined by $2(m+n)$ inequalities, where P and Q are defined above in terms of matrices A and B.

Consider a simple d-dimensional polytope S with $d+k$ facets where each facet has a label from $\{1,\ldots,d\}$, and where the first d facets have labels $1,2,...,d$, respectively, which define a special, completely labeled, vertex v_0. The function problem ANOTHER COMPLETELY LABELED VERTEX is defined by an input S and the special vertex v_0; as

described above. Its output is a completely labeled vertex other than v_0. Clearly, 2-NASH is a special case of this problem with $S = P \times Q$ of dimension $d = k = m+n$, where P and Q are defined in (2.2) and $v_0 = (\mathbf{0}, \mathbf{0})$.

Consider the dual polytope S^Δ of S. In our case, the dual of the simple d-dimensional polytope S, is a d-dimensional simplicial polytope with $d+k$ vertices; see Ziegler (1995). The labeling of a facet in S is then transformed to its corresponding vertex in S^Δ. In particular, the facets defining the vertex v_0 are mapped to vertices which, as a convention, are associated with the facet F_0. In other words, the artificial equilibrium v_0 in S is represented by F_0 in S^Δ. The problem ANOTHER COMPLETELY LABELED VERTEX is thus the dual to the problem ANOTHER COMPLETELY LABELED FACET.

The algorithm due to Lemke and Howson (1964), in short the LH-algorithm, finds one Nash equilibrium of a bimatrix game. It can be translated to labeled simplicial polytopes, which we derived for bimatrix games, and in general solves the function problem ANOTHER COMPLETELY LABELED FACET.[2] Start with a completely labeled facet; such as F_0 above. Select one label i and *drop* it to move from F_0 to the unique adjacent facet that shares all vertices with F_0 except the one with label i. Label i is called the *missing* label. In other words by dropping label i we *flip* a facet to its unique neighbouring facet. The newly obtained facet, say F_1, has a new vertex with a label j; we say that label j was *picked up*. If $j = i$, then F_1 is completely labeled and the LH-algorithm terminates with a different completely labeled facet than F_0. Otherwise the "new" label j is now the *duplicate* label of F_1. Take the vertex v of F_1 that had label j so far, i.e. the "old" vertex associated with label j, and move to the unique adjacent facet F_2 that has all vertices of F_1 except v, and continue as before. This defines a unique "path" of facets that must eventually terminate at a completely labeled facet different from F_0; see Figure 2.2 for an illustration.

Due to the relationship between Nash equilibria of bimatrix games and fully labeled facets of a simplicial polytope the LH-algorithm gives a constructive, algorithmic proof for the existence of a Nash equilibrium for bimatrix games. The LH-algorithm is central to our thesis and will be used as a machinery for proving results. The path that connects two Nash equilibria by pivots is referred to as a *Lemke-Howson path* or *pivot path*. We adapt

[2] We briefly described the LH-algorithm in Section 1.1. In particular, LH-algorithm in Section 1.1 starts with v_0 and finds another completely labeled vertex in $P \times Q$. Both translations are equivalent by the duality of the polytopes and their labeling.

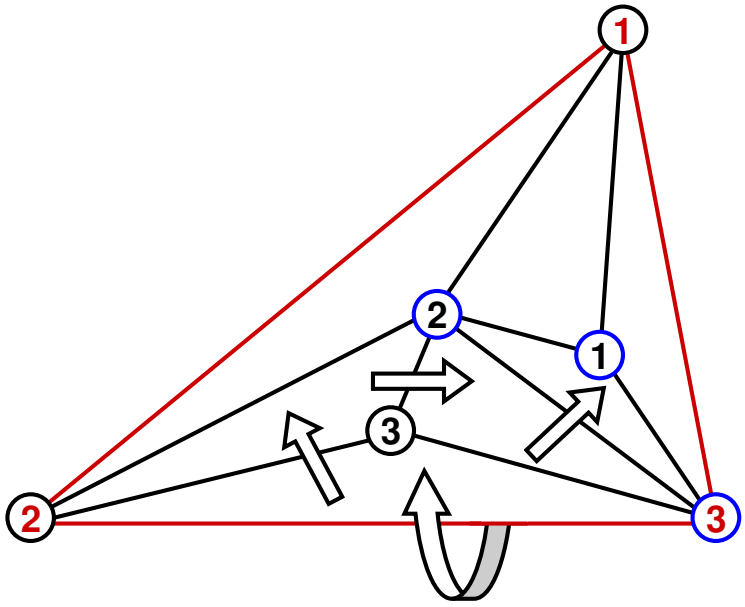


Figure 2.2 Let F_0 be the completely labeled facet given in red edges and red labels $1,2,3$; the starting facet of the LH-algorithm. Think of F_0 as the outer facet of the 3-dimensional polytope or as the bottom of the corresponding Schlegel diagram, depicted in the figure. Drop label 1 to flip the facet F_0 to the next facet given by $2,3,3$; depicted by the curved arrow. Continuing the process, leads to the other completely labeled facet given by the blue vertices.

this algorithm to find equilibria in two different settings or translations of cyclic polytopes that we describe in the next chapter. The terms "pivot path", "dropping", "picking", and "duplicate" will be overloaded but still capture similar principles in each context.

Chapter 3

Bimatrix Games, Gale Strings, and Perfect Matchings

In this chapter, we show the relationship between several areas of mathematics: Game theory, geometry, combinatorics and graph theory. In game theory, a Nash equilibrium of bimatrix games can be described geometrically by a completely labeled facet of labeled simplicial polytopes (see Section 2.3). Polytopes are objects studied in geometry. Special cyclic polytopes can be described combinatorially with Gale strings (Gale (1963)). We interpret Gale strings as graphs; as published in Casetti, Merschen and von Stengel (2010). This result implies that a perfect matching in graphs derived from a Gale string corresponds to a completely labeled Gale string and thus to a Nash equilibrium in Gale games.

In Section 3.1 we give an introduction to Gale strings and define the decision and function problem COMPLETELY LABELED GALE STRING and ANOTHER COMPLETELY LABELED GALE STRING, respectively. We adapt the LH-algorithm to the setting of Gale strings, here called the LHG-algorithm. In Section 3.2 we describe how Savani and von Stengel (2006) construct hard-to-solve bimatrix games. These are special games for which the LH-algorithm takes exponentially many steps. A special bimatrix game, (U,B), is called a *unit vector game* if the payoff matrix of player 1 is restricted to unit vectors. Balthasar and von Stengel (2010) establish that for the class of unit vector games (U,B) where B is obtained from the last n of $d+n$ vertices of a d-dimensional cyclic polytope, a Nash equilibrium is equivalent to a completely labeled Gale string.

The problem of finding a Nash equilibrium for Gale games is polynomial-time equivalent to computing the function problem ANOTHER COMPLETELY LABELED GALE STRING (Casetti, Merschen and von Stengel (2010)). We show that the labeled string defines an Euler graph and that completely labeled Gale strings are in a one-to-one relationship with perfect matchings in the corresponding graph; see Section 3.3. Computing a perfect matching was shown to be polynomial in the size of the input graph by Edmonds's (1965) famous Paths, trees, and flowers paper. Edmonds's polynomial-time result and the polynomial-time reductions between Gale games, Gale strings and Euler graphs implies that ANOTHER COMPLETELY LABELED GALE STRING and thus 2-NASH for Gale games is polynomial time solvable, a new result.

We extend Shapley's (1974) notion of orientation for Nash equilibria to define a sign for Gale strings, which corresponds to the sign of a perfect matching; see Sections 3.4 and 3.6, respectively. Shapley furthermore showed that Nash equilibria come in even numbers when including the artificial equilibrium and that they have opposite sign when they are at opposite ends of a Lemke-Howson path. We extend the LHG-algorithm to incorporate the sign of a Gale string and show that the positive and negative signed completely labeled Gale string come in pairs. This implies that perfect matchings in Euler graphs exhibit the same property; see Section 3.6.

In Section 3.5 we describe Euler complexes or oiks (Edmonds (2007)). Euler graphs are a special example of an oik in dimension 2. Edmonds (2007) showed that the number of room partitions, related to perfect matchings in low dimension oiks, is even. The orientation for oiks in dimension 2 follows logically. However, defining an orientation for oiks of higher dimension proves to be more difficult.

3.1 Gale strings

In Section 2.3 we explained that Nash equilibria of bimatrix games can be described by means of best response polytopes. Special games have the property that the dual of the product of these best response polytopes is cyclic (see Section 3.2 for the definition of a cyclic polytope). These polytopes can be combinatorially described by Gale strings which we describe next. This section is partially based on Casetti, Merschen and von Stengel (2010).

Let $[k] = \{1, \ldots, k\}$ for any positive integer k. For a set S, we consider a function $s : [k] \to S$ as a string, $s(1)s(2)\cdots s(k)$ of k elements of S. For $s : [k] \to S$ and a subset J of $[k]$, let $s(J)$ be the set $\{s(i) \mid i \in J\}$. If $S = \{0,1\}$, we call s a string of *bits*. A bitstring $s : [k] \to \{0,1\}$ can be considered as an indicator function of a subset of $[k]$ that we denote by $1(s)$, that is, $1(s) = s^{-1}(1) = \{j \in [k] \mid s(j) = 1\}$.

Let $G(d,n)$ be the set of all strings s of n bits so that exactly d bits in s are 1 and so that s fulfills the *Gale evenness condition*, that is, whenever 01^k0 is a substring of s, then k is even, due to Gale (1963). An element of $G(d,n)$ is also called a *Gale string* of dimension d and length n. For example, $G(4,6)$ consists of the nine strings 111100, 111001, 110110, 110011, 101101, 100111, 011110, 011011, 001111.

For a bitstring s, a maximal substring of s of consecutive 1's is called a *run*. A Gale string may only have interior runs, bounded on both sides by a 0, of even length but may start and thus end with an odd-length run. If d is even, then any s in $G(d,n)$ that starts with an odd run also ends with an odd run, and these two odd runs may be "glued together" to form an even run. This shows that the set of Gale strings of even dimension is invariant under a cyclic shift of the strings. Throughout this thesis we assume that d is even.

Given a length n and a parameter d, a *labeling* is a function $l : [n] \to [d]$. Given such a labeling l, a bitstring s in $G(d,n)$ is called *completely labeled* if $l(1(s)) = [d]$, that is, if every label in $[d]$ appears as $l(i)$ for at least one bit $s(i)$ so that $s(i) = 1$. Clearly, if s is completely labeled, then s has at least d bits that are 1, and if exactly d bits in s are 1, then every label in $[d]$ occurs exactly once. A bitstring is *almost completely labeled* or simply *almost labeled* if it satisfies the Gale evenness condition and exactly one label is duplicate and thus one label is missing.

For example, for the string of labels $l = 1123143$, with $d = 4$, the completely labeled Gale strings are 0110011 and 0011110. Note that the string 0111010 is completely labeled but does not fulfill the Gale evenness condition as both runs are of odd length. For $l =$ 123432 the completely labeled Gale strings are 111100, 110110, 100111, and 101101. An almost labeled Gale string for $l = 123432$ is 011110 with missing label $l(1) = 1$ completely labeled Gale strings. For $l = 121314$, there are no completely labeled Gale strings.

The decision problem that answers whether a labeled string has a completely labeled Gale string is defined as follows (Casetti, Merschen and von Stengel (2010)):

COMPLETELY LABELED GALE STRING

Input: A labeling $l : [n] \to [d]$, where d is even and $d < n$.

Question: Does there exists a Gale string s in $G(d,n)$ that is *completely labeled*?

We would like to adapt the LH-algorithm to the setting of Gale strings. We need to define the following concepts. For a Gale string s and a position $i \in [n]$ which satisfies $s(i) = 1$, a *pivoting step* is to first *drop* the label $l(i)$ by setting it to 0, i.e. $s(i) = 0$. By the Gale evenness condition, there is, either to the left or the right of $s(i)$, an odd run of 1's. Choose the side of $s(i)$ adjacent to the odd run and invert the first 0 to the left or right of the run, hence *picking* up a new label, which completes the pivoting step. In other words the 1 from $s(i) = 1$ that is set to 0, "jumps" over its adjacent odd-length run until its end is reached and then the first 0 adjacent to the run of 1's, say at position j, is inverted by setting $s(j) = 1$. The Gale evenness condition is thus satisfied after a pivot. However, s is not necessarily completely labeled anymore. A pivot for an almost labeled Gale string is defined similarly by dropping a duplicate label.

step	**1**	**2**	**3**	**4**	**2**	**4**
1	$\underline{1}$	1	1	1	.	.
2	.	$\underline{1}$	1	1	$\bar{1}$	.
3	.	.	1	$\underline{1}$	1	$\bar{1}$
4	.	$\bar{1}$	1	.	$\underline{1}$	1
5	$\bar{1}$	1	1	.	.	1

Figure 3.1 The pivot path for 123424 when $l(1) = 1$ is initially dropped. In the first step label 1 is dropped and label 2 is picked up, making it the duplicate label.

Figure 3.1 gives an example of pivots as part of the pivot path. A "$1 = s(i)$" is underlined at step k when the associated label $l(i)$ is dropped next. A "$1 = s(j)$" is "overlined" if the label $l(j)$ was just picked up, for example by having previously dropped $l(i)$. Note that a pivot step is reversible. Suppose we picked up label $l(j)$ by dropping a label $l(i)$, then $s(j) = 1$ must be at the end of a run and $s(i)$ is equal to 0 and adjacent to the opposite side of the run where $l(j)$ was just picked up. By dropping $l(j)$ we must pick $l(i)$, making the pivot reversible. The reversibility of a pivot is also captured as part of the w-skew graph, defined in Theorem 3.2.

Next we give the Lemke-Howson algorithm for Gale strings algorithm (LHG-algorithm) (see Algorithm 1) and then show its correctness in Lemma 3.1.

Algorithm 1: LHG-algorithm

input : A labeling $l : [n] \to [d]$, where d is even and $d < n$, and a completely labeled Gale string s in $G(d,n)$.

output: A completely labeled Gale string s' in $G(d,n)$ where $s' \neq s$.

```
pivot an arbitrary label of s;
while (s is not a completely labeled Gale string) do
    pivot the label that is a duplicate of the one picked up in the previous pivot;
return s' = s
```

Lemma 3.1 *The LHG-algorithm correctly computes another completely labeled Gale string.*

Proof: Clearly, there are only a finite number of possible bitstrings for each label string. Then the algorithm must terminate by finding another Gale string in a finite number of steps if cycling is not possible. This holds because of the following observation. Suppose the algorithm returns to a string s' other than the initial Gale string. Then at this bit assignment of s', because each pivot is reversible, we would have to be able to pick up two labels. This, however, is ruled out by the Gale evenness condition as only one of the adjacent runs of the dropped label is odd. Returning to the initial position is only possible by reversing the initial pivot which is not allowed. □

The only choice in the LHG-algorithm is at the beginning, where an arbitrary label w is dropped. Depending on the initial choice the output strings may vary. Figure 3.1 gives the pivot path for 123424, where 1 is initially dropped.

The set $G(d,n)$ of Gale strings has a combinatorial structure that allows the use of a "parity argument", to show the following known property; it holds for odd d as well but we assume throughout that d is even.

Theorem 3.2 *For any labeling $l : [n] \to [d]$, where d is even and $d < n$, the number of completely labeled Gale strings in $G(d,n)$ is even.*

Proof: Fix a $w \in [d]$ and consider the *w-skew graph* consisting of all completely labeled Gale strings and almost labeled Gale string with missing label w as the vertex set. Two vertices are connected by an edge if they can be reached by a pivot, either by initially dropping w from a completely labeled Gale string, or by dropping one of the duplicate labels. The degree-1 nodes of the graph are the completely labeled Gale strings and the degree-2 nodes are the ones with the missing label. By the parity argument two completely labeled Gale strings are connected by a path of pivots, showing that they come in pairs. □

Theorem 3.2 implies that if there is one completely labeled Gale string, there is also a second one. The following function problem asks to compute a completely labeled Gale string if one such string is already given.

ANOTHER COMPLETELY LABELED GALE STRING

Input: A labeling $l : [n] \to [d]$, where d is even and $d < n$, and a completely labeled Gale string s in $G(d,n)$.

Output: A completely labeled Gale string s' in $G(d,n)$ where $s' \neq s$.

3.2 Hard-to-solve games and their best reply polytopes

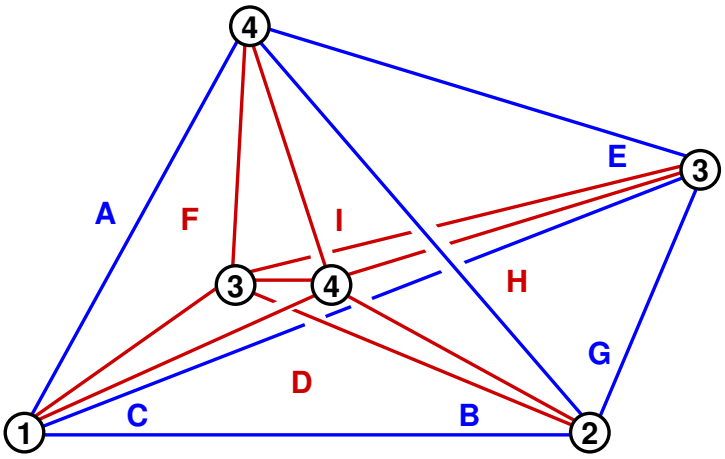
Hard-to-solve bimatrix games are games where the LH-algorithm requires exponentially many steps and where support enumeration takes on average exponential time to find a Nash equilibrium (Savani and von Stengel (2006)). We first describe how to construct a cyclic polytope and how it can be encoded by the set of Gale strings. Next, we explain the connection between general labeled polytopes and equilibria of bimatrix games. Finally, we define Gale games and hard-to-solve bimatrix games. This section is largely based on Casetti, Merschen and von Stengel (2010).

We can construct bimatrix games via cyclic polytopes, see Ziegler (1995) and von Stengel (1999). A *cyclic polytope P* in dimension d with n vertices is the convex hull of n points $\mu(t_j)$ on the *moment curve* μ: $t \mapsto (t, t^2, \ldots, t^d)^\top$ for $j \in [n]$ such that $t_1 < t_2 < \cdots < t_n$. It turns out that we need not be concerned with the actual game that is derived from

this cyclic polytope.[1] Instead, it suffices to study how such a polytope can be encoded in a simple combinatorial way with Gale strings as follows, see von Stengel (1999). The facets of P are encoded by $G(d,n)$, that is,

$$F \text{ is a facet of } P \iff F = \text{conv}\{\mu(t_i) \mid i \in 1(s)\}, \text{ for some } s \in G(d,n), \qquad (3.1)$$

as shown by Gale (1963). For this cyclic polytope P, a labeling $l : [n] \to [d]$ can be understood as a label $l(j)$ for each vertex $\mu(t_j)$ for $j \in [n]$. A completely labeled Gale string s therefore represents a facet F of P that is completely labeled; see Figure 3.2 for an example.



facets	**1**	**2**	**3**	**4**	**3**	**4**
A	1	1	1	1	.	.
B	1	1	1	.	.	1
C	1	1	.	1	1	.
D	1	1	.	.	1	1
E	1	.	1	1	.	1
F	1	.	.	1	1	1
G	.	1	1	1	1	.
H	.	1	1	.	1	1
I	.	.	1	1	1	1

Figure 3.2 A cyclic polytope in dimension 4 with 6 vertices (drawn as a Schlegel diagram) and its Gale encoding $G(4,6)$. The 9 facets are fully described by the Gale strings on the right of the figure. Facets with a red letter are "inside" the polytope; they must use the red edge $(3,4)$. Facets with a blue letter use a face belonging to the outside of the polytope. The facet A is the completely labeled outer facet, corresponding to the artificial equilibrium. One of the other completely labeled facet is D, which lies "inside" the polytope.

The following theorem, due to Balthasar (2009) and Balthasar and von Stengel (2010), establishes a connection between general labeled polytopes and equilibria of certain $d \times n$ bimatrix games (U,B).

Theorem 3.3 *Consider a labeled d-dimensional simplicial polytope Q with $\mathbf{0}$ in its interior, with vertices $-e_1,\ldots,-e_d,c_1,\ldots,c_n$, so that $F_0 = \text{conv}\{-e_1,\ldots,-e_d\}$ is a facet*

[1] To my knowledge, the games that are derived from dual cyclic polytopes (as explained in Theorem 3.3) have no special economic interpretation and were constructed only to give hard instances for the LH-algorithm and support enumeration algorithms.

of Q. Let $-e_i$ have label i for $i \in [d]$, and let c_j have label $l(j) \in [d]$ for $j \in [n]$. Let (U,B) be the $d \times n$ bimatrix game with $U = [e_{l(1)} \cdots e_{l(n)}]$ and $B = [b_1 \cdots b_n]$, where $b_j = c_j/(1 + \mathbf{1}^\top c_j)$ for $j \in [n]$. Then the completely labeled facets F of Q, with the exception of F_0, are in one-to-one correspondence to the Nash equilibria (x,y) of the game (U,B) as follows: if v is the normal vector of F, then $x = (v+\mathbf{1})/\mathbf{1}^\top(v+\mathbf{1})$, and $x_i = 0$ if and only if $-e_i \in F$ for $i \in [d]$; any other label j of F, so that c_j is a vertex of F, represents a pure best reply to x. The mixed strategy y is the uniform distribution on the set of pure best replies to x.

In the preceding theorem, any simplicial polytope can take the role of Q as long as it has one completely labeled facet F_0. Then an affine transformation, which does not change the incidences of the facets of Q, can be used to map F_0 to the negative unit vectors $-e_1, \ldots, -e_d$ as described, with Q if necessary expanded in the direction $\mathbf{1}$ so that $\mathbf{0}$ is in its interior.

A $d \times n$ bimatrix game (U,B) is a *unit vector game* if all columns of U are unit vectors. For such a game B with $B = [b_1 \cdots b_n]$, the columns b_j for $j \in [n]$ can be obtained from c_j as in Theorem 3.3 if $b_j > \mathbf{0}$ and $\mathbf{1}^\top b_j < 1$. This is always possible via a positive-affine transformation of the payoffs in B, which does not change the game. The unit vectors $e_{l(j)}$ that constitute the columns of U define the labels of the vertices c_j. The corresponding polytope with these vertices is simplicial if the game (U,B) is nondegenerate, which here means that no mixed strategy x of the row player has more than $|\{i \in [d] \mid x_i > 0\}|$ pure best replies.

Unit vector games encode arbitrary bimatrix games: An $m \times n$ bimatrix game (A,B) with (w.l.o.g.) positive payoff matrices A,B can be symmetrized so that its Nash equilibria are in one-to-one correspondence to the symmetric equilibria of the $(m+n) \times (m+n)$ symmetric game $(C^\top, C)$ where

$$C = \begin{pmatrix} 0 & B \\ A^\top & 0 \end{pmatrix}.$$

In turn, as shown by McLennan and Tourky (2009), the symmetric equilibria (x,x) of any symmetric game $(C^\top, C)$ are in one-to-one correspondence to the Nash equilibria (x,y) of the *imitation game* (I,C) where I is the identity matrix; the mixed strategy y of the second player is simply the uniform distribution on the set $\{i \mid x_i > 0\}$. Clearly, I is a matrix of unit vectors, so (I,C) is a special unit vector game.

Special games are obtained by using cyclic polytopes in Theorem 3.3, suitably affinely transformed with a completely labeled facet F_0. When Q is a cyclic polytope in dimension d with $d+n$ vertices, then the string of labels $l(1)\cdots l(n)$ in Theorem 3.3 defines a labeling $l' : [d+n] \to [d]$ where $l'(i) = i$ for $i \in [d]$ and $l'(d+j) = l(j)$ for $j \in [n]$. In other words, the string of labels $l(1)\cdots l(n)$ is just prefixed with the string $1\,2\cdots d$ to give l'. Then l' has a trivial completely labeled Gale string $1^d 0^n$ which defines the facet F_0. We call these special games *Gale games*. Note that when we refer to Gale games we assume that we know that the bimatrix game is a Gale game. That is, we do not first have to decide what kind of bimatrix game it is. To my knowledge, the complexity of the decision problem which asks whether a given bimatrix game is a Gale game is unknown.

For Gale games the problem ANOTHER COMPLETELY LABELED GALE STRING defines exactly the problem of finding a Nash equilibrium of the unit vector game (U,B). Note again that B is here not a general matrix, which would define a general game, but obtained from the last n of $d+n$ vertices of a cyclic polytope in dimension d.

For games where both P and Q are the dual of a cyclic polytope the product polytope, $P \times Q$, of P and Q is not usually cyclic anymore. Thus, these games do not belong to the class of Gale games. That is, the product polytope can no longer be represented by one set of Gale strings; it is described by a concatenation of two independent Gale strings. In particular, finding a Nash equilibrium is then not equivalent to the problem ANOTHER COMPLETELY LABELED GALE STRING.

Savani and von Stengel (2006) construct bimatrix games where both players have dual cyclic polytopes as their best response polytopes. They call these games *$m \times n$-double cyclic polytope games*. For *square games*, where $m = n = d$ and where the labels are derived from Morris's construction (see Morris (1994) and Section 4.4 for the introduction of these labels), the LH-algorithm takes exponentially many steps to find an equilibrium. However, square games are not "hard to solve" by the support enumeration algorithm; see Savani (2006). The support enumeration algorithm considers strategies of the players with equal support size and checks whether they are best replies to each other. Since square games have a unique completely mixed equilibrium, where both players play d pure strategies with positive probability, the support enumeration algorithm terminates quickly.

Games that are derived from certain dual polytopes where the LH-algorithm takes exponentially many steps in the dimension d of the polytope to find another fully labeled facet and for which support enumeration algorithms take exponential long as well, are then called *hard-to-solve bimatrix games*. These games can be thought of as a set of games that is defined by a sequence of polytopes where the LH-algorithm performs badly. Note that a game with 8 strategies for which the LH-algorithm takes, say, 60 steps is not immediately classified as hard-to-solve unless the duals of the best response polytopes are cyclic and display Morris's pattern, in addition to taking long to solve with support enumeration algorithms. There might be a larger set of games that are "hard-to-solve" but which use different methods to construct these.

One class of hard-to-solve games is constructed from $3d \times d$ games with one cyclic polytope of dimension d and one *simplotope*, which is a product of simplices, here d tetrahedra (Savani (2006)). He calls these *triple imitation games*, due to the connection to imitation games. Nash equilibria of triple imitation games are fully described by completely labeled Gale strings in $G(d, 4d)$. They are thus Gale games and an equilibrium can be found via the problem ANOTHER COMPLETELY LABELED GALE STRING. Prior to our results, described in the next section, no polynomial algorithm was known to find an equilibrium for the class of Gale games.

3.3 Euler graphs, perfect matchings, and Gale strings

We would like to determine the complexity of the function problem ANOTHER COMPLETELY LABELED GALE STRING; introduced in Section 3.1. We reformulate the problem into a graph problem.

A *graph* $G = (V, E)$ is a set of *vertices* V and a collection of *edges* E, where an edge (u, v) connects a pair of vertices, u and v. A *multigraph* allows for more than one edge between a pair of vertices, which we then call *parallel edges*. If the edges are directed, the graph is referred to as a *directed graph*, or *digraph*. A directed edge (u, v) is directed from u to v, where u is the *tail* and v is the *head* of the edge; for an undirected graph the order of endpoints does not matter. A vertex of a directed (multi)graph is *balanced* if it has the same number of heads and tails incident to it, in other words the same number of

incoming and outgoing edges. A graph is *connected* if there is a *path* p of edges between any two vertices of the graph; we assume here that all graphs are connected.

A graph $G = (V, E)$ is an *Euler* or *Eulerian* graph if each vertex v has an even number of adjacent vertices. An *Euler tour*, Et, of a connected graph is a cycle that traverses each edge exactly once and where vertices may be revisited (thus tour). A graph admits an Euler tour if and only if the graph is Eulerian (Hierholzer (1873)). The existence of an Euler tour can be easily checked by inspecting the number of neighbours of every vertex and can be efficiently computed using Hierholzer's algorithm.

Given a graph $G = (V, E)$ a set $M \subseteq E$ of pairwise non-adjacent edges is referred to as a *matching*. As a convention, when drawing graphs, edges in M are coloured red and are dashed while all other edges are black. A path is called *alternating* with respect to M if its edges alternate between edges from M and $E \setminus M$. A *free* vertex is a vertex not adjacent to an edge from M. An *augmenting* path is a simple alternating path between two free vertices. We think of a path as a set of edges when useful. A set M is a *maximum* matching if there exists no matching with more edges. A matching M is called *perfect* if every vertex $v \in V$ is incident to exactly one edge in M.

A perfect matching in a bipartite graph, if it exists, can be efficiently found by the augmenting paths algorithm. Set $M = \emptyset$ and find an augmenting path p, for example using breadth-first search. Next "flip" the edges of the path from black to red, i.e. delete the edges $M \cap p$ from M and add the unmatched edges of p to M. This amounts to setting M to $M \oplus p$, where $\oplus$ is the *symmetric difference* operator defined for two sets of edges U, V as $U \oplus V = \{e : \ e \in U \text{ or } e \in U \text{ and } e \notin (U \cap V)\}$. Repeat this argument until no augmenting path can be found. The resulting M either describes a perfect matching or the given graph does not admit one. A maximum matching has at most size $|V|/2$ and the size of M is increased by one after an augmenting path is found. It takes $O(|E|)$ time to find an augmenting path using breadth-first search, resulting in an overall time complexity of $O(|E||V|)$. The correctness of the algorithm follows from Berge's (1957) theorem stating that a matching is maximum if and only if the graph admits no augmenting path with respect to M.

For general, non-bipartite, graphs the augmenting paths algorithm can, however, lead to non-simple augmenting paths. By contracting odd cycles, so called *blossoms* into a *super vertex* b^*, Edmonds's "Paths, Trees, and Flowers" algorithm can be used to decide

the following decision problem in polynomial time (Edmonds (1965)). We consider this algorithm in more detail in Section 4.1.

PERFECT MATCHING

Input: Graph $G = (V, E)$.

Question: Is there a set $M \subseteq E$ of pairwise non-adjacent edges so that every vertex $v \in V$ is incident to exactly one edge in M?

Above, a perfect matching is defined for an undirected graph. In this thesis, a perfect matching for a directed graph is defined analogously by requiring that every vertex $v \in V$ is incident to a directed edge from M, independent of whether the tail or head of the edge is incident to v. When we refer to a perfect matching, we thus do not distinguish between a directed and an undirected perfect matching, unless we require a perfect matching of a certain sign (see Section 3.6). The notion of perfect matchings for multigraphs, directed or undirected, is defined analogously. We show at the end of the this chapter that we only need to consider graphs that have no multiple edges.

A labeling $l : [n] \to [d]$ can be interpreted as a special directed multigraph with vertex set V and edge set E as follows. Let $G = (V, E)$ where $V = [d]$ and with up to n, possibly parallel, directed edges given by $(l(i), l(i+1))$ for $i \in [n]$ whenever these endpoints are distinct; this rules out that G has loops since we just ignore repetitive labels where $l(i) = l(i+1)$. We let $n+1 \equiv 1$ so that $n, n+1$ is to be understood as $n, 1$.

The resulting graph G is an Euler graph as every vertex has an even number of edges. This holds because a label $l(i)$ has two neighbours, one to the left and one to the right of $l(i)$ in the label string. Equivalently, a resulting Euler tour (there can be more than one and in fact exponentially many, see Mihail and Winkler (1992)) is given by the label string l without considering direct repetitions of labels. Let this Euler tour specify an orientation of the edges of the Euler graph, thus the Euler graph is directed. Unless we state otherwise an Euler graph that is derived from a label string is a directed Euler graph where the orientation of the edges is given as described above. Figure 3.3 illustrates the relationship between a label string 112345643, its directed Euler graph and its perfect matching.

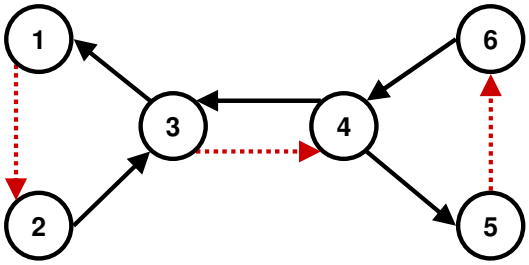


Figure 3.3 For the label string 112345643 the corresponding directed Euler graph is given by vertices 1 through 6, where the loop at vertex 1 is disregarded. The dashed red edges are one perfect matching in the graph.

The following Theorem 3.4 (Casetti, Merschen and von Stengel (2010)) describes the relationship between completely labeled Gale strings and perfect matchings in the resulting Euler graph.

Theorem 3.4 *Given a labeling $l : [n] \to [d]$, and its corresponding Euler graph $G = (V,E)$, there exists a one-to-one correspondence between completely labeled Gale strings s in $G(d,n)$ and perfect matchings M in G.*

Proof: Let s be a completely labeled Gale string in $G(d,n)$. The Gale evenness condition allows to split s into a number of runs which are uniquely split into $d/2$ pairs $i, i+1$ so that the labels $l(i)$ and $l(i+1)$ are distinct, and all labels $1,\ldots,d$ occur among them. So this defines a perfect matching for G.

Conversely, a perfect matching M of G defines a Gale string s where $s(i) = s(i+1) = 1$ if the edge that joins $l(i)$ and $l(i+1)$ is in M and $s(i) = 0$ otherwise; so s is completely labeled. Finally, if there exists no completely labeled Gale string for l then the graph does not admit a perfect matching. □

The reduction from a labeling l to a graph is polynomial in the size of the labeling as the graph has d vertices and at most, disregarding loops, n edges. The proof of Theorem 3.4 thus gives a polynomial-time reduction from COMPLETELY LABELED GALE STRING to PERFECT MATCHING. Because PERFECT MATCHING is in P (Edmonds (1965)), Corollary 3.5 follows.

Corollary 3.5 *The decision problem* COMPLETELY LABELED GALE STRING*, and thus deciding whether a labeled cyclic polytope has a completely labeled facet, can be decided in polynomial time.*

We make use of Theorem 3.2 and Theorem 3.4 to prove the following.

Theorem 3.6 *The function problem* ANOTHER COMPLETELY LABELED GALE STRING *can be solved in polynomial time.*

Proof: Consider the given completely labeled Gale string s and the matching M for it. Another completely labeled Gale string s' exists by Theorem 3.2. The corresponding matching M' does not use at least one edge in M. Hence, at least one of the $d/2$ graphs G which have one of the edges of M removed has a perfect matching M', which is a perfect matching of G, and which defines a completely labeled Gale string s' different from s. As search for M' takes again polynomial time, the result holds. □

A Nash equilibrium in a unit vector game (U,B) can be characterised as a fully labeled facet of the polytope Q from Theorem 3.3. If Q is a cyclic polytope, then these games belong to the class of Gale games (see Section 3.2). For Gale games a Nash equilibrium is given by a completely labeled Gale string. Finding a Nash equilibrium for Gale games is equivalent to ANOTHER COMPLETELY LABELED GALE STRING.

Theorem 3.7 *Finding a Nash equilibrium in Gale games takes polynomial time.*

Proof: Consider the labeled cyclic polytope of the Gale game as described in Section 3.2. The artificial Nash equilibrium is given by a completely labeled facet F_0 and has a corresponding completely labeled Gale string in the Gale representation of the polytope. Since finding ANOTHER COMPLETELY LABELED GALE STRING is in FP, another completely labeled facet F_1, which corresponds to a Nash equlibrium, is found in polynomial time. Since all reductions are polynomial in the size of the problem, a Nash equilibrium in Gale games is found in polynomial time. □

This result has an interesting implication. The Nash equilibrium that is found via the translation to perfect matchings above is not necessarily the same Nash equilibrium that is found by the LHG-algorithm for Gale strings. This holds because Edmonds's algorithm

picks out one of the other perfect matchings while the LHG-algorithm finds a specific one, described in Section 3.4. Also see Section 3.4 for a discussion about the complexity of finding a specific Nash equilibrium or a specific completely labeled Gale string.

We conclude this section by showing that a multigraph can be translated in polynomial-time into a simple graph without affecting the Euler property and without changing the number of perfect matchings. This allows us to state all proofs for simple graphs. Let $p_1, ..., p_k$ be the parallel edges between two vertices u and v. Replace each edge $p_i = (u, v)$ for $i \in [k]$ with two new vertices u_i and v_i and connect them with edges (u, u_i), (u_i, v_i) and (v_i, v). Similarly, for a parallel edge directed from v to u the new edges are oppositely directed. This creates $2k$ more vertices and edges. This construction does not change the number of perfect matchings. If p_i is in the matching of the multigraph then this corresponds to (u, u_i) and (v_i, v) being part of the matching in the simple graph. For the remaining edges in the simple graph this means that (u_j, v_j) or (v_j, u_j) for $j \neq i$ must be in the matching. Hence, (u, u_i) and (v_i, v) are in the matching if and only if p_i is in M. Clearly, the number of perfect matchings is not affected by "simplifying" the graph.

3.4 Orienting Gale strings

We showed in Section 3.1 that the number of completely labeled Gale strings is even. The question is, can we further characterise the set of completely labeled Gale strings. Shapley (1974) defined the *index* of a Nash equilibrium in a nondegenerate bimatrix game and showed that the two equilibria at the ends of a LH-path have opposite index. By convention, the artificial equilibrium has index -1; any pure-strategy equilibrium has index $+1$. The sum over the indices of all equilibria is $+1$. The index is defined in terms of the signs of the determinants of the square submatrices of the payoff matrices (which are assumed to be positive) for the equilibrium support; see Shapley (1974) or von Schemde and von Stengel (2008) for details.

Let S_n be a *symmetric group*, σ a *permutation* in S_n, and $sign(\sigma)$ the *signature function* or *sign* of σ; see Graham, Groetschel, and Lovász (1996) for an introduction. The sign of a permutation σ is defined as $sign(\sigma) = (-1)^{N(\sigma)}$, where $N(\sigma)$ is the number of inversions of σ. An *inversion* of a permutation $\sigma(1), ..., \sigma(n)$ of $1, ..., n$ is a pair i, j so that $i < j$ but $\sigma(i) > \sigma(j)$. A permutation is said to have sign $+1$ (-1) if it has an

even (odd) number of inversions. Thus exchanging pairs, for example $(\sigma(1),\sigma(2))$ with $(\sigma(5),\sigma(6))$, does not affect the sign of the permutation. Alternatively, the sign of a permutation σ can be defined by using the number of transpositions. A *transposition* is the exchange of exactly two elements of a sequence. The sign of a permutation σ is then given by $sign(\sigma) = (-1)^m$, where m (which does not have to be unique) is the number of transpositions needed to write σ as the sequence $1,...,n$. Note that any two permutations that differ in one transposition have opposite sign. We assume throughout that n is even and use the sign function not only for permutations but also for Gale strings and perfect matchings (which we discuss in detail in Section 3.6).

Motivated by the index of an equilibrium, we define the *sign* for a Gale string as follows. For a completely labeled Gale string s in $G(d,n)$, let l' be the concatenation of substrings of l where a label $l(i)$ is present in l' if $s(i) = 1$, such that two labels adjacent in a run in l are again adjacent in l' (the substrings are thus of even length). Labels $l(j)$ satisfying $s(j) = 0$ are thus not represented in l'. If s is a completely labeled Gale string the sign of s, $sign(s)$, is $+1$ or -1 (positive or negative) if the number of inversions in l' is even or odd, respectively. For example, for the labeling $l = 123424$, with $d = 4$, one completely labeled Gale string is 111100. For this completely labeled Gale string, l' is constructed by taking the labels of the run of 1's (here 1111), hence $l' = 12\ 34$. Thus s has positive sign as $l' = 12\ 34$ has an even (here zero) number of inversions. The other completely labeled Gale string 111001 has negative sign as $l' = 41\ 23$ has an odd number of inversions. We use $sign(l')$ instead of $sign(s)$ for completely labeled Gale strings when suited.

Another way to obtain l' is by splitting s into $d/2$ pairs of "11"; the Gale evenness condition guarantees that s can be uniquely split in such a way. Then the labels associated with "11" form l'. The sign of s is then defined as the sign of the permutation of labels for the "11" positions in s. Note that the order of the $d/2$ pairs does not alter the sign as pointed out earlier in this section. This alternative definition of the sign of s is related to the sign of a perfect matching, where the '11' positions are the matched edges; see Section 3.6.

Our definition of the sign does not keep the convention that when the labeling is derived from the best reply polytopes of a bimatrix game, here a Gale game, the sign of a completely labeled Gale string is equal to the index of the Nash equilibrium; see

von Schemde and von Stengel (2008) for a definition of the index. For notational purposes a positive sign corresponds to a negative index and vice versa. In particular the artificial equilibrium has a negative index and thus a positive sign in the Gale setting. In the example above, the completely labeled Gale string corresponding to the artificial equilibrium, here $s = 111100$, thus has positive sign.

In order to show that completely labeled Gale strings display opposite signs when they are the endpoints of a pivot path we need to amend Algorithm 1 to incorporate an orientation for an almost labeled Gale string. If s is a completely labeled Gale string, then l' is obtained as above. If s is an almost completely labeled Gale string it has exactly one duplicate label, which we need to treat differently. The sign of l' is then defined as follows. Let l'_1 and l'_2 be obtained by substituting the missing label for the different duplicate labels in l'. The sign of l'_1 is positive if the number of inversions in l'_1 is even. Otherwise, the sign of l'_1 is negative. The sign of l'_2 is defined analogously. Note that l'_1 and l'_2 have opposite signs, because l'_1 can be obtained from l'_2 (and vice versa) by exchanging exactly two elements – the duplicate label and the missing label of l. It is easy to see that this results in different signs of l'_1 and l'_2; as described earlier in this section. We give an example for the label string $l = 123424$. For the almost labeled Gale string $s = 100111$ the associated labeling is $l' = 42\ 41$ with missing label 3. This has positive or negative sign depending on whether we consider $l'_1 = 32\ 41$ or $l'_2 = 42\ 31$, obtained by either of the duplicate labels 4 replaced by the missing label 3. We are now ready to define the oriented form of the LHG-algorithm, see Algorithm 2.

By the same reasoning as in Chapter 3.1, Algorithm 2 terminates in finite time, returning a different Gale string s'. We show that $sign(s) \neq sign(s')$ as part of the proof of Theorem 3.8. The exact complexity of the LHG-algorithm is an open question. In Section 4.4 we explain that for any positive and even integer d, there exists a n-length labeling, where the labels are taken from $[d]$, for which the LHG-algorithm takes exponentially many steps in d; see Morris (1994).

Theorem 3.8 *For any labeling $l : [n] \to [d]$, where d is even and $d < n$, the number of completely labeled Gale strings in $G(d,n)$ with positive sign is equal to the number of the negative signed ones.*

Algorithm 2: LHG-algorithm (oriented)

input : A labeling $l : [n] \to [d]$, where d is even and $d < n$, and a completely labeled Gale string s in $G(d,n)$.

output: A completely labeled Gale string s' in $G(d,n)$ where $s' \neq s$, with $sign(s) \neq sign(s')$

```
pivot an arbitrary label of s;
while (l is not completely labeled by s) do
    if (sign(l'_1) = sign(s)) then
        pivot the label in l'_1, substituted by missing label;
    else
        pivot the other duplicate label;
return s' = s
```

Proof: By Theorem 3.2 the number of completely labeled Gale strings is even. It remains to show that the number of completely labeled Gale strings in $G(d,n)$ with positive sign is equal to the number of the negative signed ones. Fix a label $w \in [d]$ and consider the graph where vertices are completely labeled Gale strings and where each almost completely labeled Gale string with missing label w receives two vertices.[2] Additionally, each vertex in the graph receives a sign. Completely labeled Gale strings s receive their sign, that is $sign(s)$. The two vertices representing an almost completely labeled Gale strings receive positive and negative sign, respectively.

The edge set of the graph is given as follows. Connect the two vertices corresponding to one almost completely labeled vertex with a blue edge. That is, connect the positive signed with the negative signed vertex corresponding to one almost completely labeled Gale string. Also, connect any two vertices that have $d-1$ identical labels and which can be reached by a pivot with a red edge. This defines a bipartite graph as we argue below.

We first show that the vertices at the ends of a red edge have a different sign. A pivot is between two Gale strings, $\hat{s}$ and $\bar{s}$; drop a label from $\hat{s}$ and pick up a label in $\bar{s}$. Which label we drop and pick up defines the sign of a Gale string when it is not completely labeled.

[2]Note that this is not the w-skew graph defined in Theorem 3.2 as each almost labeled Gale string received two vertices.

In a pivot from a completely labeled Gale string $\hat{s}$, the dropped label "jumps" over an odd number of ones to pick up another label in $\bar{s}$. If $\bar{s}$ is completely labeled then it clearly has a different sign from $\hat{s}$. Otherwise, $\bar{s}$ is an almost completely labeled Gale string. Substitute the new duplicate label (that was just picked up) with the missing label, giving $\bar{s}$ opposite sign from $\hat{s}$. Next, consider a pivot between two almost completely labeled Gale strings, $\hat{s}$ and $\bar{s}$. Substitute the label that was dropped from $\hat{s}$ with the missing label and compute the sign. The dropped label jumps over an odd number of ones to reach $\bar{s}$. Substitute the newly picked up label in $\bar{s}$ with the missing label and compute the sign, which is different from the sign of $\hat{s}$. When pivoting from an almost completely labeled Gale string to a completely labeled one the change of sign follows similarly. Clearly this defines a bipartite graph as all edges connect a positive and a negative signed vertex.

Every vertex in the bipartite graph has either degree one or two. Hence, this undirected graph is a collection of paths and cycles. Nodes with degree one, the endpoints of the path, are the completely labeled vertices and are only incident to a red edge. The other vertices are also incident to a blue edge. Any path starts and ends with a red edge. This path must be of odd length since the colour alternates between red and blue. The endpoints, corresponding to completely labeled Gale strings, of any path must thus have opposite signs. □

Theorem 3.8 shows that the endpoints of the pivot path connecting two completely labeled Gale strings have opposite signs. The argument here is similar to the argument given by Lemke and Grotzinger (1976) and Todd (1976). Their orientation can be proved by a suitable orientation of the facets of the polytope, via the determinant of the d vertices in the order of the labels.

One implication of Theorem 3.8 is that the problem ANOTHER COMPLETELY LABELED GALE STRING belongs to the class PPADS, as we are only searching for completely labeled Gale string of a certain sign. For this we need to give path consisting of only red edges (this is the pivot path), used to prove Theorem 3.8, a direction at every vertex, as follows. Note that only the red edges used in the construction of the bipartite graph are pivots, the blue edges only "decide" which of the two duplicate vertices to drop next. Suppose one is at an almost completely labeled Gale string $\hat{s}$. To give the next red edge a direction one needs to know which duplicate label to drop next, without knowing which of the two was the duplicate label just picked up. In other words we need to

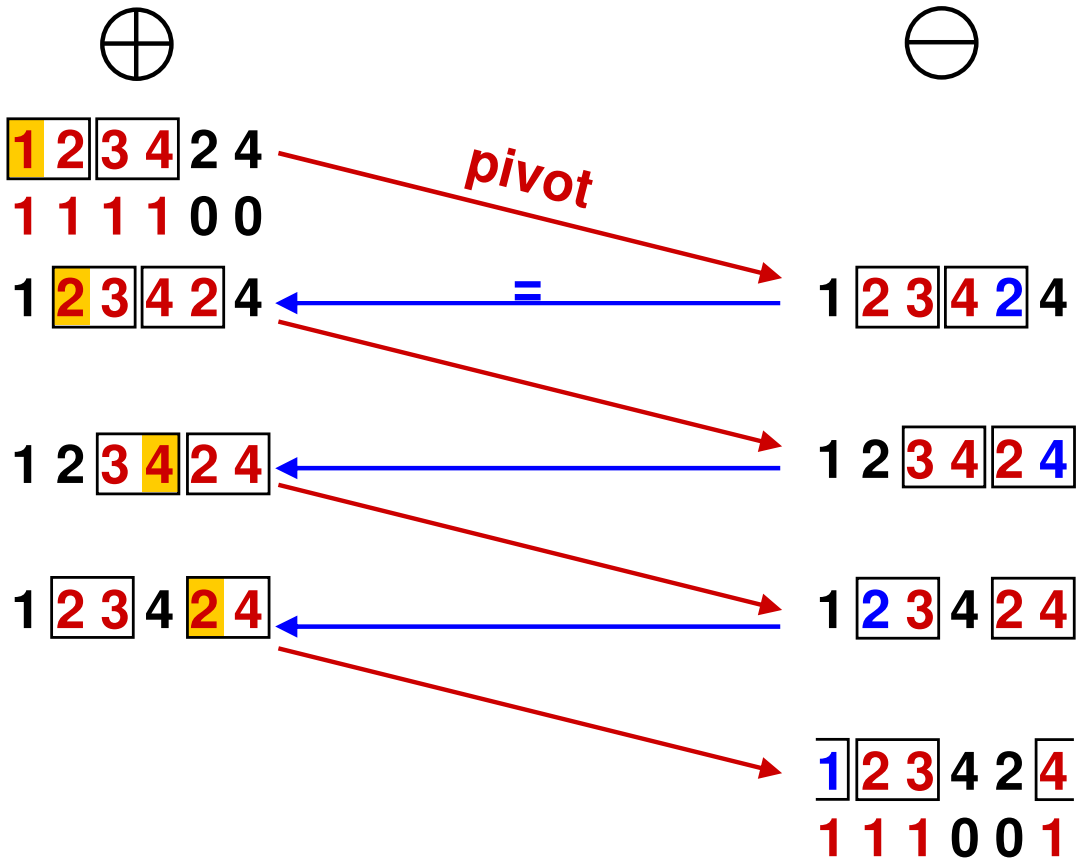


Figure 3.4 The signed pivot path for 123424 when label $l(1) = 1$, boxed in yellow, is initially dropped. Note that the directed pivot path is given by the red edges; the blue edges are only needed to "decide" which duplicate label to drop next. Going from left to right, following the red arrow, leads to $l' = 23\ 42$. The missing label 1 jumped over 234 to pick up the new duplicate label 2, in blue, where the boxes represent the pairs of "11". Substituting the missing label for the duplicate label just picked up the sign of, say l'_2, is negative. Going from right to the left along the blue arrow we "decide" which duplicate label to pivot next. This is always the "old" one, where the sign of l'_1 is equal to $sign(s)$, again boxed in yellow. Finally the missing label is picked up by a pivot following a red arrow, resulting in $sign(s) \neq sign(s')$.

know which duplicate label to pivot, starting from any arbitrary vertex. This is decided by evaluating the sign of l'_1 and l'_2 of $\hat{s}$. Drop the duplicate label substituted by the missing label in l'_1 if $sign(l'_1) = sign(s)$ where s is the start of the path. Otherwise drop the other duplicate label. The pivot edge that is chosen receives a forward orientation, i.e. its tail is adjacent to $\hat{s}$. This "check" is incorporated in Algorithm 2 in line 3 and corresponds to a blue edge in Figure 3.4. Deciding which label to pivot can be computed in polynomial

time; one only needs to compute the sign of l_1 and l_2. This is exactly the "polynomial" in the class PPAD and PPADS. The orientation follows because the correct pivot can be made without knowledge which of the two duplicate labels was just picked up. It suffices to know the sign of the initial Gale string s, given by the input, to continue in the correct direction at any point of the algorithm. Algorithm 2 follows exactly such a path from the path-following argument used to prove Theorem 3.8; see Figure 3.4 gives an illustration of a pivot path. The red directed edges are the pivots and the blue edges represent the decision which of the two duplicate labels to pivot next.

Even though Gale games may give rise to exponentially long pivoting paths, the respective computational problem of finding ANOTHER COMPLETELY LABELED GALE STRING is solvable in polynomial time (see Theorem 3.6). Hence, Gale games are most likely too simple to define a PPAD-complete problem, unless PPAD = P, which is believed to be unlikely (Daskalakis, Goldberg and Papadimitriou (2009)).

Note that with Edmonds's approach we find an equilibrium which can be either a sink or a source in the PPAD sense. That is, it can be an equilibrium of either sign. In particular, it does not necessarily have to be an equilibrium of opposite sign, that is, with the same sign as the equilibrium at the other end of the pivot path given by Algorithm 2, the directed LHG-algorithm. (See Chapter 7 for a discussion about the complexity of computing exactly the equilibrium at the other end of the pivot path.)

3.5 Euler graphs and 1-oiks

In Section 3.3 we showed that the label string associated with a Gale string can be interpreted as an Euler graph. Euler graphs are a special, low-dimension case of an *Euler complex* or "oik" as defined by Edmonds (2007). Concepts such as the skew-graph, used to prove Theorem 3.2, and results showing that the number of perfect matchings in Euler graphs is even (see Section 3.3) translate to oiks.

An Euler complex or *oik* is given by a set of vertices V and a family $\mathcal{R}$ of d-size subsets of V called *rooms* such that any set of $d-1$ vertices is contained in an even number of rooms. It is assumed that $d > 1$. Edmonds (2007) calls this a $(d-1)$-oik while Edmonds, Gaubert and Gurvich (2009) call it a d-dimensional oik; we use Edmonds's original wording. We allow for *parallel* rooms which have the same vertices, hence $\mathcal{R}$ is

in general a family or multiset rather than a set. For a set R and u in R we write $R-u$ for $R-\{u\}$. If R is a room we call $R-u$ a *wall* of R. By the oik property, $R-u$ is contained in an odd number of rooms other than R.

We give an example. A 1-oik is an *Euler graph* where $\mathcal{R}$ is a family of edges so that every vertex is the endpoint of an even number of edges. Parallel edges are allowed, although we can restrict ourselves to simple edges; see Section 3.3. We assume the graph is connected, so it has an Euler tour. If not, all our considerations can be applied separately to each connected component. The Euler tour defines an *orientation* of each edge so that the in-degree and out-degree of each vertex is the same. We write a directed edge from vertex u to vertex v as (u,v).

A labeling $l:[n]\to[d]$ defines an Euler graph with vertex set $[d]$ and an Euler tour with edges $l(i)l(i+1)$ for $1\le i<n$ and $l(d)l(1)$; see Section 3.3. The digraph in Figure 3.5 is obtained for the labeling $[6]\to[4]$ written as a string of labels $l=123424$. We can

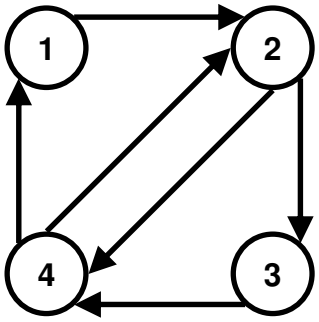


Figure 3.5 The digraph for Euler tour 123424.

distinguish the two edges 24 and 42 by their orientation even though they are parallel in the unoriented version of this digraph as a 1-oik.

Given an oik, a *room partitioning* is a partition of the vertex set V into pairwise disjoint rooms. The rooms in the 1-oik in Figure 3.5 are edges, the walls are the vertices of the graph and the sets $\{12,34\}$ and $\{41,23\}$ represent the two room partitions. Thus for a 1-oik the room partitions are perfect matchings of the Euler graph. As shown in Section 3.3, these perfect matchings are exactly the completely labeled Gale strings for the corresponding label string.

Edmonds also considered the more general case that each element of the partition may be allowed to come from a separate oik with the same vertex set V. However, we consider only a single oik. The following theorem is due to Edmonds (2007).

Theorem 3.9 *In an oik, the number of room partitionings is even.*

Proof: Fix a vertex w. Consider the *exchange graph* whose vertices are all room partitionings and the so-called *w-skew room partitionings*. A w-skew room partition is a set of rooms which are subsets of $V - w$ so that any two of these rooms are disjoint, except for one pair of rooms that have a single vertex u, not equal to w, in common.

The neighbors of a *room partitioning* in this exchange graph are defined as follows: Take the room R that contains w and consider the wall $R - w$. That wall is contained in an odd number of rooms R' other than R; replacing R by any of these rooms R' gives either a neighbouring room partitioning, in case R' is parallel to R, or, more typically, a neighboring w-skew room partitioning. Thus, a room partitioning has always an odd number of neighbours in the exchange graph.

The neighbors of a *w-skew room partitioning* in the exchange graph are obtained as follows: Consider the unique pair R, R' of rooms that intersect in the single vertex u. These two rooms are not parallel because they would have other vertices in common; consequently, $R - u$ and $R' - u$ are two different sets. There is an odd number of rooms that share the wall $R - u$ with R, and replacing R with such a room gives a neighboring w-skew room partitioning or room partitioning, and similarly there is an odd number of such neighbors in the exchange graph when considering R' instead of R.

So the w-skew room partitionings in the exchange graph have an even number of neighbors, whereas the room partitionings have an odd number of neighbors. Because the number of odd-degree nodes in a graph, here the exchange graph, is even, there is an even number of room partitionings. □

One method to compute another room partition from an existing one is to walk along a path in the exchange graph such that no edge is repeated. Here an edge connects two neighbouring nodes of the exchange graph. Edmonds (2007) calls this the *exchange algorithm*. As a special case, consider a d-oik where every set of d vertices is in exactly zero or two rooms; Edmonds (2007) calls these *d-dimensional simplicial pseudo-manifolds*.

An example is a connected Euler graph where every vertex has exactly two neighbours. In this case the exchange graph consists of disjoint cycles and simple paths, showing that finding another room partition is in PPA. The outcome of the exchange algorithm is thus uniquely determined by the initial room partition and the initially dropped vertex.

On the other hand, if a set of d vertices can be in more than two rooms then the exchange graph does not consist of simple paths anymore. Thus, at some steps we may have a choice of how to continue and we can pick one of the odd many neighbouring vertices of the exchange graph. Depending on which neighbour we picked, the other endpoint of the path can vary. Suppose we visit a w-skew room partition $V_1 = V - w$ by following an edge e and leave V_1 with edge e'. Because we cannot repeat an edge in the exchange algorithm if we revisit V_1 then this has to be with another edge, say $\hat{e}$, different from e or e'. Similarly, we cannot leave V_1 with e or e'. The choice we make at V_1 can be seen as a pairing of edges, here (e, e') and some other pairing for $\hat{e}$. We elaborate this argument for Euler graphs in Section 4.2.

Edmonds (2007) and Edmonds, Gaubert and Gurvich (2009) introduce two special variants of oiks that are relevant to our setting. The first are Gale oiks, which are defined via a d-dimensional cyclic polytope with n vertices. In a d-dimensional *Gale oik* a room corresponds to a facet and a room partition is then a set of facets such that every vertex of the polytope is on exactly one facet.

The second special case is a 1-oik which is equivalent to an Euler graph, as explained earlier in this section. Due to the equivalent characterisation of completely labeled Gale strings, perfect matchings of the corresponding Euler graph which are in turn room partitions in the 1-oik, the sign of a Gale string transfers to an 1-oik. This equivalence allows to extend Theorem 3.9 to incorporate an orientation.

Corollary 3.10 *In a 1-oik, the number of room partitionings with positive sign is equal to the number with negative sign.*

Extending the sign to oiks of dimension higher than $d = 1$ proves to be more difficult, and it is in fact an open problem whether this can be achieved. The following 3-dimensional example, see the left part of Figure 3.6, has four room partitionings indexed by given pairs of facets A, B, C and D, where the outer A can be seen as the bottom of the octahedron.

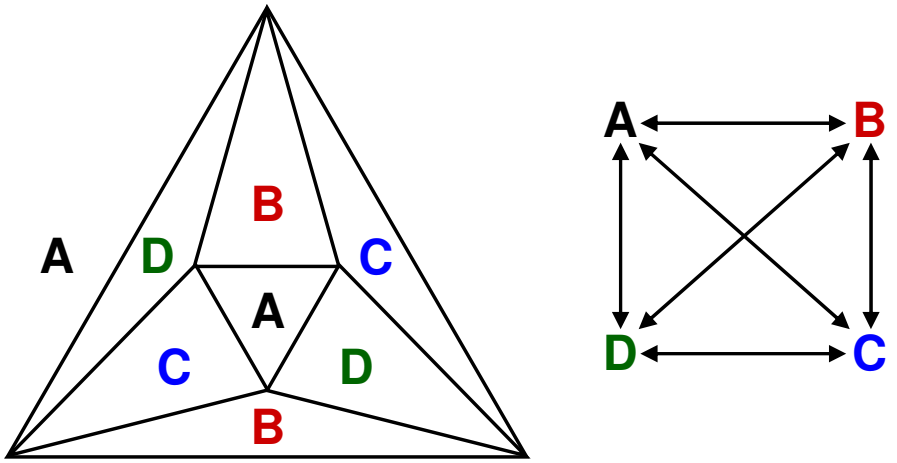


Figure 3.6 A 2-oik and its exchange graph.

The right graph of the figure gives the connections between the room partitionings. Two room partitionings are connected by a path, here simply an edge, if they are obtained by dropping a vertex w and reaching the other room partition via the exchange algorithm. The exchange graph is given by K_4, the complete graph with four vertices, as every room partition can be reached from every other partition by dropping a suitable label. Clearly, as the graph is fully connected, it is not easily possible to consistently assign a sign to a room partition such that connected partitions have opposite sign. Considering a room partition R together with an initial dropped vertex w may result in a consistent definition of a sign as a function of both R and w, which would have to be further explained.

3.6 Signed perfect matchings

We have seen that perfect matchings in Euler graphs correspond to completely labeled Gale strings where the label string is derived by an Euler tour of the graph; see Section 3.3. The sign of a Gale string thus translates to perfect matchings in Euler graphs. The sign of a perfect matching was first implicitly introduced by Cayley in 1852 as part of the definition of a Pfaffian.

Let $A \in \{-1,0,1\}^{2k\times 2k}$ be a *skew-symmetric matrix*, that is, the entries of A satisfy $a_{ij} = -a_{ji}$ for all $i,j \in [2k]$, where $2k = n$. We usually associate A with the adjacency

matrix of a directed graph $G = (V,E)$ with n vertices by

$$a_{ij} = \begin{cases} 0 & \text{if } (i,j) \notin E, \\ 1 & \text{if } (i,j) \text{ is directed from } i \text{ to } j \\ -1 & \text{if } (i,j) \text{ is directed from } j \text{ to } i, \end{cases} \tag{3.2}$$

where A clearly satisfies the skew-symmetric property. The *Pfaffian* of A is defined by

$$Pf(A) = \frac{1}{2^k k!} \sum_{\sigma \in S_n} sign(\sigma) \prod_{i=1}^{k} a_{\sigma(2i-1)\sigma(2i)}, \tag{3.3}$$

where S_n is the symmetric group, σ a permutation in S_n and $sign(\sigma)$ the sign of σ.

In equation (3.3) the term $\prod_{i=1}^{k} a_{\sigma(2i-1)\sigma(2i)}$ is nonzero if and only if the permutation σ is such that for each $i = 1,\ldots,n$, the pair $(\sigma(2i-1),\sigma(2i))$ is a directed edge of the graph, that is, these edges form a perfect matching, because otherwise one of the terms $a_{\sigma(2i-1)\sigma(2i)}$ of the adjacency matrix would be zero. The sign of a perfect matching σ is then given by $sign(\sigma)$. In addition, there are $2^k k!$ ways to represent each matching by such a permutation σ: By writing the k edges of the matching in any of $k!$ possible orders, and by writing down each edge in one of two ways as either $(\sigma(2i-1),\sigma(2i))$ or $(\sigma(2i),\sigma(2i-1))$. Exchanging the two endpoints of one edge changes the sign of σ and also of $a_{\sigma(2i-1)\sigma(2i)}$, so for all $2^k k!$ permutations σ that represent the same perfect matching in equation (3.3) the term $sign(\sigma)\prod_{i=1}^{k} a_{\sigma(2i-1)\sigma(2i)}$ has the same sign.

The Pfaffian has the following interpretation. Given a graph and some orientation then $Pf(A)$ counts the number of matchings. However, since some perfect matchings might have positive sign and others negative sign, perfect matchings can cancel out in the sum, in which case the Pfaffian does not correctly capture the number of perfect matchings. The Pfaffian only gives the correct number of perfect matchings of a graph if all perfect matchings are of the same sign, which is then called a Pfaffian orientation. We address this property in Section 5.2. Note, that the Pfaffian can be efficiently computed; see, for example, Rote (2001).

In the context of Euler graphs, whose Euler tour defines an orientation of the graph, the following corollary follows from Theorem 3.8.

Corollary 3.11 *If an Euler graph $G = (V,E)$ admits a perfect matching then the number of all perfect matchings is even and the number of positive signed perfect matchings equals the number of negative signed perfect matchings.*

As explained above, the sign of a perfect matching is part of the definition of the Pfaffian. We would like to use this fact to give an alternative proof of Corollary 3.11. For this we need to consider the relationship between the determinant and the Pfaffian of a skew-symmetric matrix; this is stated in Lemma 3.12 which is due to Muir (1883).

Lemma 3.12 *If A is a skew symmetric matrix, then $Pf(A)^2 = det(A)$.*

Proof of Corollary 3.11: Since G is an Euler graph, the skew adjacency matrix has an equal number of plus and minus 1's in every row and column. It follows that $A \cdot \mathbf{1} = \mathbf{0}$, showing that the kernel of A is non-empty. In other words A is singular and $det(A) = Pf(A)^2 = 0$. Thus, for Euler graphs the Pfaffian of its skew-adjacency matrix is always zero. From the definition of $Pf(A)$ we see that the only terms that are non-negative in the sum belong to a perfect matchings of which half of them must have positive and the other half negative sign. □

Note that the alternative proof of Corollary 3.11 does not construct a different signed perfect matching; it only shows that it exists. To my knowledge, by using the one-to-one correspondence between perfect matchings in Euler graphs and completely labeled Gale strings where the label string is derived from an Euler tour of the graph (see Section 3.3) we give the first constructive proof (apart from exhaustive search) to find a different signed perfect matching. An algorithm which computes another signed perfect matching is given in Section 4.2.

The proof of Corollary 3.11 is similar to a special version of the following well known result from Lovász and Plummer (1986). We include the result and the proof for the sake of completeness. An undirected graph $G = (V,E)$ has an even number of perfect matchings if and only if there is a set $S \subseteq V$, $S \neq \emptyset$, such that every vertex in V is adjacent to an even number of points in S. The result is shown by giving the edges a suitable orientation and by multiplying the adjacency matrix of G with a $0-1$ vector which has 1's corresponding to the vertices from S and zeros otherwise. This shows that the sets of desired property are in the kernel of the adjacency matrix, similar to the proof of Corollary 3.11. If G is an Euler graph then S is equal to the entire vertex set.

Chapter 4

Another Signed Perfect Matching

This chapter concentrates on the function problem of ANOTHER SIGNED PERFECT MATCHING, introduced in Section 4.1. Here, we also describe Edmonds's (1965) matching algorithm and characterise pairs of signed matchings. In Section 4.2 we adapt the LH-algorithm to the setting of directed Euler graphs derived from Gale strings, referred to as the pivoting algorithm. We show that if the Euler graph is additionally bipartite then the pivoting algorithm displays linear running time in the size of the graph; see Section 4.3. For general Euler graphs, determining the complexity of ANOTHER SIGNED PERFECT MATCHING is still an open question.

The pivoting algorithm generalises the LHG-algorithm in the sense that it allows us to specify a set of rules, a so-called edge pairing, which determines the next pivot. In case the edge pairing is derived from the Euler tour in the Morris graphs, the pivoting algorithm takes exponentially many steps; see Section 4.4.

4.1 Matching theory and another signed perfect matching

In his seminal paper, "Paths, Trees, and Flowers", Edmonds (1965) not only introduced the first polynomial-time algorithm to find a maximum matching in a non-bipartite graph but also coined the concept of efficient combinatorial algorithms. Since then the field of matching theory and algorithms has received wide attention in the literature and a number of algorithms improving the running time of Edmonds's algorithm have been given,

see Lovász and Plummer (1986). As already mentioned in Section 3.3, the well known augmenting paths algorithm for bipartite graphs constructs non-simple augmenting paths, or walks, in general graphs. Edmonds solved this problem by contracting non-simple augmenting paths; we give a brief description in the following.

Start with $M = \emptyset$, as in the bipartite case, and construct an augmenting path p with respect to M. One method is to start at a free vertex using breadth-first search, thus building a search tree. An odd cycle stemming from an edge in M is called a *blossom*; see Figure 4.1. The odd cycle and the path, the *stem*, that leads up to the cycle, form a *flower*. Note that an even cycle cannot be encountered because that would mean revisiting

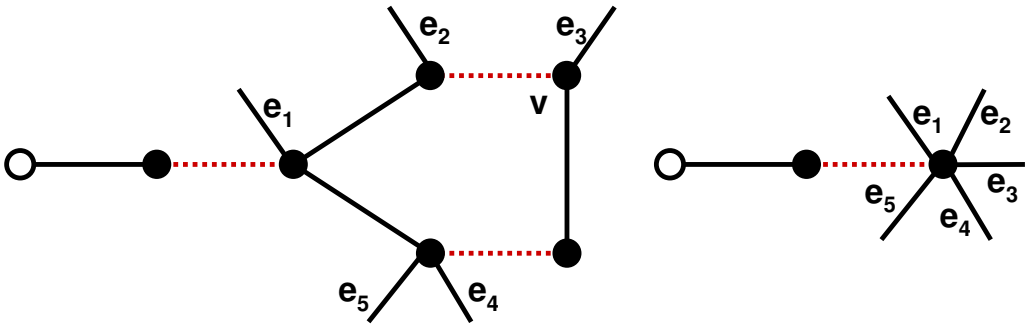


Figure 4.1 The left graph starts from the free vertex followed by a black (unmatched) edge not in M; red dashed (matched) edges are in M. The stem leads to the blossom which is then contracted into a super vertex; see the right graph.

a vertex in the breadth-first search. A blossom is shrunk into a super vertex, keeping the (unmatched) edges in $V \setminus M$ as adjacent edges. After this the breadth-first search is continued, shrinking further blossoms as they are encountered.

Having found an augmenting path p that traverses through a super vertex, we can find an augmenting path through the original blossom, after which the search tree can be discarded. The argument is simple. Suppose the augmenting path leaves the blossom at some vertex v, then because the blossom is an odd cycle, there exists an augmenting path either clockwise or counterclockwise leaving the blossom at v; see Figure 4.1. Finally, set M to $M \oplus p$ and continue the process above until no augmenting path can be found with respect to M, upon which the algorithm terminates with a maximum matching.

The complexity of Edmonds's algorithm is $O(|V|^4)$; it can, however, be decreased by recycling the old search tree to construct a new one, resulting in a $O(|V|^3)$ running time.

Further increases in running times can be accomplished using special data structures or using approximation algorithms; see Galil (1986) for a survey. Note, that we can use Edmonds's algorithm to efficiently check whether an edge $e = (u,v)$ is in any perfect matching of a graph $G = (V,E)$. Consider the auxiliary graph G' with vertex set $V' = V \setminus \{u,v\}$ and check if it admits a perfect matching. If so, e is in a perfect matching, else it is not.

When we consider an Euler graph G with d vertices we label the vertices 1 through d; we sometimes relabel them to simplify arguments. Finding another perfect matching for Euler graphs that admits one perfect matching is polynomial-time solvable. From here on, unless otherwise stated, let this Euler graph be directed such that its orientation guarantees that each vertex is balanced. In other words we require that its orientation is balanced. As we usually consider an Euler graph obtained from a label string, the orientation of the Euler graph is given by the label string l, which is then an Euler tour of the graph (see Section 3.3).

The question we are concerned with is: how hard is it to find another perfect matching of different sign with respect to the given orientation (see Section 3.6 for the definition of the sign of a perfect matching)? This total function problem is stated as follows:

ANOTHER SIGNED PERFECT MATCHING

Input: Directed Euler graph $G = (V,E)$ such that its orientation is balanced and a perfect matching M.

Output: A perfect matching M' s.t. $sign(M) \neq sign(M')$.

We could rephrase the problem to the following: find a perfect matching of a certain sign. However, since finding one perfect matching is easy, we use the initially found M as a starting point, in case it is of opposite sign to find another one. As a convention, let M have positive sign and M' opposite sign.

We now explain why ANOTHER SIGNED PERFECT MATCHING is a total problem by translating it to the Gale string setup. Given that the orientation of the directed Euler graph is balanced, an Euler tour exists and is easily found. Let this Euler tour be a labeling in the Gale string setup. Since M is a perfect matching of G, by Theorem 3.4 there exists a completely labeled Gale string s for the labeling. The sign of the completely labeled Gale string s and its corresponding perfect matching M have the same sign. This follows from

the construction of the directed Euler graph from a labeling as explained in Section 3.3 (this construction goes both ways). By Theorem 3.8 another completely labeled Gale string s' of different sign exists. Again using Theorem 3.4 there exists a perfect matching M' corresponding to s'. The other perfect matching M' has a different sign from M. The above function problem is thus a total problem.

Note that for general directed graphs the function problem ANOTHER SIGNED PERFECT MATCHING is not total and in fact FNP-complete (see Section 6.1). As a reminder, the problem ANOTHER SIGNED PERFECT MATCHING is equivalent to 2-Nash for Gale games.

We consider an alternative condition to capture the sign of two perfect matchings. The following observation is well known. The symmetric difference $M \oplus M'$ of two perfect matchings M and M' is a set of even length cycles. The edges of such a cycle belong alternatively to M and M'. If an edge e is both an element of M and M' then it is not in $M \oplus M'$ and since no two edges e and f are adjacent in M or M' the cycle must be alternating.

In the following, C is a cycle in $M \oplus M'$, which is of even length. For the subgraph restricted to the vertices in C, both $C \cap M$ and $C \cap M'$ are perfect matchings, which have a well-defined sign for the subgraph. If these signs are opposite, then one obtains from M a differently signed perfect matching for the whole graph by replacing the edges of M in $C \cap M$ by those in $C \cap M'$. Then C is called a *sign-switching* cycle for M, and it suffices to find such a sign-switching cycle to solve the problem ANOTHER SIGNED PERFECT MATCHING.

Whether C is sign-switching or not depends on the orientation of the edges in C. Suppose that all edges point in the same forward direction around the cycle. For example, if C has six vertices numbered $1,\ldots,6$, its six directed edges are $(1,2)$, $(2,3)$, $(3,4)$, $(4,5)$, $(5,6)$, and $(6,1)$. Then one matching $C \cap M$ has edges $(1,2)$, $(3,4)$, $(5,6)$, where the corresponding identity permutation has no inversions and positive sign. The other matching $C \cap M'$ has edges $(2,3)$, $(4,5)$, $(6,1)$, where the corresponding permutation has five inversions and therefore negative sign. So C is a sign-switching cycle. By the same argument, this holds for any cycle of even length k with all edges pointing forward.

We would like to characterise sign-switching cycles by pairs of edges. Fix one vertex v of C and partition the cycle into pairs of successive edges such that the edge containing v is at the beginning of one such pair. For the example above, if we fix one vertex, say vertex 1, the partition is given by three pairs of edges $((1,2),(2,3))$, $((3,4),(4,5))$, and $((5,6),(6,1))$. We call a pair of edges equally oriented or oppositely oriented: A pair of adjacent edges in C is called *equally oriented* if both edges have the same orientation, and *oppositely oriented* if the edges have different orientation. For the example above, the pair $((1,2),(2,3))$ is a pair of equally oriented edges. Sign switching cycles are characterised as follows.

Lemma 4.1 *The following statements are equivalent for an even length cycle C:*

1. *It is sign switching.*
2. *It has an even number of oppositely oriented pairs of edges.*
3. *It has an even number of edges pointing clockwise.*

Proof: Let C be a sign-switching cycle and consider edges in pairs, in each pair the first edge from M, the second from M'. As explained above, if all edges are pointing forward (clockwise) then C is a sign-switching cycle. For each pair of edges, make the first edge (from M) point forward, if needed by switching directions of both edges in a pair. This switch does not change relative signs of $C \cap M$ and $C \cap M'$. All "equally oriented" pairs now point forward and all "oppositely oriented" pairs have the second edge pointing backwards (counterclockwise).

Changing two pairs of oppositely oriented pairs of edges such that now all edges in the two pairs point forward does not change the sign of $C \cap M'$. If there is an odd number of oppositely oriented pairs of edges then this gives one inversion less, so C is not sign switching. Hence, a cycle is a sign switching cycle if and only if there is an even number of oppositely oriented pairs of edges, showing the equivalence between 1 and 2.

If C has an even number of oppositely oriented pairs of edges then each such pair has one edge pointing clockwise. Equally oriented pairs of edges either have two or zero edges pointing clockwise. This shows the equivalence between 2 and 3. □

Lemma 4.1 lets us easily decide whether a cycle is a sign switching cycle and whether two given perfect matchings are of opposite sign. Two matchings are of opposite sign if

the symmetric difference has an odd number of sign-switching cycles. Given a perfect matching it is enough to find one sign-switching cycle to find another perfect matching of different sign. However, finding a sign-switching cycle in general is not as easy as finding any perfect matching. In the special case where the graph has a parallel unmatched edge that has opposite direction to a matched edge we have found a sign-switching cycle with two edges, or one pair of equally oriented edges. Not all graphs admit such an edge, for example consider graphs without multiple edges.

The naive approach to solve ANOTHER SIGNED PERFECT MATCHING is to successively find all perfect matchings until a differently signed one is found. This can take exponentially long because the number of perfect matchings of a graph can be exponential. It raises the question how to find a perfect matching of a certain sign without using a brute force approach.

4.2 The pivoting algorithm for Euler graphs

We want to solve the problem ANOTHER SIGNED PERFECT MATCHING. Below, we adapt the LHG-algorithm to directed Euler graphs with a balanced orientation to find a signed perfect matching. To my knowledge this pivoting algorithm is the only algorithm which solves ANOTHER SIGNED PERFECT MATCHING without considering all perfect matchings.

For a directed Euler graph $G = (V, E)$ with a balanced orientation and an Euler tour Et following the orientation of G let M be a perfect matching. Note that for a directed Euler graph with a balanced orientation an Euler tour can can be easily found; the important condition is that the orientation is balanced. In the last section we explained that by using the results from Section 3.4, that if G has a perfect matching M the LHG-algorithm for the equivalent labeling given by the Euler tour finds another perfect matching, M', such that $sign(M) = -sign(M')$.

We would like to adapt the LHG-algorithm to the setting of directed Euler graphs. As a reminder, when we refer to an Euler graph we mean that it is directed and has a balanced orientation.

In an Euler graph two edges, e_1 and e_2, are *paired* if they both have the same orientation and if they are both incident to one vertex v. Thus, if the head of, say, e_1 is incident to v, then the tail of e_2 has to be incident to v. An *edge pairing* of a directed Euler graph is a set of paired edges, such that each edge $e \in E$ is exactly paired twice, once where its head is incident to a vertex v and once where the tail is incident to a different vertex w (recall that the graph is loop free).

For the Euler graph given by $Et = 12345643$ in Figure 3.3 a different pairing to the Euler tour is if we pair $(3,4)$ with $(4,3)$ and $(4,3)$ with $(3,4)$. Once these two pairings are part of the edge pairing we cannot pair $(3,4)$ with $(4,5)$ anymore. Algorithm 3 outputs another perfect matching of different sign given a pairing of edges.

Algorithm 3: Pivoting algorithm for perfect matchings

input : A directed Euler graph $G = (V,E)$, a perfect matching M, and an edge pairing of G.

output: Perfect matching M' s.t. $sign(M) = -sign(M')$.

1 $S_1 = M \setminus (w,a_0) \cup (a_0,y_1)$ s.t. (w,a_0) and (a_0,y_1) are paired;
2 **while** *(*w not covered by S_t for $t \geq 1$*)* **do**
3 **if** *(*$(y_t,a_t) \in S_t$ *and* $a_t \neq a_{t-1}$*)* **then**
4 $S_{t+1} = S_t \setminus (y_t,a_t) \cup (a_t,b)$ s.t. (y_t,a_t) and (a_t,b) are paired;
5 **else**
6 **if** *(*$(a_t,y_t) \in S_t$ *and* $a_t \neq a_{t-1}$*)* **then**
7 $S_{t+1} = S_t \setminus (a_t,y_t) \cup (b,a_t)$ s.t. (b,a_t) and (a_t,y_t) are paired;
8
9 $y_{t+1} = b$;
10 t++;
11 **return** $M' = S_t$

In this context, a w-skew matching after $t \geq 1$ pivots in the pivoting algorithm is a set of edges $S_t \subseteq E$ where every vertex $v \in V$, apart from two special vertices w and y_t, is adjacent to exactly one edge from S_t. The special vertex w is not covered by S_t while y_t is incident to two edges. The duplicate vertex y_t can be either at the head or at the tail of an edge which makes the case distinction in Line 3 and Line 6 of the pivoting algorithm necessary. We often just refer to a w-skew matching as a skew matching. The start of the

pivoting algorithm is to set $S_0 = M$ and drop any w to obtain S_1. We sometimes say that

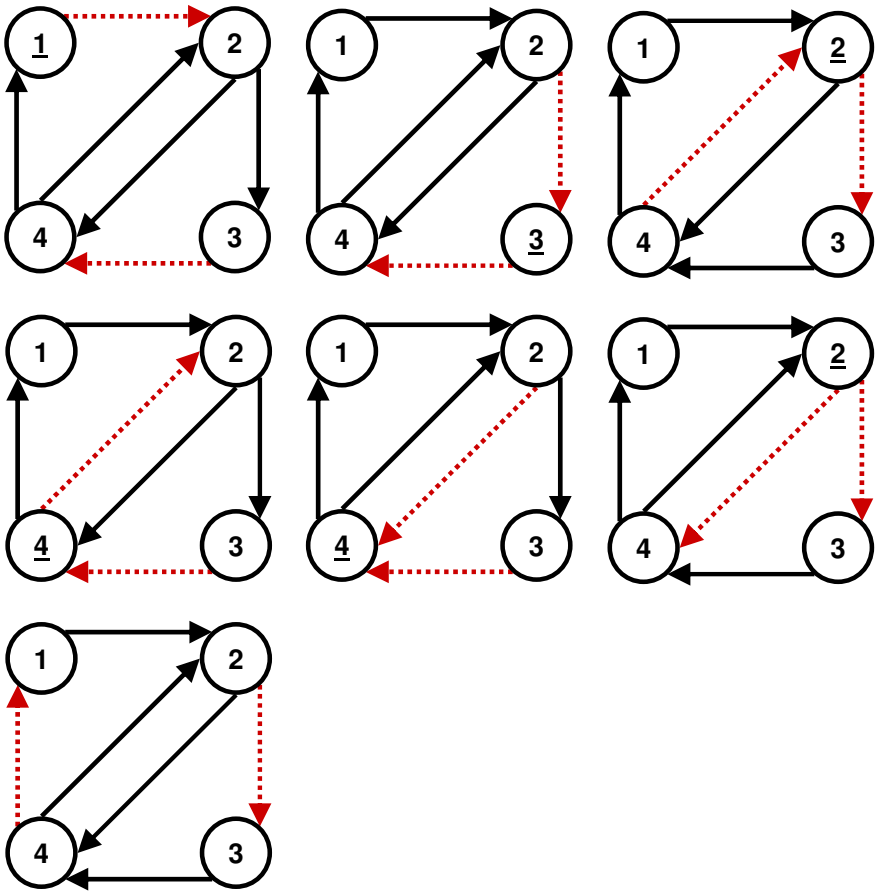


Figure 4.2 The pivot path for the graph derived from the Euler tour 123424. The pairing of edges is given by the Euler tour. The initial perfect matching, given by the edges $(1,2),(3,4)$ which are red and dashed, has positive sign. Start by dropping vertex 1, which is underlined, from the top-left graph. This picks up the edge $(2,3)$ in the top-middle graph. Here, the red, dashed edges correspond to S_2 (in general to S_t after t pivots). From S_2 drop vertex 3 to pick up edge $(3,4)$ and continue by replacing each edge incident to the duplicate (underlined) vertex with its paired edge. Note that the second-to-last picture results from replacing the edge $(3,4)$ with its paired edge $(2,3)$, where these two paired edges point backwards. This corresponds to the case where the duplicate label, here 4, is at the head of an edge. The final step replaces $(2,4)$ with $(4,1)$ and results in another perfect matching $(2,3),(4,1)$, which has negative sign.

we "pick" up an edge which means that the edge is included in the skew matching and is synonymous with pivoting a vertex which is adjacent to one end of the edge. Figure 4.2

gives an example of a pivot path for the Euler graph corresponding to the Euler tour 123424.

First we show the correctness of the algorithm. Although the pairing might not define an Euler tour anymore the algorithm terminates with a perfect matching as before. Line 3 and Line 6 require that $a_t \neq a_{t-1}$ when dropping an edge. This implies that at each iteration there is a unique next step which can not undo the previous pivot. Thus cycling is ruled out (the argument is analogous to the one given for the correctness of Algorithm 1).

We next show $sign(M) = -sign(M')$. The proof is similar to the one given for Gale strings but differs because some extra steps taken in the pivoting algorithms are not taken in the LHG-algorithm, as we argue below. Suppose M has positive sign and is given by $S_0 = M = \{(i_1, j_1), (i_2, j_2), \ldots, (i_n, j_n)\}$ such that each edge $(i_k, j_k) \in M$ is directed from i_k to j_k. Drop an initial vertex w, say i_1. This means the edge (i_1, j_1) is replaced by the edge (j_1, y_1) that it is paired with, which has the same orientation. If $y_1 = i_1$, then the algorithm terminates, with a different signed matching because the original edge (i_1, j_1) is replaced by (j_1, i_1). Otherwise, y_1 is now a duplicate vertex.

After the first pivot the skew matching S_1 is of the form $S_1 = \{(j_1, y_1), (i_2, j_2), \ldots, (i_n, j_n)\}$. When we substitute the missing label w for the duplicate label that was just picked up, the set $\{(j_1, w), (i_2, j_2), \ldots, (i_n, j_n)\}$ has negative sign and the permutation where w is substituted for the duplicate label already present must have positive sign as we simply switch two elements. From S_1 (and S_t in general if the algorithm has not yet terminated) we next drop the duplicate vertex that was already present prior to the last pivot. So the next skew matching S_2 (and S_t in general) has a negative sign when we substitute the missing label w for the duplicate label that was just picked up. This is because when we pivot the duplicate label y_t, assume y_t is at the tail of an edge, the only difference between S_t and S_{t+1} is that the edge (y_t, a_t) is exchanged with the edge (a_t, b), where $y_{t+1} = b$ is now the duplicate label S_{t+1}. The case where y_t is at the head of an edge, corresponding to Line 6 in the pivoting algorithm, is proven analogously. Substituting w for the dropped duplicate vertex in S_t and the newly picked up duplicate vertex in S_{t+1} we get (w, a_t) in S_t and (a_t, w) is S_{t+1}. This explains the sign change between S_t and S_{t+1} after the substitution as we simply switch two elements. Clearly, if the missing vertex w is picked up we do not have to carry out the substitution to determine the sign

because this pivot results in another signed matching M'. By the above argument (a_t, w) is included in $S_{t+1} = M'$, which then must be of negative sign. This completes the proof.

Note that the pivoting algorithm needs to know which duplicate vertex to pivot next at each skew matching S_t for $t \geq 1$, this is because the duplicate vertex y_t is part of two edges. For this we need to keep track of a_{t-1} in order to drop the correct edge given by the duplicate vertex y_t and another vertex a_t which is not equal to a_{t-1}. After a pivot we set y_t to $y_{t+1} = b$, the new duplicate vertex. However, we could easily do without keeping track of a_{t-1} and y_t by exploiting the sign of a skew-matching (analogously to Algorithm 2). Suppose you are at some S_t for $t \geq 1$ and you would like to know which edge to drop next. The duplicate vertex y_t in S_t is easily computed. Next, substitute the missing vertex w for a duplicate vertex y_t and determine the sign of the permutation. If this sign is the same as the sign of M, drop the edge including this duplicate vertex else drop the edge including the other duplicate vertex. If we substitute the duplicate vertex that we pivot next (this vertex is part of the edge given by y_t and a_t) with the missing vertex, then this permutation always has the same sign as M, as explained in the previous paragraph. This argument shows that ANOTHER SIGNED PERFECT MATCHING is in PPADS.

If the pairing of the edges in Algorithm 3 is consistent with the Euler tour corresponding to the Gale string the pivoting algorithm outputs the same matching as the Gale string version of the LHG-algorithm as we explain next. When dropping a label in the LHG-algorithm the one corresponding to the dropped label "jumps" over an odd run of 1's in one step while the pivoting algorithm takes each step separately until the end of the run is reached. That is, if a 1 pivots over $k+1$ adjacent $1's$ of a run, where k is odd, then the pivoting algorithm needs $\lfloor k/2 \rfloor$ pivots to reach the same configuration as the LHG-algorithm. This explains the equivalent outcome of the two algorithms. This argument can be seen by comparing the steps taken in Figures 4.5 and 4.6.

Algorithm 3 is more general than the LHG-algorithm as it lets us specify an edge pairing, possibly different from the one given by the Euler tour of the graph, in advance of running the algorithm. However, two different pairings can result in different outcomes of the pivoting algorithm. If, instead of defining an edge pairing in advance, we let the algorithm pick a pairing of the edges at each pivot step, such that once two edges are paired they remain paired throughout the algorithm, then this resembles Edmonds's (2007) exchange algorithm; see Section 3.5. However, the edge pairing of Algorithm 3 is more

specific than the one from the exchange algorithm as it requires that only pairs of equally oriented edges can be paired. This assures that the other perfect matching is of different sign. A skew matching, S_t, then refers to Edmonds's w-skew room partition and a perfect matching to a room partition.

We will show in the next section that the pivoting algorithm terminates in linear time when the given graph is bipartite. In order to see this we analyse the symmetric difference between M and S_t after t pivots. Let the auxiliary graph be given by $D_t = M \oplus S_t$. The auxiliary graph D_t is comprised of not necessarily sign-switching cycles and one path of even length. This path can, however, have a vertex in common with a cycle; clearly this vertex must be the duplicate one. In that case, the next pivoting step is to either drop an edge from the cycle, resulting in one path, or to drop an edge from the path, resulting in a path and a separate cycle. Starting from the beginning of the path in D_t, that is from an edge in M, a pair of consecutive edges in D_t is called "equally oriented" if both edges in E have the same direction and "oppositely oriented" otherwise (as defined in Section 4.1). Upon termination D_t is a set of cycles. An odd number of them must be sign switching. In Figure 4.3 we give an example which depicts the pivot path of a bipartite graph as the symmetric difference.

Lemma 4.2 *Neglect cycles that might be in D_t and assume that the path in D_t has the form $p = (w,u_1),(u_1,u_2)\ldots,(u_t,y_t)$, where $w \in M$ and w and y_t are the missing and duplicate labels, respectively and such that the pair of edges of p of the form $(\cdot,u_i)(u_i,u_{i+1})$ for i odd are paired. Let the index i of a vertex denote the distance in the number of edges from w. If the next duplicate label y_{t+1} is u_i where i is even or y_t then $(u_i,u_{i+1}),\ldots,(y_t,b)$, (b,y_{t+1}) is a sign-switching cycle.*

Proof: Suppose we "hit" the initial path again at an even length i, giving the next duplicate label u_i. The cycle $(u_i,u_{i+1}),\ldots,(y_t,b),(b,y_{t+1})$ is then created by dropping the black edge (u_{i-1},u_i). Suppose the cycle is not sign switching. As we could have started the algorithm by initially dropping u_i, which would have resulted in the same pivot sequence as D_t starting from u_i. This holds because the pair of edges of p of the form $(\cdot,u_i)(u_i,u_{i+1})$ for i odd are paired. Then u_i would be picked up at step $t+1-i$ terminating the algorithm with a different signed perfect matching, a contradiction. □

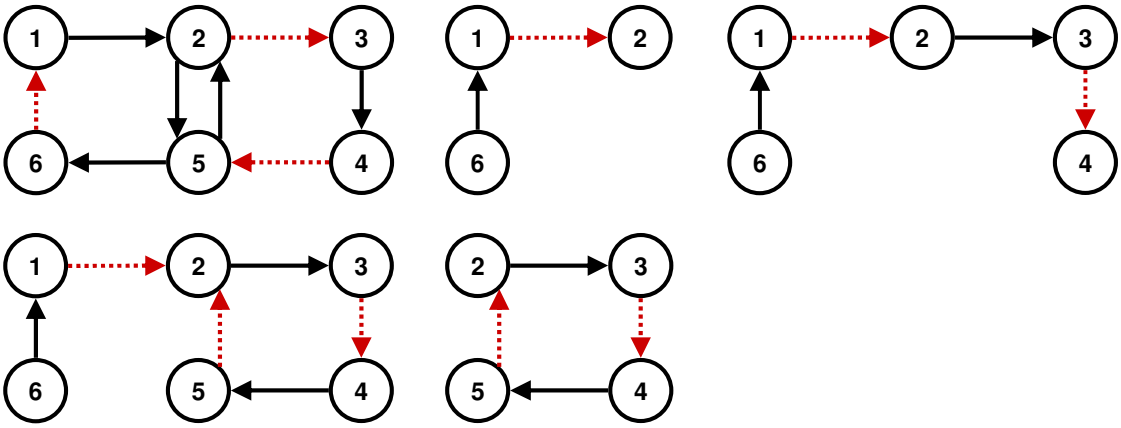


Figure 4.3 The pivot path for the Euler graph given by the Euler tour 12345256. The pivot steps are given by the auxiliary graph D_t after t pivots. The top left graph gives the original graph and a perfect matching. Start by dropping vertex 6. After the first pivot the path D_1 (top middle graph) consists of one pair of equally oriented edges. After 3 pivots the newly added edges "hit" the original path, forming the sign-switching cycle given by edges $(2,3)(3,4)(4,5)(5,2)$. The next step is to drop the duplicate vertex 2 which unravels the path leading to the cycle. This results in the sign-switching cycle $D_4 = M \oplus M'$.

Lemma 4.2 shows that D_t is time independent in the sense that the starting point of algorithm is irrelevant. Flipping the black edges in S_t to cover the red edges in D_t creates a new skew matching $\tilde{S}_1$ where the duplicate label from S_t is now the missing label $\tilde{w}$. If $\tilde{w}$ is then picked up at some point then we found a sign-switching cycle as we could have started the algorithm from $\tilde{S}_1$ in the first place. Hence after u_i is picked up the path must unravel back to w. If the pairs of edges in Lemma 4.2 are not paired, Lemma 4.2 is not true anymore. This is because a cycle that is found by picking up vertex u_i can be non-sign switching.

4.3 Bipartite Euler graphs

The question we would like to answer is: How fast is the pivoting algorithm when applied to bipartite graphs? In this section assume that $G = (X \cup Y, E)$ is a bipartite Euler graph with the bipartition given by the sets X and Y. Lemma 4.3 shows that another signed

matching is found in linear time by the pivoting algorithm; see Figure 4.3 for an example.

Lemma 4.3 *Given a bipartite directed Euler graph $G = (X \cup Y, E)$ and a perfect matching M, a different perfect matching M' is found in $O(|X|)$ time.*

Proof: We need the well known fact that a bipartite graph does not admit odd length cycles. Let the edge pairing be given by the Euler tour and assume that M has positive sign; the proof is identical for a general pairing. Drop any initial vertex $w \in X$. If w is picked up in the first step we are done. Otherwise, D_t is a path of pairs of equally oriented edges after t pivots, i.e. pivoting has not picked up a vertex already present in D_t. In other words we have not hit the path again.

If D_t picks up w without hitting the path then the algorithm terminates after at most $|X|$ steps. In case D_t hits the path for some t then the new duplicate vertex b is part of the path. D_t now consists of a path only consisting of pairs of equally oriented edges from w to b and an even length cycle going through b but not intersecting the path from w to b otherwise. The next step of the algorithm is to drop the edge at the end of the path from w to b, thus unraveling the path step by step until w is picked up and the algorithm terminates after at most $2|X|$ steps. □

A game-theoretic significance or economic interpretation of Gale games derived from bipartite Euler graphs is not known to me. It would, however, be interesting to investigate the difference between games derived from bipartite and general Euler graphs and possibly classify them by their economics meaning.

4.4 Morris's construction and long pivot paths

The pivoting algorithm computes ANOTHER SIGNED PERFECT MATCHING in linear time when the input graph is bipartite. How fast is this algorithm in general? We show that when Morris's construction of dual cyclic polytopes (Morris (1994)) is translated to Euler graphs via Gale strings, the pivoting algorithm generates exponentially long pivot paths. Extending the construction by Morris, Savani (2006) defined "triple imitation" games that are hard-to solve bimatrix games (and also Gale games); see Savani (2006). We adapt the

construction by Morris, with a small simplification that omits repeated labels so that the Euler graph does not have loops.

Morris (1994) constructs d-dimensional dual cyclic polytopes T_d with $2d$ facets that have exponentially long pivoting paths, originally referred to as "Lemke paths" by Morris. While Morris considers both even and odd d we restrict ourselves to the case when d is even. In Section 3.2 we explained that these polytopes can be captured by the set of Gale evenness strings $G(d,2d)$. Morris gives the $1,\ldots,2d$ facets the labels as follows:

$$l(k)=k, \quad 1\le k\le d, \qquad l(d+k)=\begin{cases} d, & k=1, \\ d-k, & k \text{ even and } 2\le k<d, \\ d+2-k, & k \text{ odd and } 2\le k\le d, \\ 1, & k \text{ even and } k=d. \end{cases} \tag{4.1}$$

For $d=8$, for example, the 16 facets corresponding to Gale strings in $G(8,16)$ are labeled with $1,2,3,4,5,6,7,8,8,6,7,4,5,2,3,1$. The only two completely labeled vertices of T_d are 1^d0^d and 0^d1^d. We give the proof for this in the setting of Euler graphs and perfect matchings. As a convention, let the completely labeled Gale string s_0 for Morris's example be such that the first d bits are set to one and the remaining bits are zero; s_0 corresponds to the perfect matching M.

To give an expression for the length of the pivot path, i.e. the number of pivots, we use the following standard notation for functions $f,g:\mathbb{N}\to\mathbb{R}$. A function $g(n)$ is a member of $\Theta(f(n))$ if there exist positive constants c_1 and c_2 and $n_0\in\mathbb{N}$, such that $c_1f(n)\le g(n)\le c_2f(n)$ for all $n\ge n_0$. Intuitively, this means that $g(n)$ has up to a linear factor the same growth rate as $f(n)$ after a certain threshold n_0.

Proposition 4.4 *(Morris (1994)) The shortest of the d pivoting paths on T_d starting at s_0 is obtained by initially dropping label $d/2$. The length of this path is $\Theta((1+\sqrt{2})^{d/4})$. The longest path is obtained by initially dropping label d, which is of length $\Theta((1+\sqrt{2})^{d/2})$.*

We described in Section 3.3 how to derive the Euler graph from the labeling function. For an Euler tour Et, define $Et:[n]\to[d]$ as the function that maps the i-th position to its vertex and let $Et^{-1}(i)$ give the position of a vertex i. Although vertices and their positions are not necessarily in a one-to-one relationship one can think of a vertex as being associated with a number that gives it a position in the Euler tour. For example for

$Et = 123423$ vertex 2 is associated with position 2 and 5. When using Et^{-1} the ambiguity will be removed. The Euler tour specific to Morris's example is given by Et_d, where

$$Et_d(i) = \begin{cases} i, & i = 1,\ldots,d, \\ 2d-i+1, & i \text{ even and } d < i \leq 2d-2, \\ 2d-i-1, & i \text{ odd and } d < i \leq 2d-2. \end{cases} \tag{4.2}$$

Comparing $Et_d = 1,\ldots,d,(d-2),(d-1),(d-4),(d-3),\ldots,2,3$ with the labeling l from equation (4.1) we see that the difference is the omitting of loops from the labeling l. For the above example the Euler tour is $Et_8 = 12345678674523$. The corresponding Euler graph, referred to as a *Morris graph*, G_d, has a ladder structure where each step is given by two parallel edges directed in the same direction. Figure 4.4 shows G_8 on the left and G_d for general d, where d is even but not divisible by 4 (here dashed edges are not matched edges - they indicate that some vertices are omitted). When d is a multiple of 4, the only difference is that the bottom steps are directed from left to right, as seen in G_8.

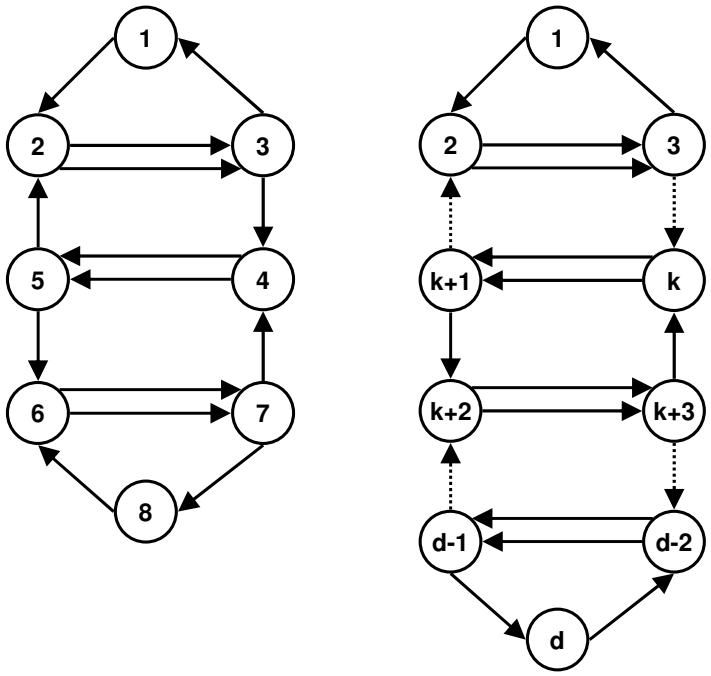


Figure 4.4 G_8 and G_d, for d indivisible by 4.

Proposition 4.5 *The only two perfect matchings of G_d are $M = \{(1,2),(3,4),\ldots,(d-1,d)\}$ and $M' = \{(3,1),(5,2),\ldots,(d,d-2)\}$.*

Proof: Clearly M and M' are perfect matchings of G_d. None of the parallel edges that form the "steps" of the ladder can be part of a perfect matching because removing them decomposes the graph into two sets where each has an odd number of vertices and therefore is not a perfect matching. The only remaining edges are those of the "outer" cycle whose edges are $M \cup M'$. So M and M' are the only perfect matchings. □

Denote by $\pi(i,d)$ the *pivot path*[1] of G_d with missing label i where the pairing is naturally given by the Euler tour Et_d, as described in Section 4.2. The path is comprised of a sequence $M_0, M_1, \ldots, M_L$ of i-skew matchings, where the starting point is M as a convention. Define $\pi(i,d)^{-1}$ to be the inverse of $\pi(i,d)$, which is the path traversed in the opposite direction, starting from M'. That is, if $M_0 = M$ then the sequence when starting from M' and dropping label i is given by $\pi(i,d)^{-1}$. The *length* of the path $\pi(i,d)$, i.e. the number of pivots taken to reach another perfect matching, is given by $L(i,d)$ when starting from from M. Figure 4.5 gives the pivoting steps for the Gale representation of G_6 when dropping label 1, which takes 10 pivots. In comparison, Figure 4.6 gives the complete pivot path $\pi(1,6)$ for G_6 when dropping label 1 with length $L(1,6) = 17$.

step	**1**	**2**	**3**	**4**	**5**	**6**	**4**	**5**	**2**	**3**
1	$\underline{1}$	1	1	1	1	1	.	.	.	.
2	.	1	1	$\underline{1}$	1	1	$\bar{1}$	.	.	.
3	.	1	1	.	$\underline{1}$	1	1	$\bar{1}$	.	.
4	.	$\underline{1}$	1	.	.	1	1	1	$\bar{1}$	.
5	.	.	1	$\bar{1}$	.	1	$\underline{1}$	1	1	.
6	.	.	1	1	$\bar{1}$	1	.	$\underline{1}$	1	.
7	.	.	$\underline{1}$	1	1	1	.	.	1	$\bar{1}$
8	.	.	.	$\underline{1}$	1	1	$\bar{1}$	.	1	1
9	.	.	.	.	$\underline{1}$	1	1	$\bar{1}$	1	1
10	$\underline{1}$	.	.	.	.	1	1	1	1	1

Figure 4.5 The pivot path $\pi(1,6)$ for the Gale setting.

The difference between the pivoting of Gale strings as in Figure 4.5 and of skew matchings as in Figure 4.6 is that pivoting in a Gale string via a longer run of 1's requires several steps when applied to matchings. For example, the single step from 1 to 2 in Figure 4.5, which "jumps" over a run of six 1's and results in duplicate label 4, corresponds

[1] Sometimes we refer to $\pi(i,d)$ as the pivot path in the Gale string setup, but we indicate when doing so.

to the three pivoting steps from M_0 to M_3 in Figure 4.6. The reason is that every pair of 1's in a Gale string is represented by one edge in the matching.

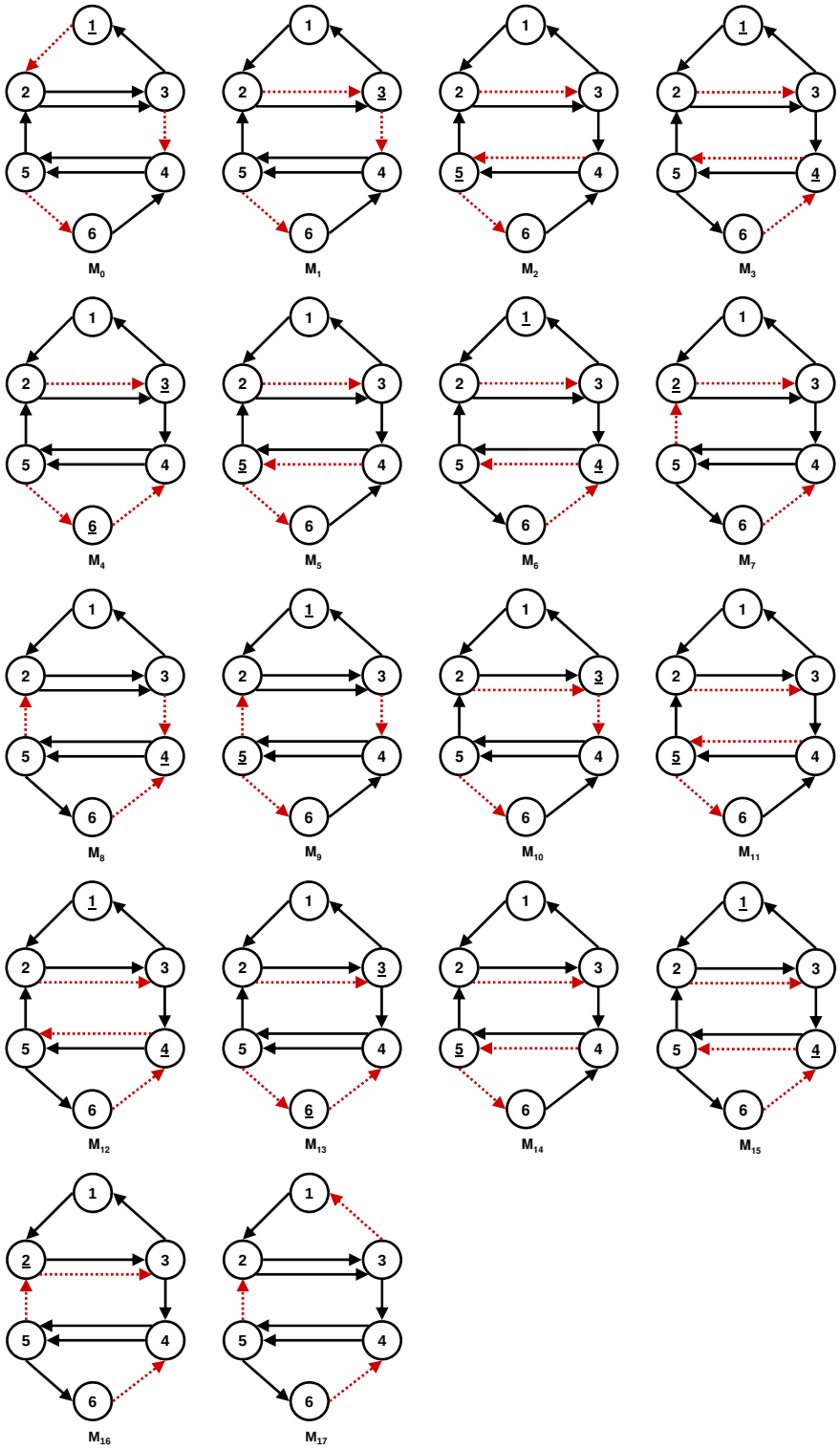


Figure 4.6 The pivot path $\pi(1,6)$.

The following proofs, given for starting from a perfect matching, are similar to Morris (1994) and Savani (2006) but differ from the "Lemke"-paths in its length and by omitting the self loops on vertices 1 and d. The symmetry is also slightly altered, as the proofs will show.

In the following we derive a number of lemmas, similar to Lemmas 3.1, 3.2, and 3.3 of Morris (1994), that use this symmetry. The most important of these, Lemma 4.8 below, is a recurrence that shows that the path obtained when dropping label d is essentially given by a concatenation of these paths for the graphs with $d-2$, $d-4$, and again $d-2$ vertices, which gives rise to their exponential growth. Other lemmas show certain relationships for other dropped labels.

In order to prove Lemma 4.6 we need to introduce the following mapping Φ_d. As can be seen in the example of Figure 4.7, there is a bijection Φ_d on the nodes of the graph that preserves the edge incidences (that is, Φ_d is a graph automorphism) and that maps the matching M to M'. This bijection Φ_d maps also skew matchings on the pivot path to each other. The mapping Φ_d itself is essentially given by the second half of the label string and thus the Euler tour: For example, for $d = 6$, E_6 is given by 1234564523, and Φ_6 maps $1,2,3,4,5,6$ to $1,3,2,5,4,6$.

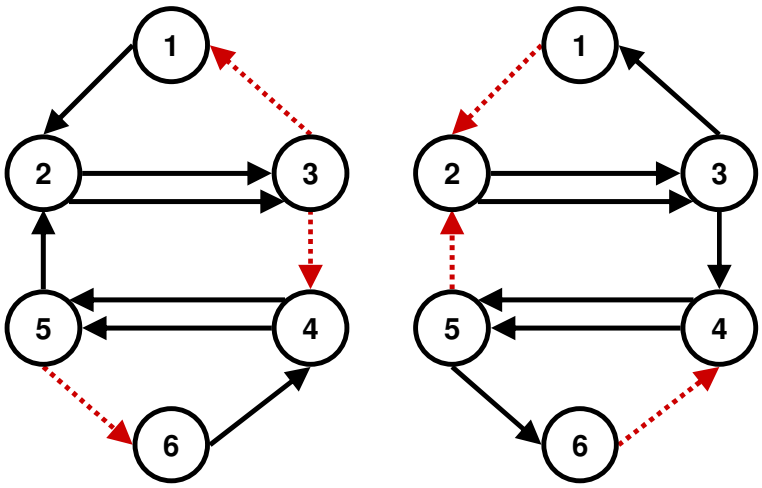


Figure 4.7 The skew matching M_1 with missing vertex 2 for G_6 on the left and $\Phi_6(M_1)$. In both graphs the set of red dashed edges constitutes a skew-matching.

In general, let Φ_d be a mapping between M and M' and any two skew matchings. Define $\Phi_d(\hat{M})$ by mapping or relabeling every vertex v for each time (two computations are needed if i is the duplicate vertex) it is at the end of a matched edge in $\hat{M}$ with the

following function:

$$\phi'(v) = \begin{cases} 1, & \text{for } v = 1, \\ d, & \text{for } v = d, \\ Et_d(2d - Et_d^{-1}(v)), & \text{else.} \end{cases} \tag{4.3}$$

where $Et_d^{-1}(v)$ is now unambiguous because the position of a vertex is defined by the matched edge adjacent to it. In case v is part of a parallel edge it is assumed that the correct position is picked. This could easily be incorporated into the definition of $\Phi'(v)$ by numbering the edges. We feel, however, that this would only add unnecessary notation. Also, we could simplify the graph as described in Section 3.3 by removing parallel edges. Using Φ_d, the pivot path that starts at M and ends at M' is symmetric.

Lemma 4.6 *$L(i,d) = L(i+1,d)$ for $i < d$ even.*

Proof: Let $M_0 = M$ and the path $\pi(i,d)$ by dropping label i at position $Et_d(i) = i$ be given by $(M_0, M_1, \ldots, M_{L(i,d)})$ where $M_{L(i,d)} = M'$ and let $(N_0, N_1, \ldots, N_{L(i+1,d)}) = \pi^{-1}(i+1,d)$ where $N_{L(i+1,d)} = M$. The last pivot to $M_{L(i,d)}$ picks up label i which is at position $2d-i-1$ in the Euler tour. The vertex i at position $2d-i-1$ maps to $\Phi'_d(Et_d^{-1}(i) = 2d-i-1)) = Et_d(i+1) = i+1$, the first vertex dropped from $N_{L(i+1,d)}$. Continuing this argument gives $\Phi(M_k) = N_k$ for $k = 1, \ldots, L(i,d)$ and the result follows. □

Similarly, to prove Lemma 4.7, define another mapping Ψ_d by relabeling the vertices v in a perfect or skew matching via the following function.

$$\psi'(v) = \begin{cases} Et_d(d - Et_d^{-1}(v) + 1), & \text{for } Et_d^{-1}(v) = 1, 2, \ldots, d \\ Et_d(3d - 1 - Et_d^{-1}(v)), & \text{for } Et_d^{-1}(v) = d+1, \ldots, 2d-2 \end{cases} \tag{4.4}$$

The mapping Ψ_d is well defined for a perfect matching or skew matching because it maps the covered vertices about the inflection point of the Morris graph, which is between vertices $d/2$ and $d/2+1$ if d is divisible by 4 and is in the middle of the cycle given by vertices $d/2-1$ to $d/2+2$ otherwise. This keeps the edge structure correct, see Figure 4.8 for an example. If two vertices, say $d-2$ and $d-1$, are connected by a matched edge then these are transformed to vertices 2 and 3 and the edge that is matched is the one between vertices 2 and 3. If the vertex $d-2$ is from the second part of the Euler tour, i.e. $Et_d^{-1}(d-2) > d$, then the vertex 2 also has a position in the Euler tour which is greater than d.

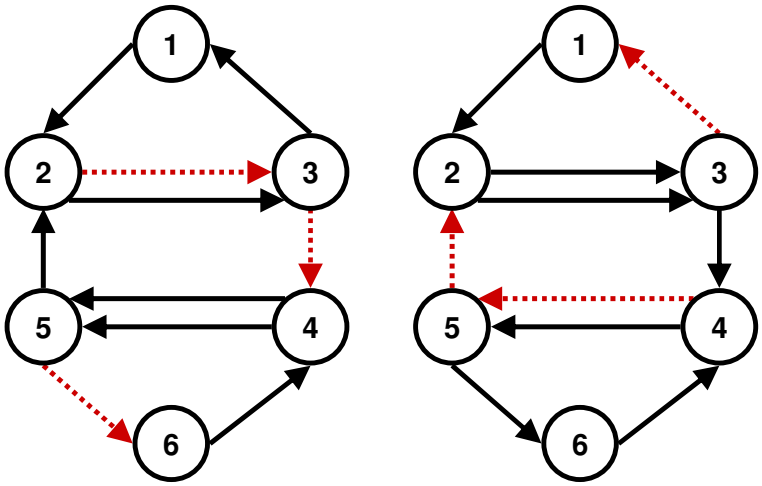


Figure 4.8 On the left the skew matching M_1 with missing vertex 1 for G_6 and on the right $\Psi_6(M_1)$.

Lemma 4.7 *$L(i,d) = L(d-i+1,d)$ for $i \in [d]$.*

Proof: Let $M_0 = M$ and let the path $\pi(i,d)$, given by dropping label i at position $Et_d^{-1}(i) = i$, be given by $(M_0, M_1, \ldots, M_{L(i,d)})$ where $M_{L(i,d)} = M'$ and let $(N_0, N_1, \ldots, N_{L(d-i+1,d)}) = \pi(d-i+1,d)$ with $N_{L(d-i+1,d)} = M'$. By the definition of the mapping $\Psi(M_k)$ it follows that $\Psi(M_k) = N_k$ for $k = 0, 1, \ldots, L(i,d)$. □

The following proofs of Lemmas 4.8 and 4.9 are similar to the proofs given by Morris (1994) and Savani (2006), however the argument becomes simpler when considering the graph setting. The proofs rely on the property that the pivot path of G_d can be concatenated from the pivot paths of Morris graphs of lower dimension. Unlike the polytopes we can draw these graphs to give an intuitive indication why the proof of Lemma 4.8 holds, see Figure 4.9.

Lemma 4.8 *$L(d,d) = 2L(d-2,d-2) + L(d-4,d-4) + 4$ for $d \geq 6$.*

Proof: From $M_0 = M$, drop d to reach $M_1 = \{(1,2), \ldots, (d-3,d-2), (d-2,d-1)\}$, where the new edge is from the first part of the Euler tour. All edges (u,v) are written such that they are directed from u to v. See Figure 4.9 for an intuition of the argument to follow.

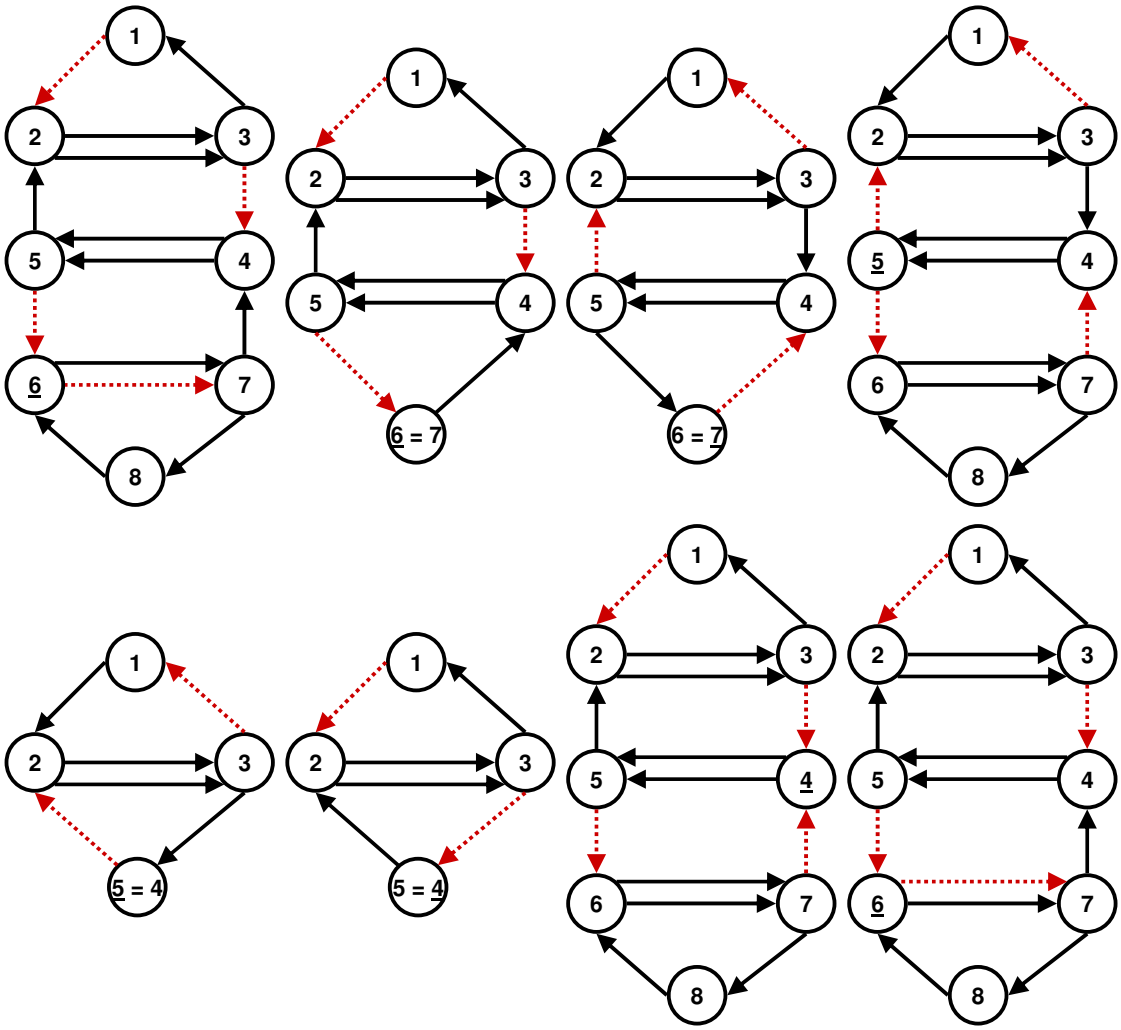


Figure 4.9 The components of the pivot path $\pi(8,8)$. After dropping $d = 8$ the duplicate label is 6 which is underlined, and so far one step has been taken. Note that the "lower" parallel edge $(6,7)$ from S_1 is paired with $(7,8)$, that was just dropped. This starts the pivot path in the "component" G_6 using only vertices 1 to 5 and 7, where one can think of vertices 6 and 7 being one vertex, for this subpath. It takes $L(6,6)$ steps to pick up vertex 7, as part of the edge $(4,7)$, in the component G_6; see the 3rd top graph. The next step is to drop the edge $(6,7)$ after which 5 is duplicate vertex; see the top right graph. From here we now pivot only in $G(4,4)$ (by the same argument as for $G(6,6)$). It takes $L(4,4)$ steps until 4 is picked up; see 2nd bottom graph. Next 4 is dropped from the 3rd bottom graph and the other parallel edge $(6,7)$ joins the skew matching; see the bottom right graph. Note that the parallel edge $(6,7)$ is paired with edge $(7,4)$ and $(8,6)$. Up to now, $L(6,6)+L(4,4)+3$ pivots have been carried out. The next steps are again on G_6 (drop 6 from the bottom right graph) and it takes $L(6,6)$ steps until 7 is picked up. The last step is to drop the edge $(6,7)$ (which is paired with edge $(8,6)$) from the skew matching and terminate by picking up 8 after $L(8,8) = 2L(6,6)+L(4,4)+4 = 23$ steps.

The next pivot drops label $d-2$ from edge $(d-3,d-2)$ and this label will not be again part of a pivot unless the duplicate label is $d-1$ and $(d-2,d-1)$ is dropped from a skew matching. Suppose this is not the case and that $d-2$ becomes the duplicate label by picking up edge $(d-3,d-2)$ instead. The next step would then to pick up the missing label d including the edge $(d,d-1)$, a contradiction as the pivoting algorithm terminates with M. In other words $d-2$ is picked up again only after the edge $(d-1,d-4)$ joins the skew matching. The next steps occur on the Morris graph of "lower" dimension [2] $d-2$ and picking up $d-2$ again corresponds to picking up the edge $(d-1,d-4)$ in G_d, which takes $L(d-2,d-2)$ steps. After $L(d-2,d-2)+1$ steps the skew partition is given by

$$M_{L(d-2,d-2)+1} = \{(3,1),(5,2),\ldots,(d-1,d-4),(d-2,d-1)\}.$$

The next pivot drops the edge $(d-2,d-1)$ and picks up the edge $(d-3,d-2)$ which starts the second path of length $L(d-4,d-4)$ in G_{d-4}. Remember that the matched edges in the upper part of the ladder starting upwards from vertex $d-3$ are from M'.

By identical reason as for label $d-2$, label $d-3$ will only be picked up again if the edge $(d-5,d-4)$ joins the skew matching. Picking up the edge $(d-5,d-4)$ takes exactly $L(d-4,d-4)$ steps which gives

$$M_{L(d-2,d-2)+L(d-4,d-4)+2} = \{(1,2),\ldots,(d-5,d-4),(d-3,d-2),(d-1,d-4)\},$$

where the duplicate label now is $d-4$. The next pivot drops the edge $(d-1,d-4)$ and picks up the edge $(d-2,d-1)$. Note that this edge is paired with the edge $(d,d-2)$ from the second part of the Euler tour.

The last part of the pivot path is identical to the first part of the pivot path on the Morris graph of dimension $d-2$. It takes $L(d-2,d-2)$ steps to pick up the edge $(d-4,d-1)$. The last pivot drops $(d-2,d-1)$ to pick up the missing label d with edge $(d,d-2)$ after $2L(d-2,d-2)+L(d-4,d-4)+4$ steps. □

The proof of the next Lemma 4.9 is similar. After dropping the initial label i the pivot path first "treats" the lower part of the ladder, compared to i, and then the remaining upper part, see Figure 4.10.

Lemma 4.9 *$L(i,d) = L(i,i)+L(d-i,d-i)$ for i even.*

[2] The pivots taken are still on G_d but only use $d-2$ vertices, with labels from 1 to $d-3$ and $d-1$.

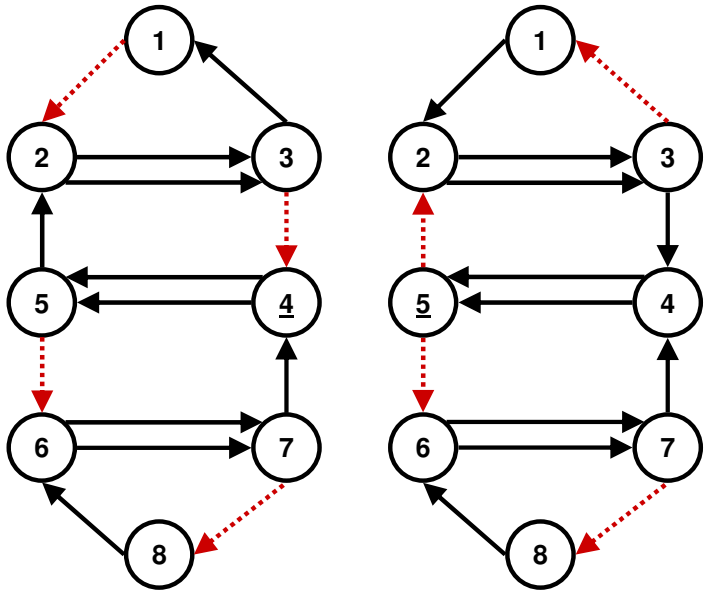


Figure 4.10 The components of the pivot path $\pi(4,8)$. After dropping 4 in the left graph, the pivots are only on the vertices 1 to 3 and 5. Picking up 5 takes $L(4,4)$ steps. After the upper part of the graph is pivoted through, the argument repeats for the lower part of the graph, starting from duplicate vertex 5. This takes $L(4,8) = L(4,4) + L(4,4) = 3+3 = 6$ steps.

Proof: Drop label i starting from M. Due to similar reasoning as in the proof of Lemma 4.8 label i cannot be picked up from the part of the ladder "above" vertex i which results in $L(i,i)-1$ pivots of the part of the Morris graph corresponding only to the first i vertices. Pivot number $L(i,i)$ picks up label $i+1$ on the opposite side of the edge $(i-1,i)$ which was initially dropped. Repeating the above argument the following pivots are on vertices $d-i$ to d. Relabeling them naturally from 1 to $d-i$ and dropping label i results in $L(1,d-i) = L(d-i,d-i)$ pivots until label i in G_d is picked up again. □

Proposition 4.10 *The longest pivot path of G_d starting from M is to drop label 1 or d. The length of this path is $\Theta((1+\sqrt{2})^{d/2})$. The shortest pivot path is obtained by dropping label $d/2$. The length of this path is $\Theta((1+\sqrt{2})^{d/4})$.*

Proof: We need to solve the recurrence

$$L(d,d) = 2L(d-2,d-2) + L(d-4,d-4) + 4 \tag{4.5}$$

from Lemma 4.8 with initial conditions $L(2,2) = 1$ and $L(4,4) = 6$. Due to the similarity to the formulae given in Morris (1994) we use a "guess and check" approach to show

$$L(d,d) = \frac{1}{2}\left[(1+\sqrt{2})^{\frac{d+2}{2}} + (1-\sqrt{2})^{\frac{d+2}{2}}\right] + \frac{1}{2\sqrt{2}}\left[(1+\sqrt{2})^{\frac{d-2}{2}} - (1-\sqrt{2})^{\frac{d-2}{2}}\right] - 2. \tag{4.6}$$

We write (4.6) in this form to illustrate the difference between the number of steps taken by the LHG-algorithm in the Gale setting and the pivoting algorithm in the graph setting. For a Morris graph G_d, the pivoting algorithm takes $\frac{1}{2\sqrt{2}}\left[(1+\sqrt{2})^{\frac{d-2}{2}} - (1-\sqrt{2})^{\frac{d-2}{2}}\right] - 1$ more steps than the LHG-algorithm when initially dropping label d.

Using induction, for $d = 2$ we have $L(2,2) = 1$ and for $d = 4$ we have $L(4,4) = 6$ so the induction start $L(6,6) = 2L(4,4) + L(2,2) + 4 = 12 + 1 + 4 = 17$ is correct. Assume that equation (4.6) holds and substitute it into the recurrence equation (4.5) to verify the induction step.

$$\begin{aligned} &L(d,d) = 2L(d-2,d-2) + L(d-4,d-4) + 4 = \\ &2\left(\frac{1}{2}\left[(1+\sqrt{2})^{\frac{d}{2}} + (1-\sqrt{2})^{\frac{d}{2}}\right] + \frac{1}{2\sqrt{2}}\left[(1+\sqrt{2})^{\frac{d-4}{2}} - (1-\sqrt{2})^{\frac{d-4}{2}}\right] - 2\right) + \\ &\left(\frac{1}{2}\left[(1+\sqrt{2})^{\frac{d-2}{2}} + (1-\sqrt{2})^{\frac{d-2}{2}}\right] + \frac{1}{2\sqrt{2}}\left[(1+\sqrt{2})^{\frac{d-6}{2}} - (1-\sqrt{2})^{\frac{d-6}{2}}\right] - 2\right) + 4 \end{aligned} \tag{4.7}$$

Group $(1+\sqrt{2})^{\frac{d}{2}}$ and $(1+\sqrt{2})^{\frac{d-2}{2}}$ of equation (4.7) as follows:

$$\begin{aligned} &(1+\sqrt{2})^{\frac{d}{2}} + \frac{1}{2}(1+\sqrt{2})^{\frac{d-2}{2}} = \frac{1}{2}\left[(1+\sqrt{2})^{\frac{d-2}{2}}(2+2\sqrt{2}) + (1+\sqrt{2})^{\frac{d-2}{2}}\right] = \\ &\frac{1}{2}\left[(1+\sqrt{2})^{\frac{d-2}{2}}(1+2\sqrt{2}+2)\right] = \frac{1}{2}\left[(1+\sqrt{2})^{\frac{d-2}{2}}(1+\sqrt{2})^2\right] = \frac{1}{2}(1+\sqrt{2})^{\frac{d+2}{2}} \end{aligned} \tag{4.8}$$

Similarly, group $(1-\sqrt{2})^{\frac{d}{2}}$ with $(1-\sqrt{2})^{\frac{d-2}{2}}$, $(1+\sqrt{2})^{\frac{d-4}{2}}$ with $(1+\sqrt{2})^{\frac{d-6}{2}}$, and $(1-\sqrt{2})^{\frac{d-4}{2}}$ with $(1-\sqrt{2})^{\frac{d-6}{2}}$. This completes the induction step, showing the correctness of equation (4.6). To show that $L(d,d)$ is in $\Theta((1+\sqrt{2})^{\frac{d}{2}})$ pick $c_1 = 1/4$ and $c_2 = 4$. Then for all $d \geq d_0 = 4$,

$$L(d,d) \geq \frac{1}{2}(1+\sqrt{2})^{\frac{d}{2}}\left[(1+\sqrt{2}) + \frac{1}{\sqrt{2}}(1+\sqrt{2})^{-1}\right] - 3 \geq c_1(1+\sqrt{2})^{\frac{d}{2}}, \tag{4.9}$$

where the first inequality holds because $|(1-\sqrt{2})| \leq 1$.

$$L(d,d) \leq (1+\sqrt{2})^{\frac{d}{2}}\left[(\frac{1}{2} + \frac{1}{2\sqrt{2}}) + \frac{1}{2\sqrt{2}}(1+\sqrt{2})^{-1}\right] \leq 4(1+\sqrt{2})^{\frac{d}{2}} \leq c_2(1+\sqrt{2})^{\frac{d}{2}} \tag{4.10}$$

This shows that the length of the path $\pi(d,d)$ behaves asymptotically like $(1+\sqrt{2})^{\frac{d}{2}}$.

	dimension									
label	4_G	4_P	6_G	6_P	8_G	8_P	10_G	10_P	12_G	12_P
1	3	6	9	17	23	44	57	109	139	266
2	2	2	4	7	10	18	24	45	58	110
3	2	2	4	7	10	18	24	45	58	110
4	3	6	4	7	6	12	12	23	26	50
5			4	7	6	12	12	23	26	50
6			9	17	10	18	12	23	18	34
7					10	18	12	23	18	34
8					23	44	24	45	26	50
9							24	45	26	50
10							57	109	58	110
11									58	110
12									139	266

Figure 4.11 Pivot path lengths for Morris's example in even dimension when dropping each label. The first column for each dimension corresponds to the Gale setting and the second column to the perfect matching case of G_d.

Next evaluate $L(i,d)$ for i even.

$$L(i,d) = L(i,i) + L(d-i,d-i) = \tag{4.11}$$
$$\frac{1}{2}\left[(1+\sqrt{2})^{\frac{i+2}{2}} + (1-\sqrt{2})^{\frac{i+2}{2}}\right] + \frac{1}{2\sqrt{2}}\left[(1+\sqrt{2})^{\frac{i-2}{2}} - (1-\sqrt{2})^{\frac{i-2}{2}}\right] - 2 +$$
$$\frac{1}{2}\left[(1+\sqrt{2})^{\frac{d-i+2}{2}} + (1-\sqrt{2})^{\frac{d-i+2}{2}}\right] + \frac{1}{2\sqrt{2}}\left[(1+\sqrt{2})^{\frac{d-i-2}{2}} - (1-\sqrt{2})^{\frac{d-i-2}{2}}\right] - 2$$

With $a = (1+\sqrt{2})$ and $b = (1-\sqrt{2})$ (4.11) becomes

$$\frac{1}{2}\left[a^{\frac{i+2}{2}} + b^{\frac{i+2}{2}}\right] + \frac{1}{2\sqrt{2}}\left[a^{\frac{i-2}{2}} - b^{\frac{i-2}{2}}\right] - 2 + \frac{1}{2}\left[a^{\frac{d-i+2}{2}} + b^{\frac{d-i+2}{2}}\right] + \frac{1}{2\sqrt{2}}\left[a^{\frac{d-i-2}{2}} - b^{\frac{d-i-2}{2}}\right] - 2 \tag{4.12}$$

Since $|b| \leq 1$ it is sufficient to rewrite (4.12) in terms of a.

$$\frac{1}{2}\left[a^{\frac{i+2}{2}} + \frac{1}{\sqrt{2}}a^{\frac{i-2}{2}} + a^{\frac{d-i+2}{2}} + \frac{1}{\sqrt{2}}a^{\frac{d-i-2}{2}}\right] - 4 \tag{4.13}$$

To minimize 4.13 we only need to consider the a terms with $+2$ in the exponent, that is

$$\min_{i\in\{2,\ldots,d\}} \left(a^{\frac{i+2}{2}} + a^{\frac{d-i+2}{2}}\right). \tag{4.14}$$

The minimum of (4.14) is attained by minimizing the maximum of the exponents $i+2$ and $d-i+2$, which is at $i = d/2$. Similarly as for $L(d,d)$ the dominating term for $L(d/2,d)$ is $(1+\sqrt{2})^{\frac{d}{4}}$, showing that $L(d/2,d)$ is in $\Theta((1+\sqrt{2})^{\frac{d}{4}})$. By similar reasoning equation 4.11 is maximized at $i = d$, so $L(d,d)$ is indeed the longest path. □

The table in Figure 4.11 shows the path lengths for Morris graphs.

Chapter 5

Pfaffian Orientations, Planar Graphs, and Counting Nash Equilibria

In Section 4.4 we showed that the pivoting algorithm can take exponentially many steps when the Euler graph is built from Morris's construction. In this chapter we show that if we can count the number of perfect matchings in an Euler graph efficiently then the problem ANOTHER SIGNED PERFECT MATCHING is in FP.

In Section 5.1 we give the definition of the complexity class #P which consists of counting problems. One member of #P is the problem which asks to count the number of perfect matchings in a bipartite graph. Valiant (1979) showed that this problem is #P-complete; a #P-complete is at least as hard an NP-complete problem. We show that this completeness result translates to counting perfect matchings in general bipartite Euler graphs.

However, in Section 5.2 we describe how the number of perfect matchings in planar graphs can be computed efficiently via a Pfaffian orientation (Kasteleyn (1967)). Thus, for the special case where the Euler graph is additionally planar we solve the problem ANOTHER SIGNED PERFECT MATCHING in polynomial time. This result is joint work with Ron Peretz. To my knowledge, no game-theoretic significance is known for games that are derived from planar Euler graphs. Section 5.3 gives an alternative proof to the already known result, namely that counting Nash equilibria in bimatrix games is #P-complete.

5.1 The complexity class #P and counting perfect matchings

If we can count the number of perfect matchings in an Euler graph in polynomial time then this would be a good starting point for an efficient algorithm to solve ANOTHER SIGNED PERFECT MATCHING. Counting perfect matchings is a so-called counting problem and thus a member of the complexity class #P; where "#" stands for "counting", "sharp" or "number".

The computational complexity class #P was first introduced by Valiant (1979) to settle the complexity of computing the permanent of an $n \times n$ matrix; A *counting* Turing machine consists of a nondeterministic Turing machine that additionally outputs the number of accepting computations induced by the input, e.g. a counting Turing machine that outputs the number of Hamiltonian tours of a graph. A problem P is a member of #P if there exists a counting Turing machine that computes P in polynomial time (Valiant (1979)). A problem P_2 is #P-complete, analogously to NP-completeness, if for all problems P_1 in #P, $P_1 \leq_p P_2$. Valiant (1979) makes a further distinction between members in #P. A problem is *P-enumerable* if the solutions can be printed in time $p(|x|) \cdot N$ where N is the number of solutions and $p(|x|)$ is a polynomial in the input size of the problem.

A counting Turing machine is an extension of a nondeterministic Turing machine. Output "NO" if the count is zero or else output "YES". Thus, the corresponding counting problem of an NP-complete problem is at least as hard as the original decision problem. Surprisingly, as we will see in more detail below, there exist problems that are in P, for example PERFECT MATCHING and EULER TOUR, which decides whether a graph admits an Euler tour, but their corresponding counting problems are both #P-complete; see Valiant (1979) and Mihail and Winkler (1992).

We adapt the notation from Section 3.6, where A is an $n \times n$ matrix and σ is a permutation in the symmetric group S_n, consisting of all the $n!$ permutations of $\big(1,2,\ldots,n\big)$; see Valiant (1979) and Vazirani and Yannakakis (1989). The permanent of A is defined by

$$perm(A) = \sum_{\sigma \in S_n} value(\sigma), \tag{5.1}$$

where

$$value(\sigma) = \prod_{1 \leq i \leq n} a_{i,\sigma(i)}. \tag{5.2}$$

Consider the bipartite graph $G = (X \cup Y, E)$ with the bipartition $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$ and adjacency matrix $A \in \{0,1\}^{n \times n}$. The permutation σ describes a perfect matching for G if and only if for all $i \in [n]$, $a_{i,\sigma(i)} = 1$. Thus

$$value(\sigma) = \begin{cases} 1, & \text{if } \sigma \text{ is a perfect matching of } G, \\ 0, & \text{otherwise.} \end{cases} \tag{5.3}$$

The permanent is closely related to the determinant of a matrix, with the only exception that the determinant includes the sign of the permutation in its formula,

$$det(A) = \sum_{\sigma \in S_n} sign(\sigma) \cdot value(\sigma). \tag{5.4}$$

Although similar in definition, the computational aspect of the two concepts could not be more different. The determinant of a matrix can be computed in *strongly polynomial time*. That is, in addition to admitting a polynomial algorithm, for example Gaussian elimination, the number of steps taken by the algorithm does not depend on the size of the numbers of the matrix; see Edmonds (1966). Computing the permanent was shown to be #P-complete by Valiant, even when A is restricted to a $0-1$ matrix. It follows from the complexity result of the permanent (Valiant (1979)) that counting the number of perfect matchings in a bipartite graph, #PMBG, is #P-complete; although computing one perfect matching is in P. Clearly this result translates to counting perfect matchings in general non-bipartite graphs, referred to as #PMG.

Next we transform an undirected bipartite graph into an Eulerian graph which is still bipartite and where the number of perfect matchings between the two graphs is in direct relation.

Lemma 5.1 *For a bipartite Euler graph $G = (V,E)$, counting the number of perfect matchings of G, #PMEBG, is #P-complete.*

Proof: Clearly #PMEBG is in the class #P, since it is easy to check whether a set of edges is a perfect matching for a graph. To show the completeness part we reduce #PMBG to #PMEBG. Let $G = (X \cup Y, E)$ be a bipartite graph with the bipartition $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$ and let the number of perfect matchings of G be k. If all nodes are balanced then the original graph is already Euler and we are done.

If there are unbalanced nodes, then there must be an even number of them. Let the augmented graph G' be given by $G' = G \cup C_4 \cup C_4$, where C_4 is a cycle with four vertices;

see Figure 5.1. Each cycle C_4 has a special vertex[1]. Label one vertex with x' in one

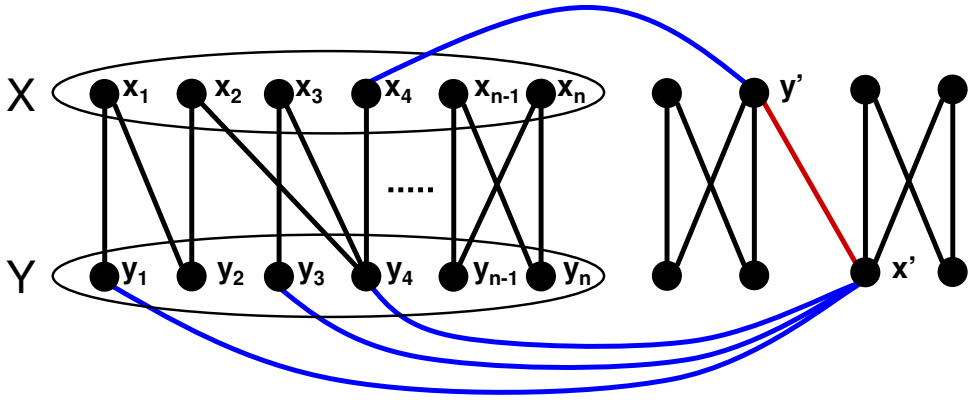


Figure 5.1 The augmented graph G'.

cycle, s.t. $x' \in X$ and the other vertex y' in the other cycle s.t. $y' \in Y$. Next, connect all unbalanced vertices from X with x' and similarly all unbalanced vertices of Y with y'. If the number of unbalanced nodes in X (and hence in Y) is even we are done because G' is already Eulerian. If not, then after connecting the unbalanced vertices, the vertices x' and y' are now unbalanced. Hence, add an edge between x' and y' to make G' Eulerian. Clearly the augmented graph is still bipartite and this construction is polynomial in the size of G as it adds a constant number of vertices and at most $2n+9$ edges.

The last part of the proof is to show that the number of perfect matchings in G' is $4k$. The cycle with vertex x' has two perfect matchings and none of the edges connecting x' to the other cycle or to G can be part of a perfect matching, this includes the edge (x', y'). The same follows for the other cycle. Hence, this construction does not influence the number of perfect matching in G as none of the added edges, apart from the two cycles which have 4 perfect matching between them, are in any matchings. This multiplies the original number of perfect matchings of G by 4, resulting in $4k$ perfect matchings in G'.

□

Clearly, Lemma 5.1 implies that counting perfect matchings in general, not necessarily bipartite, Euler graphs is also #P-complete. For bipartite graphs determining the number of perfect matchings is equivalent to computing the permanent of their $0-1$ adjacency matrix, as described above. The formula of the permanent is closely related to the one

[1] We could have used C_2 for the construction which would, however, have resulted in a multigraph instead.

of the determinant, but some matchings in the determinant may cancel out due to the negative sign of odd permutations.

This raises the following question formulated by Polya (1913), referred to as Polya's scheme (Vazirani and Yannakakis (1989)). Can we amend the initial adjacency matrix A by changing the sign of some entries from $+1$ to -1 such that for this new matrix, here called B, all perfect matchings σ satisfy $sign(\sigma) = value(\sigma)$? [2] Recall that the $value(\sigma)$ for a matrix with entries from $\{-1,0,+1\}$ is either plus or minus one if and only if σ constitutes a perfect matching, else it equals zero. If this can be achieved then clearly $det(B) = perm(A)$. The function problem associated with Polya's scheme is given by

POLYA'S PROBLEM

Input: An $n \times n$ $\{0,1\}$ matrix A.

Question: Is there an $n \times n$ $\{-1,0,1\}$ matrix B satisfying Polya's scheme?

Deciding the complexity of POLYA'S PROBLEM has been studied extensively over the last 30 years; see Thomas (2006) for a survey. It has recently been solved independently by McCuaig (2004) and Robertson, Seymour, and Thomas (1999). The authors show that a bipartite graph admits a Pfaffian orientation (see Section 5.2) if and only if it can be obtained from planar graphs and the so-called Heawood graph by repeating certain operations on the graph. Their analysis is, however, restricted to the bipartite case and no complete characterization for general graphs is known. Note that for bipartite graphs the pivoting algorithm computes another signed perfect matching in linear time (see Section 4.3). Interestingly, POLYA'S PROBLEM is equivalent to asking the following questions: When does a bipartite graph have a Pfaffian orientation, and when does a given digraph have no directed circuit of even length?

The concept of a Pfaffian orientation can be applied to general graphs. In the next section we investigate the class of Euler graphs which admit such a Pfaffian orientation. This allows us to efficiently solve ANOTHER SIGNED PERFECT MATCHING for Euler graphs that admit a Pfaffian orientation.

[2] Matrices A and B have no relationship to the matrices defining a bimatrix game.

5.2 Pfaffian orientations and planar graphs

In the previous section we outlined the difficulty to count the number of perfect matchings, even for Euler graphs. In this section we describe how for planar graphs we can calculate the number of perfect matchings efficiently; see Kasteleyn (1967). We will use this result as a "black box" to solve ANOTHER SIGNED PERFECT MATCHING for planar directed Euler graphs. We first discuss Pfaffian orientations, related to the Pfaffian discussed in Section 3.6.

An even-length cycle C of a graph $G = (V,E)$ is *central* or *good* if $G = (V \setminus C, E)$ has a perfect matching. For a directed graph a good cycle is called *oddly oriented* if it is not a sign-switching cycle (Vazirani and Yannakakis (1989)). For two different perfect matchings M and M' the symmetric difference of the two matchings is exactly a set of central cycles. Suppose we require that every central cycle of a graph is oddly oriented (for directed graphs). Then every perfect matching must have the same sign. An orientation which satisfies the above condition is called a *Pfaffian orientation* of G. Similarly, a graph that admits one is called *Pfaffian*. As discussed in Section 3.6 the Pfaffian can be used to count the number of perfect matchings in general graphs.

The importance of a Pfaffian orientation is that if we find such an orientation then the number of perfect matchings can be efficiently computed by evaluating the determinant via the equation from Lemma 3.12, $Pf(A)^2 = det(A)$, where A is the skew adjacency matrix derived from the graph with a Pfaffian orientation. Because the sign does not depend on σ, no two perfect matchings cancel out and the number of perfect matchings is thus $|Pf(A)|$. In the special case of a bipartite graph this means that we can also efficiently compute the permanent with the help of the Pfaffian. Kasteleyn (1967) showed that certain graphs always admit a Pfaffian orientation.

Lemma 5.2 *(Kasteleyn (1967)) Every planar graph is Pfaffian.*

We give a detailed constructive argument why Lemma 5.2 holds because we need this results to prove a later theorem (see Figure 5.2 for an example of the argument). Given a planar graph we can apply the Fisher, Kasteleyn, and Temperley [FKT]-algorithm to find a Pfaffian orientation of the graph, which we briefly outline next. Given a planar embedding of a directed graph G, first compute a directed spanning tree T of G. Create a second undirected tree T' which has the vertex set of the dual graph of G, that is one

vertex for every face of G. Connect two vertices of T' if they are on adjacent faces of G and the connecting edge does not cross any edge from T. Now, given T', add edges to T by starting from the leaves of T'. Pick any leaf l of T' and direct the missing edge of the face representing l such that this face has an odd number of clockwise directed edges. Delete the leaf l from T', i.e. prune the tree, and continue the last step until T' has no more vertices. The resulting graph has the same vertex and edge set than G, however, some edges might be differently oriented. This graph is now Pfaffian which means by taking the square root of the determinant of the skew adjacency matrix we can efficiently calculate the number of perfect matchings.

We explain how to obtain a Pfaffian orientation for the adjacency matrix A of G_6. The Pfaffian orientation is given in the augmented matrix B. We neglect multiple edges in A; see below.[3],

$$A = \begin{pmatrix} 0 & 1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & -1 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & -1 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & -1 & 0 \\ -1 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & -1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 \end{pmatrix}$$

From the planar graph G_6 we construct a spanning tree T.[4] Then construct the tree T' (see Figure 5.2) and switch the edges according to the FKT-algorithm. The result is a graph that is Pfaffian.

We can place an additional restriction on the directed spanning tree such that given an Euler graph and a perfect matching M the spanning tree T contains all edges from M. This is achieved by giving weight zero to matched and weight one to unmatched edges and then constructing a minimum spanning tree, for example with Kruskal's algorithm. This has the nice property that only unmatched edges with respect to M are redirected to obtain a Pfaffian orientation. This leads to an interesting observation. Although we can efficiently compute the number of perfect matchings for any planar graph, the question is whether we can use the Pfaffian to find an opposite signed perfect matching for a planar Euler graph.

[3] We showed in Chapter 3.3 that this is not a restriction.

[4] Note that more than one spanning tree exists.

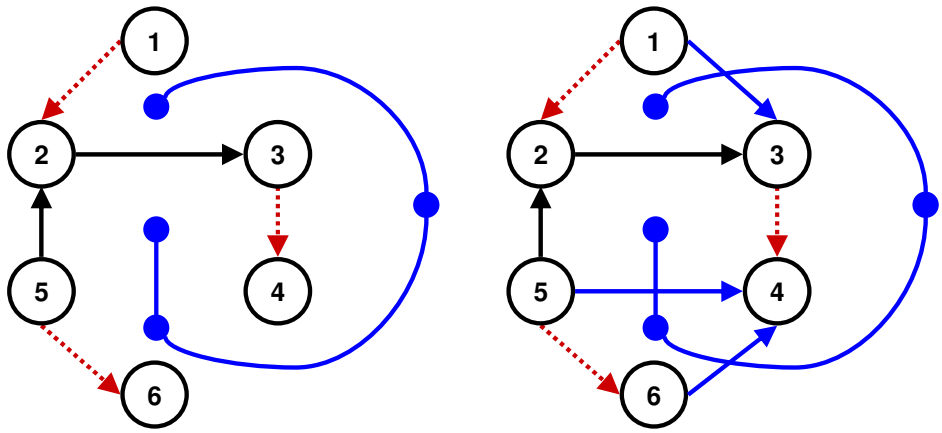


Figure 5.2 The left graph is the spanning tree T for G_6 together with the blue tree, T', which has the vertex set of the dual of G_6. On the right, T is completed to G_6 such that G_6 is now Pfaffian. By redirecting the original edge $(3,1)$ the only sign-switching cycle of G_6 is now oddly oriented.

In the above example we can quickly check that only one of the two edges that were switched to obtain a Pfaffian is part of a perfect matching, namely the edge $(3,1)$. In this simple case we then obtain the new matching by deleting the edge $(1,3)$ to find the only other matching which is of opposite sign. In general, however, an exhaustive approach can lead to exponential search time when starting from a perfect matching M. Consider only switched edges that are part of some perfect matching other than M. Delete one such edge e, find another matching M' and calculate the symmetric difference $M \oplus M'$ of two matchings. If e is in a cycle with an odd number of edges or one of the cycles in $M \oplus M'$ has an odd number of switched edges from the KST-algorithm then we have found another signed matching. However, e could also be in a cycle with two (or any other even number) switched edges, such that the cycle with respect to the Pfaffian orientation is not sign switching. This can occur when the reoriented edges have to be switched because they are members of a different central cycles. Hence, even though starting from an edge that is in a switching cycle we can end up with a perfect matching M' that is not sign switching. Because there can be an exponential numbers of non sign-switching cycles this approach fails. It seems that although we guarantee that all central cycles are not sign switching, finding one that was previously a sign-switching cycle given the original Euler orientation with the knowledge of the Pfaffian orientation does not follow immediately. Theorem 5.3 shows that by selecting a special edge the problem is polynomial-time solvable; see Figure 5.3 and Figure 5.4 for an example of the argument.

Theorem 5.3 ANOTHER SIGNED PERFECT MATCHING *is polynomial-time solvable if the graph is a planar Euler graph.*

Proof: Our proof constructs M' in at most polynomially many steps. First note that for a directed Euler graph G we know that the number of negative and positive signed perfect matchings is non zero; in our setup these numbers are identical.[5] Preprocess G by removing all edges that are not part of any perfect matching, which takes at most polynomial time (see page 58). Then G may be no longer Eulerian but this does not matter.

Next, we describe how to find an edge e in G which is part of both positive and negative signed perfect matchings; their numbers do not have to be identical. First calculate $Pf(A)$ and the number k of perfect matchings of G, where A is the skew-symmetric adjacency matrix of G. Both numbers can be calculated in polynomial time due to Lemma 3.12 and because we can compute the number of perfect matchings of G by finding a Pfaffian orientation with the FKT-algorithm, as described earlier in this chapter. Next let G' be given by removing e and its adjacent vertices from G. Then G' is still planar. Again calculate the Pfaffian $Pf(A')$, where A' is the skew-symmetric adjacency matrix of G', and the number k' of perfect matchings in G'. If

$$\big||Pf(A)| - |Pf(A')|\big| \neq \big|k - k'\big| \tag{5.5}$$

then e is part of both negative and positive signed matchings, otherwise the $k - k'$ fewer perfect matchings in G' would all have the same sign, and we remove it from G. After deleting an edge, G' is not necessarily Eulerian anymore, but still has a nonnegative number of positive and negative signed perfect matchings. This is the invariant of the argument. First note that we cannot remove all perfect matchings by removing one edge e. This would imply that after deleting e the graph is empty. This is a contradiction to the fact that e is part of positive and negative signed perfect matchings. Also, the number of positive (and negative) matchings cannot be zero after removing e. This holds because if an edge e is part of both signed perfect matchings then there must exist at least two different edges in the graph, say e_1 and e_2, such that $e \cup e_1$ and $e \cup e_2$ are in a positive and negative matching, respectively. In other words the edges e_1 and e_2 determine the sign of the matching, not e. Then after removing e the different matchings containing e_1 and

[5] In fact, we can relax this condition to require that we only know that there are a nonnegative number of positive and negative signed perfect matchings in the planar graph.

e_2 cannot both have the same sign. After e is deleted continue the deletion process in the smaller graph.

If equation (5.5) holds with equality then e is only part of matchings of one sign, upon which it is not deleted from G and we inspect the edges of G that have not been checked. If we cannot delete any more edges then we are left with a subgraph G^* of G for which each edge is only part of either positive or negative signed matchings. By the invariant it is assured that G^* has matchings of both signs. We arrive at G^* after at most $|E|^2$ steps, because we need to inspect each edge only once after some other edge was deleted.

To find a perfect matching of negative sign of the original graph, compute a perfect matching $\hat{M}$ in $G - G^*$, whose vertex set is given by $V - V^*$; note that $G - G^*$ is not necessarily connected anymore. Depending on the sign of $\hat{M}$ find a perfect matching M^* in G^*, such that $sign(\hat{M} \cup M^*) = -1$. To find M^* we only need to find one perfect matching for each edge of G^* which by construction has a unique sign, and of which at least one of them has the appropriate sign. Lastly, set $M' = \hat{M} \cup M^*$. This solves ANOTHER SIGNED PERFECT MATCHING in polynomial time. □

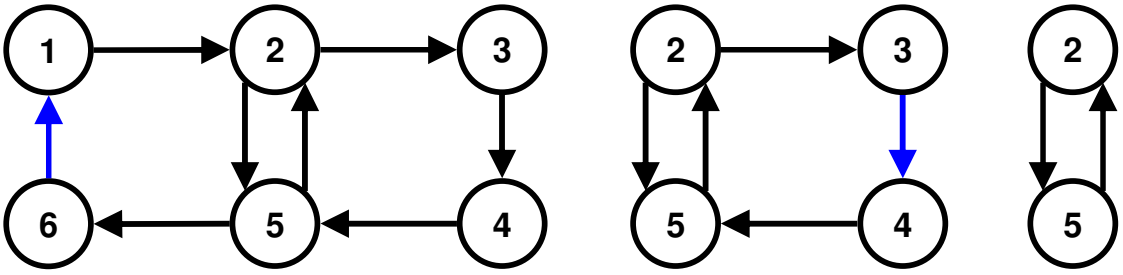


Figure 5.3 The left graph G is the original graph for which we have to find a negative signed matching. G has 4 perfect matchings and its Pfaffian is zero. Pick an edge, say $(6,1)$. G', the middle graph, has 3 perfect matchings and $|Pf(A')|$ is equal to one because the only negative signed perfect matching is given by edges $(3,4)$ and $(5,2)$. Hence the edge $(6,1)$ is part of matchings of both signs and we delete it. Repeat the process for the middle graph. Here the only edge which can be deleted is $(3,4)$. This gives us G^*, the rightmost graph.

In the proof of Theorem 5.3 we used a "black box" to decide whether an edge is in a positive and negative signed perfect matching (here the black box counted the number of perfect matchings by calculating the Pfaffian of the graph). Note that this black box still gave an answer in polynomial time even after we deleted edges including its incident

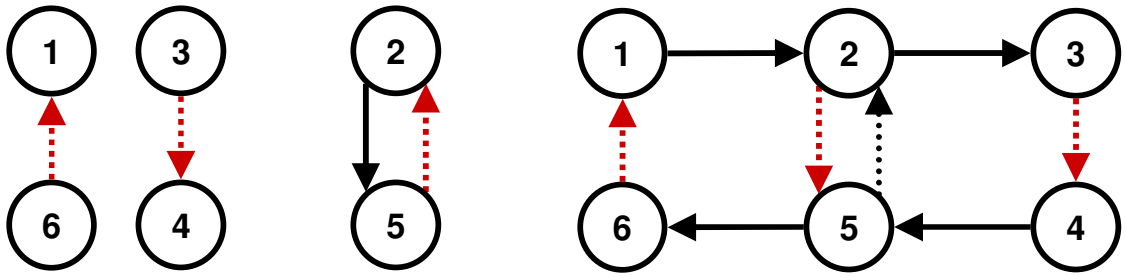


Figure 5.4 Find a perfect matching in $G - G^*$, the leftmost graph, given by edges $(6,1)$ and $(3,4)$. There is only one such perfect matching, which has odd sign. In G^* there exists a perfect matching of either sign, we pick edge $(2,5)$, which has positive sign. Combining the two perfect matching gives a negative signed perfect matching of G at $M' = \{(6,1)(2,5)(3,4)\}$.

vertices; the black box is thus closed under edge deletion. We also had an "oracle" that answered us whether a graph has a positive and negative perfect matching before we started to delete edges (here simply the fact that the graph is Eulerian). Note that we used the oracle only once before we started to delete edges. If both, the black box and the oracle, take polynomial time and the black box is additionally closed under edge deletion then the argument used to prove Theorem 5.3 extends to a larger class of graphs; this is stated in the following Corollary 5.4.

Corollary 5.4 *Suppose that for a class of graphs $\mathcal{G}$ we can decide the following questions in polynomial time:*
(1) Does a graph in $\mathcal{G}$ admit positive and negative signed perfect matchings?
(2) Is an edge of a graph in $\mathcal{G}$ in positive and negative signed perfect matchings?
Additionally suppose that for graphs in $\mathcal{G}$ we can answer question (2) in polynomial time even after we deleted some edges in any sequence. Then for this class of graphs the problem ANOTHER SIGNED PERFECT MATCHING *is in FP.*

Interestingly, if we can answer question (1) in polynomial time after some sequence of edge deletions, as specified above, then we can also answer question (2) in polynomial time. To see this, take any edge e and delete it including its incident vertices. Now answer question (1) for the graph without e. The answer of question (1) is then the answer to question (2) before deleting e. However, this relationship does not hold in the other direction. For example, consider the directed graph consisting of vertices 1 and 2, connected by directed edges $(1,2)$ and $(2,1)$. Question (2) answers "NO" for this graph since each

edge is in either a positive or negative perfect matching, but not in both. Question (1), however, answers "YES".

5.3 Counting Nash equilibria is hard

Computing one Nash equilibrium in a bimatrix game is PPAD-complete; see Section 2.2. An important question for characterising the strategic structure of a bimatrix game (and games in general) is to ask the question, how many Nash equilibria does a game admit? For certain bimatrix games with d pure strategies for each player, von Stengel (1999) showed that the number of equilibria can be exponential in the number of pure strategies. The problem of computing or counting the number of Nash equilibria is referred to as #NASH, Theorem 5.5 shows that the following counting problems are all #P-complete. The first three results are new to my knowledge, while the result regarding Nash equilibria has been shown by Conitzer and Sandholm (2003). They reduce #CNF, that is the problem of counting the number of assignments satisfying a Boolean formula in conjunctive normal form, directly to a symmetric bimatrix game. Our proof uses the areas central to this thesis to arrive at the same result.

Theorem 5.5 *The following counting problems are #P-complete.*

1. *Counting perfect matchings in an Euler graph,* #PMEG*,*
2. *Counting completely labeled Gale strings,* #CLGS*,*
3. *Counting completely labeled facets in a labeled cyclic polytope,* #CLFCP*,*
4. *Counting Nash Equilibria in a bimatrix game,* #NE*.*

Proof:

The first result follows from Lemma 5.1. Clearly the remaining problems are in #P because they can be computed with a counting Turing machine in polynomial time. We elaborate for the Nash equilibria case. A strategy profile for a bimatrix game only has rational entries and thus can be encoded by a Turing machine. It can be checked in polynomial time whether each strategy that is played with positive probability has the same expected payoff against the strategy in the support of the other player. This payoff must be strictly higher than the payoffs from the other zero-probability strategies, due

to the nondegeneracy assumption. For games with more than two players this is not as straightforward as the probability vector can have irrational entries.

Next we show #PMEG $\leq_p$ #CLGS. An Euler tour for an Euler graph can be computed with Hierholzer's algorithm in polynomial time. From Section 3.3 it follows that a perfect matching in an Euler graph is in a one-to-one relationship with a completely labeled Gale string for a given Euler tour, thus preserving the number of perfect matchings. Clearly this reduction is polynomial, so the second statement follows.

From Gale (1963), completely labeled Gale strings and completely labeled facets of a cyclic polytope are in a one-to-one correspondence; see Section 3.1. In case there does not exist a completely labeled Gale string it is still possible to construct a cyclic polytope. The labeling, however, will not admit a completely labeled facet, the count thus being zero. The third statement follows as a result.

Let $\tilde{Q}$ be a cyclic labeled polytope. The last reduction requires an additional step, because the polytope $\tilde{Q}$ might not admit a completely labeled facet. In contrast, a bimatrix game must admit a Nash equilibrium. First check whether the polytope admits a completely labeled facet. If no completely labeled facet exists, output zero as the count. Otherwise, compute a completely labeled facet and call it F_0. This requires at most polynomially many steps, by the equivalence to Gale strings and perfect matching as shown in Chapter 3.

In case there exists one completely labeled facet, we know that there exists another. To use Theorem 3.3, F_0 has to be of the form $F_0 = conv\{-e_1, \ldots, -e_d\}$. Let the vertices of F_0 be given by the d-dimensional vectors $f_1, \ldots, f_d$. Define an affine mapping $A : \tilde{Q} \to Q$ by the $d \times d$ matrix A which is the inverse of the matrix whose d column vectors are the vertices of F_0. The inverse exists because the vectors $f_1, \ldots, f_d$ are linearly independent. Constructing the inverse of a matrix with, for example, the Gaussian algorithm and mapping the n vertices of $\tilde{Q}$ with the affine transformation A to Q takes at most polynomially many steps. This transformation preserves the number of completely labeled facets. The polytope Q now has the required format to apply Theorem 3.3 and to construct the game (U,B). The number of completely labeled facets of $\tilde{Q}$ is equal to the number of Nash equilibria in the bimatrix game (U,B) plus one, as F_0 is only an artificial equilibrium of the game. □

Chapter 6

Towards a Polynomial Algorithm

In this chapter we consider general Euler graphs and discuss several approaches that may possibly lead to a polynomial-time algorithm to solve ANOTHER SIGNED PERFECT MATCHING. With the exception of the hardness result in Lemma 6.1, this chapter only presents partial results in that direction.

In Section 6.1 we show that for general directed graphs the decision problem which decides whether another signed perfect matching exists, called EXISTENCE OF SIGNED PERFECT MATCHING, is NP-complete. Here, we also show that for Morris graphs there exists an edge pairing, derived from a cycle partition of the Euler tour, which gives rise to short pivot paths for the pivoting algorithm. However, this is not a general algorithm.

In Section 6.2 we describe the solution set of the pivoting algorithm and show that for general graphs it lacks a "nice" structure; contrary to the solution set of bipartite graphs. We conjecture that if an Euler graph admits a pair of equally oriented edges then we can use a contraction argument to find another perfect matching efficiently. In Section 6.3 we describe how to extend Edmonds's algorithm to incorporate the Euler tour. We argue that this is not easily achieved.

6.1 The necessity of the Euler tour and special pairings

The LH-algorithm is the only algorithm known to us which specifically computes another perfect matching with a different sign without checking all the possibly exponentially many perfect matchings. As we have seen in the previous section, using the Euler tour for

the pairing of the edges resulted in an exponential running time of the pivoting algorithm for the Morris graph, independent of which initial vertex was dropped.

This poses two questions. First, is it necessary to consider the Euler tour, or in general any pairing of the edges, to find another matching? Second, does there exist a pairing of the edges that will speed up the pivoting algorithm? To address the first point we define the following decision problem.

EXISTENCE OF SIGNED PERFECT MATCHING

Input: A directed graph $G = (V,E)$ and a perfect matching M, s.t. $sign(M) = 1$.

Question: Does there exist a perfect matching M', s.t. $sign(M') = -1$?

Note that if the graph does not admit an Euler tour or edge pairing Theorem 3.8 does not necessarily hold anymore. The decision problem is in fact not total, for example the answer for the graph consisting of only one directed edge is "NO".

Lemma 6.1 *Deciding* EXISTENCE OF SIGNED PERFECT MATCHING *is NP-complete, even if the graph is bipartite.*

Proof: Let M' be a set of edges of G. Then it is easy to check in polynomial time in the size of M' and G whether M' is a perfect matching with negative sign; NP membership follows. Next, we give a polynomial-time reduction from the NP-complete problem EVEN PATHS IN DIGRAPHS; see LaPaugh and Papadimitriou (1984).

EVEN PATH IN DIGRAPHS

Input: A directed graph $G = (V,E)$ and $s,t \in V$.

Question: Does there exist an even-length simple path from s to t?

Given a digraph $G = (V,E)$ and two nodes s and t of V, we construct a new digraph $G' = (V',E')$ by copying the vertex set as follows:

$$V' = X \cup Y, \text{ where } X = (V \setminus \{t\}) \times \{1\} \text{ and } Y = (V \setminus \{s\}) \times \{2\}, \tag{6.1}$$
$$E' = \{((v,1),(v,2)) | v \in V \setminus \{s,t\}\} \cup \{((s,1),(t,2))\} \cup \{((u,1),(v,2)) | (u,v) \in E\}.$$

The new digraph G' is bipartite, where X and Y are the bipartition. We can assume that $(s,1)$ is the only source and $(t,2)$ is the only sink of G', because any sink or source will not be used for the reduction. For ease of notation and to be consistent with the sign of a perfect matching, label the vertices in X and Y naturally with ascending odd and

even numbers such that $(t,2)$'s label is the successor of the label of $(s,1)$ and in general $(v,2)$'s label is the successor of the label of $(v,1)$, as illustrated in Figure 6.1. Clearly $M =$

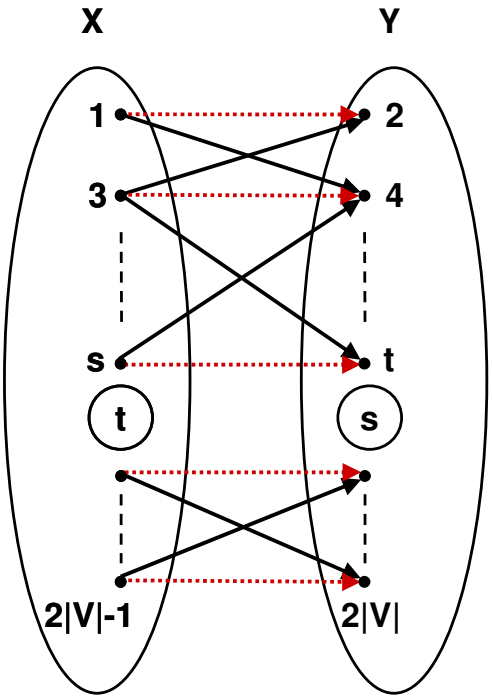


Figure 6.1 The augmented graph G'.

$\{((v,1),(v,2))|v \in V \setminus \{s,t\}\} \cup \{((s,1),(t,2))\} = \{(1,2),(3,4),\ldots,(2|V|-1,2|V|)\}$ is a positively signed perfect matching of E'. If we take out the matched edge $((s,1),(t,2))$, then another perfect matching M' of G' that does not use $((s,1),(t,2))$ has opposite sign to M if and only if there is an augmenting path from $(s,1)$ to $(t,2)$ of length $4k-1$ where k is a positive integer. This defines a sign-switching cycle of length $4k$ that contains $(s,1)$ and $(t,2)$, where the matched and unmatched edges are swapped. Because G' is bipartite, any cycle is of even length which is either $4k$ or $4k-2$. The latter does not change the sign of the permutation because it involves an odd number of unmatched edges which are switched when they become the new matched edges. This shows EVEN PATHS IN DIGRAPHS $\leq_P$ EXISTENCE OF SIGNED PERFECT MATCHING, completing the proof. □

Lemma 6.1 shows that even in the bipartite case, it is NP-complete to find another matching of different sign if we consider a general directed graph. Note that this does not automatically imply that we have to consider the Euler tour in order to hope to find a polynomial-time algorithm for solving ANOTHER SIGNED PERFECT MATCHING. It might suffice to use different properties of a directed Euler graph with a balanced orientation.

We now address the second question. If the Euler tour does not give us a good edge pairing, as in the Morris graph, does there always exist a pairing of the edges that requires the pivoting algorithm to only take a linear number of steps? We give such a pairing for Morris graphs. Independent of which initial vertex is dropped the number of steps taken by the pivoting algorithm is linear in the size of G_d.

Lemma 6.2 *For $d \geq 2$ there exists an edge pairing for G_d such that for all $i \in [d]$, $L(i,d) \in \Theta(d)$.*

Proof: We first give the pairing and then show that the length of each pivot path is at most d. Consider the Euler tour of the Morris graph, given by equation (4.2) and construct the pairings from the edge disjoint set of cycles $\{C_1, \ldots, C_{d/2}\}$[1] that forms the original Euler tour. For $d = 2$ there is only one possible pairing, so we are done as $L(1,2) = L(2,2) = 1$. For $d = 4$, let $C_1 = \{1,2,3\}$ and $C_2 = \{2,3,4\}$. For $d \geq 6$ partition G_d into

$$\begin{aligned} C_1 &= \{1,2,3\} \\ C_2 &= \{2,3,4,5\} \\ &\vdots \\ C_{d/2-1} &= \{d-4, d-3, d-2, d-1\} \\ C_{d/2} &= \{d-2, d-1, d\}, \end{aligned} \tag{6.2}$$

where adjacent edges in a cycle are paired.

The remaining proofs are similar to the ones from the Section 4.4 and we thus only point to the lemmata from the previous section, when possible. First observe that $L(i,d) = L(d-i+1,d)$ for $i \in [d]$. See Lemma 4.7 for the proof. Next, it is easy to to see that $L(1,d) = L(1,d-2)+2$, with initial condition $L(1,2) = 1$ and $L(1,4) = 3$. [2] Starting from $M_0 = M$, for $d \geq 6$, dropping vertex 1 leads to $M_1 = \{(2,3),(3,4),\ldots,(d-1,d)\}$ where the duplicate vertex is now 3. Dropping label 3 starts a sub-path of pivots equivalent to dropping vertex 1 in G_{d-2} with the pairing given above. By similar reasoning as in the proof of Lemma 4.8, vertex 3 can only be picked up again by first including edge $(5,2)$ and then $(2,3)$, showing that the subpath only belongs to G_{d-2}. The first extra step is

[1] Here a cycle is conveniently given by its vertices, where edges are between successive vertices.

[2] This is where the runtime of the pivoting algorithm differs from the Morris graph with the original pairing from Et_d as there is only one recursive call of $L(i,d)$.

incurred prior to dropping label 3 and the second extra step by dropping label 2 after $L(1,d-2)+1$ steps to pick up label 1. Solving the recursion $L(1,d) = L(1,d-2)+2$ with initial condition $L(1,2) = 1$ leads to $L(1,d) = d-1$. This holds for $d = 2$ and by induction we have $L(1,d) = L(1,d-2)+2 = (d-2)-1+2 = d-1$.

The statements $L(i,d) = L(i+1,d)$ and $L(i,d) = L(1,i)+L(1,d-i)$ for $i < d$ and i even are proven identically to Lemma 4.6 and Lemma 4.7. Insert $L(1,d) = d-1$ into $L(i,d) = L(1,i)+L(1,d-i)$ to obtain $L(i,d) = d-2$. The linear asymptotic behaviour for $L(i,d)$ for all $i \in [d]$ follows. □

Lemma 6.2 asks whether there always exists a *good edge pairing* which assures polynomial running time of the pivoting algorithm. This would imply that if there exists a short path between any two perfect matchings of different sign then finding this path would solve the different signed matching problem efficiently. Finding this short path would be interesting because the graph of skew matchings is of exponential size and there are paths of exponential length connecting two perfect matchings. This poses another question. Assuming that such a path always exists, what is the complexity of computing it?

6.2 The solution set of the pivoting algorithm and contracting pairs of equally oriented edges

In Section 4.3 we showed that the pivoting algorithm takes only linearly many steps if the graph is bipartite. The reason for this is that D_t (see Section 4.2) and $M \oplus M'$ admitted a "nice" structure. The question is, whether this structure exists for general graphs as well. This structure would give a good starting point for an efficient algorithm.

The output or solution space of the pivoting algorithm admits a special structure for bipartite graphs. Analysing the symmetric difference between M and M' when these are the end points of the pivot path, the symmetric difference $M \oplus M'$ for bipartite graphs satisfies the following properties.

1. $M \oplus M'$ is comprised of only one sign-switching cycle C.
2. C has only pairs of equally oriented edges that are paired in the Euler tour or given by the pairing.

These properties follow directly from the proof of Lemma 4.3. A consistent structure for general graphs, however, is not known to us as the following properties hold:

1. $M \oplus M'$ can be comprised of more than one cycle, some of which can be non-sign switching.
2. The sign-switching cycles can have only oppositely oriented pairs of edges.
3. Edges in C may not be paired, even if they are pairs of equally oriented edges.

We list the graphs for each case. Instead of giving each skew matching or D_t of the pivoting algorithm we give an abridged version of the pivot steps by considering the LHG-algorithm for the equivalent Gale string, defined by the Euler tour.

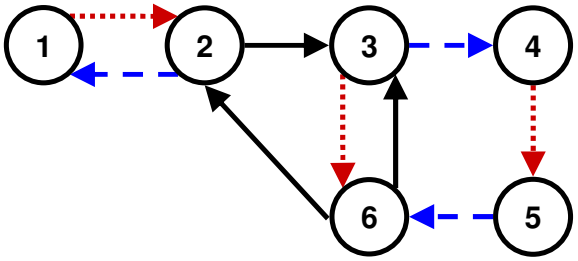


Figure 6.2 A sign and non sign-switching cycle of $M \oplus M'$.

step	**1**	**2**	**3**	**4**	**5**	**6**	**3**	**6**	**2**
1	$\underline{1}$	1	.	1	1	.	1	1	.
2	.	1	$\bar{1}$	1	1	.	$\underline{1}$	1	.
3	.	$\underline{1}$	1	1	1	.	.	1	$\bar{1}$
4	.	.	1	1	1	$\bar{1}$	.	$\underline{1}$	1
5	$\bar{1}$	.	1	1	1	1	.	.	1

Figure 6.3 Pivot path for Figure 6.2 when dropping label 1.

Case 1. From the initial perfect matching M, in red, the other perfect matching M' in blue is reached by dropping label 1 from the label string $l = 123456362$; see Figure 6.2. The pivot steps for the corresponding Gale string setup are in Figure 6.3. The symmetric difference between M and M' contains two cycles, $C_1 = \{(1,2),(2,1)\}$ and $C_2 = \{(3,4),(4,5)(5,6)(3,6)\}$ where the edges (u,v) are directed from u to v. The cycle C_1 is sign switching.

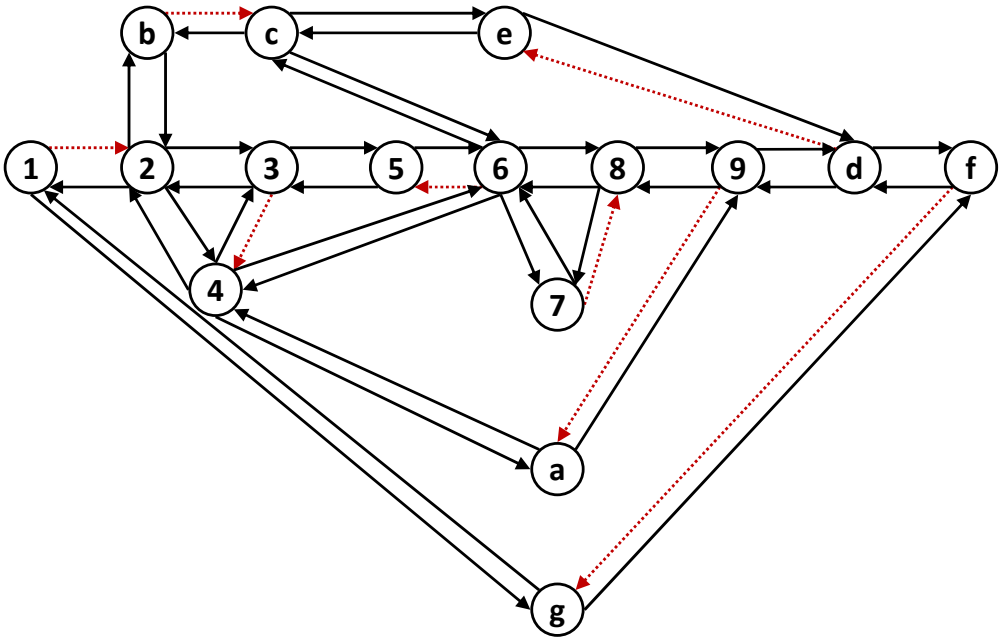


Figure 6.4 A graph where $M \oplus M'$ consists only of pairs of oppositely oriented edges. The sign-switching cycle C_1 is given by edges $C_1 = \{(3,5),(6,5),(6,4),(3,4)\}$ when initially dropping vertex 1.

Case 2. The graph is given in Figure 6.4 with the corresponding Euler tour given by $Et = 1235678689a4342b24a9decbc646c646cedfg1gfd986532$ with the initial matching given by the red dotted edges. The matching M' reached by dropping label 1 of M is such that $M \oplus M'$ is given by the only sign-switching cycle C defined by the edges $C = \{(3,5),(6,5),(4,6),(3,4)\}$, where C has two pairs of oppositely oriented edges. Although $M \oplus M'$ does not have a pair of equally oriented edges, the graph has sign-switching cycles where both edges are equally oriented, $C = \{(1,2),(2,1)\}$ for example.

Case 3. We only give the label string and the two perfect matchings for this case. For $l = 12345643$ the only two matchings are given by the strings $s = 11111100$ and $s' = 11001111$, whose symmetric difference is given by edges $\{(3,4)(4,3)\}$ which are not adjacent in the Euler tour.

step	1	2	3	5	6	7	8	6	8	9	a	4	3	4	2	b	2	4	a	9	d	e	c	b	c	6	4	6	c	6	4	6	c	e	d	f	g	1	g	f	d	9	8	6	5	3	2
1	$\underline{1}$	1	.	.	.	1	1	.	.	1	1	.	1	1	.	.	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.	.	1	1	.	.
2	.	1	$\bar{1}$	.	.	1	1	.	.	1	1	.	$\underline{1}$	1	.	.	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.	.	1	1	.	.
3	.	$\underline{1}$	1	.	.	1	1	.	.	1	1	.	.	1	$\bar{1}$	.	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.	.	1	1	.	.
4	.	.	1	$\bar{1}$	.	1	1	.	.	1	1	.	.	1	1	.	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.	.	.	1	$\underline{1}$	.	.
5	.	.	1	1	.	1	$\underline{1}$	.	.	1	1	.	.	1	1	.	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.	$\bar{1}$	1	.	.	.
6	.	.	1	1	$\bar{1}$	1	.	.	.	1	1	.	.	1	1	.	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.	.	1	$\underline{1}$	.	.	.
7	.	.	1	1	1	1	.	.	.	$\underline{1}$	1	.	.	1	1	.	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.	$\bar{1}$	1	.	.	.	.
8	.	.	1	1	1	1	.	.	.	.	1	$\bar{1}$	.	$\underline{1}$	1	.	.	.	.	.	1	1	.	1	1	.	.	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.	1	1	.	.	.	.
9	.	.	1	1	1	1	.	.	.	.	1	1	.	.	1	$\bar{1}$	.	.	.	.	1	1	.	$\underline{1}$	1	.	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.	1	1	.	.	.	.
10	.	.	1	1	$\underline{1}$	1	.	.	.	.	1	1	.	.	1	1	.	.	.	.	1	1	.	.	1	$\bar{1}$	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.	1	1	.	.	.	.
11	.	.	1	1	.	1	$\bar{1}$	.	.	.	1	1	.	.	1	1	.	.	.	.	1	1	.	.	1	1	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.	.	1	$\underline{1}$	.	.	.	.
12	.	.	1	1	.	1	1	.	.	.	1	1	.	.	1	1	.	.	.	.	$\underline{1}$	1	.	.	1	1	.	.	.	.	.	.	.	.	1	1	.	.	.	.	$\bar{1}$	1	.	.	.	.	.
13	.	.	1	1	.	1	1	.	.	.	1	1	.	.	1	1	.	.	.	.	.	1	$\bar{1}$	.	$\underline{1}$	1	.	.	.	.	.	.	.	.	1	1	.	.	.	.	1	1	.	.	.	.	.
14	.	.	1	1	.	1	1	.	.	.	1	$\underline{1}$	.	.	1	1	.	.	.	.	1	1	.	.	1	$\bar{1}$	.	.	.	.	.	.	.	.	1	1	.	.	.	.	1	1	.	.	.	.	.
15	.	.	1	1	.	1	1	.	.	$\bar{1}$	1	.	.	.	1	1	.	.	.	.	1	1	.	.	1	1	.	.	.	.	.	.	.	.	1	1	.	.	.	.	1	$\underline{1}$	.	.	.	.	.
16	.	.	1	1	.	1	1	.	.	1	1	.	.	.	1	1	.	.	.	.	1	1	.	.	1	1	.	.	.	.	.	.	.	.	$\underline{1}$	1	.	.	$\bar{1}$	1	.	.	.	.	.	.	.
17	.	.	1	1	.	1	1	.	.	1	1	.	.	.	1	1	.	.	.	.	1	1	.	.	1	1	.	.	.	.	.	.	.	.	.	1	$\bar{1}$	.	1	1	.	.	.	.	.	.	.

Figure 6.5 Pivot path for Figure 6.2 when initially dropping label 1. The sign-switching cycle C_1 is given by edges $C_1 = \{(3,5),(6,5),(6,4),(3,4)\}$ and the other cycle is $C_2 = \{(1,2),(2,b),(b,c),(e,c),(d,e),(f,d),(f,g),(g,1)\}$.

Although Case 2 (see Figure 6.4) gives an example which shows that the symmetric difference of M and M', as endpoints of the pivot path, can have only one sign-switching cycle which has only oppositely oriented pairs of edges, this does not disprove the following Conjecture 6.3.

Conjecture 6.3 *Given an Euler graph and a perfect matching M, there exists a different signed perfect matching M' such that $M \oplus M'$ has a sign-switching cycle C which has at least one pair of equally oriented edges.*

Suppose Conjecture 6.3 holds. The idea is to contract pairs of equally oriented edges in order to find a sign-switching cycle. The contraction must be such that afterwards the conjecture still holds; see Figure 6.6. Suppose $(u,v),(v,w)$ is some pair of equally oriented edges where $(u,v) \in M$. Note that we do not know a priori if the pair of equally oriented edges is in a sign-switching cycle. Contract the pair of edges into a *starred vertex* v^* as shown in Figure 6.6. Adjacent edges of v, other than (u,v) and (v,w), for

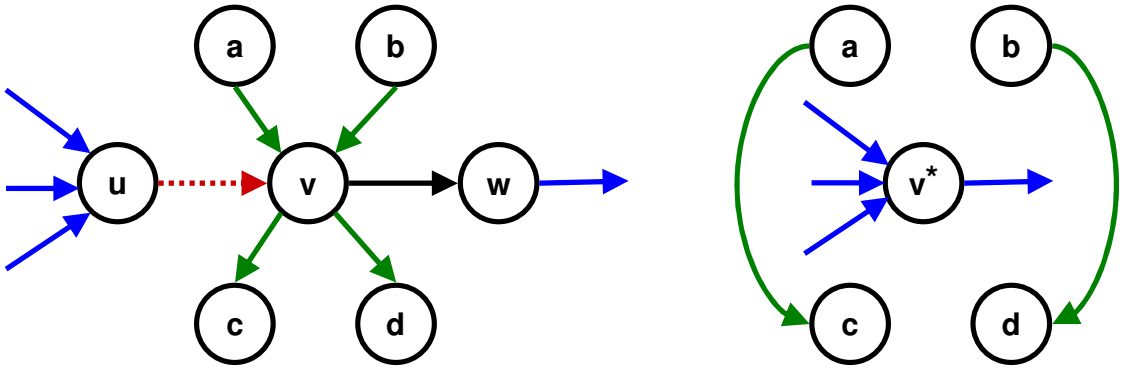


Figure 6.6 Contraction of a pair of equally oriented edges.

example (a,v) or (v,c), are not contracted. We simply connect these edges according to the Euler tour. For example, if $(a,v),(v,c)$ are adjacent in the Euler tour then they receive a single edge (a,c) after the contraction. Clearly the Euler property is still satisfied after the contraction. In particular, Corollary 6.3 then still holds after a contraction.

In order to find a sign-switching cycle, successively contract pairs of equally oriented edges until the graph consists of only one such pair. This must be a sign-switching cycle after a sequence of contractions. Next, reverse the contraction to build a sign-switching cycle of the original graph. This might, however, not lead to a sign-switching cycle of

the original graph because we could have contracted a pair of edges that is not part of a sign-switching cycle. Hence, in order to avoid exponential running time, the contraction process has to be such that once an edge is contracted it will either be contained in a sign-switching cycle or the edge cannot be contracted again. This will ensure that not all combinations of contracting a pairs of equally oriented edges need to be checked.

6.3 Giving a sign to Edmonds's algorithm

In Section 4.1 we described Edmonds's (1965) algorithm. His algorithm can be used to efficiently compute another perfect matching. However, the other perfect matching can have either sign. We would like to find a perfect matching of opposite sign. Our overall idea is to build an augmenting path which gives us a sign-switching cycle. For this we extend Edmonds's approach by giving vertices of the augmenting path states. These states correspond to the orientation of previous edges in the augmenting path. Hence we exploit the Euler tour as required by Lemma 6.1.

We next describe how to alter the breadth-first search used by Edmonds to incorporate a state for each vertex and certain even-length cycles. Start from a perfect matching M, assume this has positive sign and delete any edge $e = (u, v)$ of the $n/2$ matched edges. This creates two free vertices u and v that are not covered by M. Our goal is to find an augmenting path p starting from v to u s.t. $p \cup e$ is a sign-switching cycle. We allow the breadth-first search to revisit vertices in order to update their state. For this we need to keep a list of parent pointers for already visited vertices.

A vertex added with an unmatched edge has one of three states from $\{-, 0, +\}$ with the following interpretation. The "-" ("+") *state* of a vertex indicates that the path leading from the start vertex to this vertex has an odd (even) number of oppositely oriented pairs of edges. For example starting from v, since the deleted edge (u, v) is oriented from u to v set the state of the next vertex w of the newly added edge (v, w) to "+" if the edge is directed from v to w and otherwise to "-"; see the left graph of Figure 6.7 for an example. After the initial edge is added, the state of the next vertex depends on the state of previously added vertex, we often refer to this as the *parent* vertex. Note that the start vertex does not have a parent. We allow for multiple parents and amend the original breadth-first search such that we permit revisiting certain vertices which we explain below.

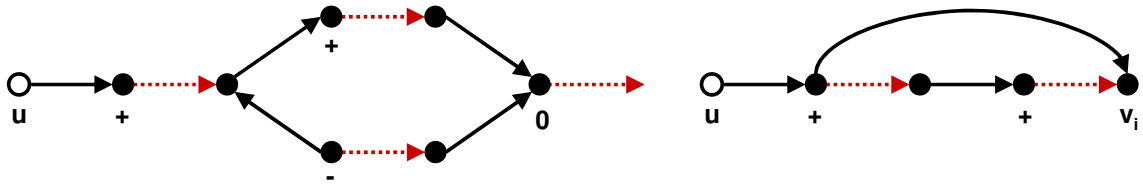


Figure 6.7 Two cases for a "0" state vertex.

A vertex receives state "0" if it should receive both state "+" and "-" but only if the path diverged at a common parent node and then meets again at the corresponding "0" vertex; see the left graph of Figure 6.7. This can be quickly checked by backtracking through the search tree using the parent pointers to find the last common parent.

We do not continue an augmenting path at an already updated vertex of odd distance to the start vertex which additionally belongs to the path from the start vertex. When a path is revisited by its tail we often refer this as *hitting* the path. Although this might create a sign-switching cycle, we would find this cycle if we started the algorithm at a vertex that is adjacent to a matched edge from that cycle, and thus neglect this cycle in general, see the right graph of Figure 6.7. On the left of Figure 6.7, after adding the first matched edge from u, both neighbouring black edges are added, one receiving a plus and one a minus. After another step the two paths meet and since the parents of the new vertex have a different state it receives state "0".

The idea of this approach is that after a "0" vertex has been found the remaining problem is to find any augmenting path to the goal node. Depending on the number of switches of the remaining path we can then take one of the paths to the "0" vertex such that the entire path has the desired sign.

There are two problems with this approach. First, the paths can be non-simple. This occurs when adding a non-matched edge such that this produces an odd cycle, a blossom in Edmonds's sense. Note that this can only occur when we hit a path at a node with even distance to the start vertex. Edmonds's solution was to contract a blossom to a super vertex b^*. Suppose we find a blossom and contract it the usual way. Lemma 6.4 shows that after the contraction the Euler property of the graph is still satisfied.

Lemma 6.4 *Let B be a blossom of an Euler graph G. Then the graph G' obtained by contracting B is still Eulerian.*

Proof: Let the blossom be given by $B = \{b_1, \ldots, b_k\}$. Let the *in* and *out-degree* of each vertex b_i be $d^-(b_i)$ and $d^+(b_i)$, respectively. Before the contraction the Euler property implies

$$\sum_{i=1}^{k} d^-(b_i) = \sum_{i=1}^{k} d^+(b_i). \tag{6.3}$$

Edges that are between a vertex from the blossom and the vertices from the remaining graph are not contracted. Edges connecting two vertices from B add one to the in-degree sum and one to the out-degree sum. Contracting them does not change the equality (6.3) and hence the Euler property of G' still holds. □

The next step is to assign the outgoing edges of b^* the correct states. This depends again on the parent vertices. Note that each vertex of the blossom receives a state according to how the blossom is traversed, either in clockwise or counter clockwise direction. So far we have only modified Edmonds's algorithm slightly to keep track of the signs. We need to contract the blossoms to keep the path simple. However, by contracting the blossoms we also contract the edges that are inside the blossom. Edmonds shows that this is sufficient because for the non signed version only one path around the blossom needs to be found, either clockwise or counter clockwise. We, however, need to to consider all paths, and not only on the periphery of the blossom. In the case that an initial blossom is contracted its super vertex b^* can again be part of a larger blossom and there can be exponentially many path inside the last super vertex. All of these we have to consider. So in order to keep the path simple we need to contract a blossom, but this can eliminate the path we are looking for.

The second problem is that the path from a "0" state vertex to the goal vertex might hit the path from the start vertex to the "0" vertex. This means that we need to update the state of the "0" vertex because we do not necessarily have a choice anymore, in case there were only two paths to the "0" vertex, as described in Figure 6.8.

Our approach to keep track of the number of switches of a path leading to a vertex does not directly solve the problem, as explained above. We might not find the correct path, out of the exponentially many, that we need to find unless we investigate each possible path. This is not feasible for a polynomial algorithm. Hence, the approach to keep track of sign as described above rests on the following two assumptions:

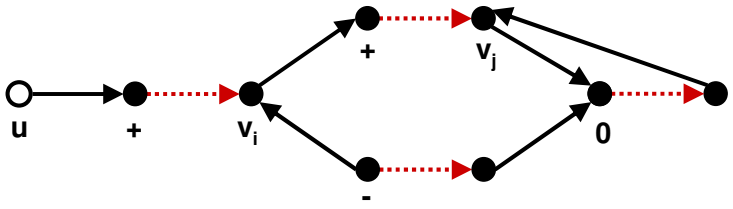


Figure 6.8 One branch of the sign neutral path is revisited, changing the status of the "0" vertex. This forms a blossom at v_j which is contracted into b_1^*, which then creates another blossom starting at v_i. This destroys the "0" state because part of the path is needed to construct the augmenting path.

1. There exists a sign changing augmenting path from u to v that does not traverse the edges inside a blossom.
2. There exists an augmenting path from a "0" vertex which does not hit the original path.

Chapter 7

Conclusion and Open Problems

7.1 Recent PSPACE-completeness results

The complexity class PSPACE is the set of decision problem that are solved by a deterministic Turing machine requiring a polynomial amount of space of the input size n. The "deterministic" part of the definition was proven not be a restriction as shown in Savitch's theorem (1970), which states that this complexity class is not enlarged by allowing for a nondeterministic Turing machine. PSPACE-completeness is defined analogously to NP or PPAD-completeness. Note that the relation $PPAD \subseteq NP \subseteq PSPACE$ holds.

The PPAD-complete problem END OF THE LINE, described formally in Section 2.2, is given by a directed graph G with in and out-degree of at most one and a vertex of degree one, find another vertex of degree one. That is, in a graph consisting of paths, cycles and isolated vertices, given a special source called 0^n, find any another sink or source. By the construction of the graph, when following the path starting from 0^n we find exactly one sink at the end of this line. This problem is called OTHER END OF THIS LINE, referred to as OEOTL. The difference between "any" sink or source or exactly the sink at the end of this line means that the complexity of the problem changes from PPAD-complete to PSPACE-complete, see Papadimitriou (1994).

Goldberg, Papadimitriou, and Savani (2011) have recently shown that a number of solution methods for finding a Nash equilibrium are PSPACE-complete to implement. They proved their results by giving a polynomial-time reduction from OEOTL. These include the Harsanyi-Selten process and other homotopy methods such as the Herings-

van den Elzen, Herings-Peeters and the van den Elzen-Talman algorithms as well as the Lemke-Howson algorithm.

The *Harsanyi-Selten process* finds a unique equilibrium, for most games G_1, by starting to play a simple game G_0 that admits a unique equilibrium and which is easy to compute. Stepwise, from $t = 0$, the simple game is changed more and more towards the original game G_1 by letting t converge towards one, where $G_t = (1-t)G_0 + tG_1$.

Homotopy methods have a similar flavour, see Herings and Peeters (2010) for a recent survey. A transformation between two functions is called a *homotopy* if they can be continuously transformed into each other. In algorithmic game theory a homotopy method connects two fixpoint problems by a continuous transformation, below there is an example for a linear homotopy. Start with a Brouwer function F_0 that is easy to solve and admits a single fixpoint, and linearly change the starting problem to F_1 by considering the intermediate Brouwer functions $F_t = (1-t)F_0 + tF_1$ along the way from t equals zero to one. Intuitively, mapping the fixpoints of F_t for $t \in [0,1]$ gives a path of fixpoints connecting the initial one with the one we are interested in computing.

By expressing the LH-algorithm as a homotopy method Goldberg, Papadimitriou, and Savani (2011) strengthen the exponential runtime result of Savani and von Stengel (2006) and show that it is PSPACE-complete to implement the LH-algorithm. They start with OEOTL and embed it in a game in such a way that dropping any initial vertex of the artificial equilibrium results in a Nash equilibrium that corresponds to the solution of the OEOTL problem. In other words, any solution of the LH-algorithm is a solution of the original OEOTL problem, and when solving a bimatrix game with the LH-algorithm we are basically solving a PSPACE-complete problem. Furthermore, this means that it is PSPACE-complete to compute any equilibrium reached by the LH-algorithm by any method.

7.2 Open problems

Section 7.1 described the recent result of Goldberg, Papadimitriou, and Savani (2011) which states that finding a Nash equilibrium that is computed with the LH-algorithm is PSPACE-complete. An interesting question is what implication this result has for signed perfect matchings in Euler graphs and room partition in d-oiks.

The starting point of the LH-algorithm is the artificial pure strategy equilibrium, which has positive sign, see Section 3.4, and thus all equilibria at the other end of the pivoting paths must have negative sign. These are exactly some of the equilibria we are searching for when considering signed perfect matchings as Nash equilibria in Gale games, as they correspond to matchings with negative sign.[1] Goldberg, Papadimitriou, and Savani (2011) use a general game, which is not derived from a dual-cyclic polytope, in which they embed OEOTL. In particular, if their reduction could be achieved with Morris games then this would imply that finding the other completely labeled vertex at the end of the pivot path on the best response polytope is also PSPACE-complete. Thus, computing the other perfect matching of a different sign at the end of the pivot path on the corresponding Euler graph would then also be PSPACE-complete.

A Morris graph only has one other perfect matching than M; see Proposition 4.5. Clearly, we can find this perfect matching, M' in polynomial time. Because this is the only other perfect matching, its equivalent Nash equilibrium is at the other end of the LH-path when we are considering the polytope setup of the graph. This suggests that the embedding of OEOTL cannot be made with games build by Morris's construction unless $P = PSPACE$ and NL, the complexity class containing problems of nondeterministic logarithmic-space, is a proper subset of P. This implication would follow from the tower of class inclusions between complexity classes (Papadimitriou (1994a), Chapter 7): $NL \subseteq P \subseteq NP \subseteq PSPACE$, which requires that at least one of these inclusions is strict. Another indication that the embedding cannot be constructed with Morris's construction is that there exist shortcuts to the pivot path as shown in Section 4.4.

Allowing for a larger class of games defined by a general Euler graph, not necessarily the Morris labels, asks whether this set of Gale games is "rich" enough to encode the OEOTL problem. This would imply that finding a Gale string at the other end of the pivot path would be PSPACE-complete as well, an interesting open question. Note that although we have shown that for planar Euler graphs which admit a Pfaffian orientation we can find another signed matching efficiently, this does not mean that this is the matching that is at the end of the pivot path. The same implication holds when we neglect the Euler tour to find another matching.

[1] In fact, we consider a larger set of negative signed equilibria as they must not be connected to the artificial equilibrium via a pivot path.

If we only want to compute some other signed perfect matching then this is unlikely to be a PPAD-complete problem. We are only looking to find solutions that correspond to half of the solution space, that is we are only looking for, say, a sink. If this problem is hard then we conjecture that it is PPADS-complete, where the extra "S" stands for "signed" and restricts the solution space to only sinks, see Yannakakis (2009) for a recent survey. From the definition it follows that $PPAD \subseteq PPADS$ but whether this relationship holds with equality is an interesting open question. While it is believed that PPAD-complete problems are not tractable, they are thought to be simpler than NP-complete problems. However, the exact relationship between these two classes remains an open question.

Settling the complexity of the total function problem ANOTHER SIGNED PERFECT MATCHING for a non-bipartite, non-planar Euler graph is an open question. We conjecture that this problem is polynomial-time solvable for general directed Euler graphs with a balanced orientation. For directed Eulerian planar graphs, we showed that we can find a different signed perfect matching by using the fact that the number of perfect matchings of a planar graph is efficiently computed via a Pfaffian orientation and because we know that the original graph has matchings of both signs. If we can extend the class of Euler graphs which admit a Pfaffian orientation then this approach can be used to solve ANOTHER SIGNED PERFECT MATCHING for this class efficiently. Note that deciding whether a graph is Pfaffian is so far only polynomially time solvable for the bipartite case; in fact this problem is in co-NP (Vazirani and Yannakakis (1989)). A better characterisation or classification for non-bipartite graphs is needed to understand when a graph admits a Pfaffian orientation.

Edmonds (2007), Edmonds, Gaubert and Gurvich (2009) and Edmonds and Sanitá (2010) give an undirected path driven argument, via the exchange algorithm, which connects two room partitions in a d-dimensional oik. This argument shows that finding another room partition is a member of the complexity class PPA. We gave an example in Figure 3.6 where the exchange graph is the complete graph K_4, making it unlikely that a directional path driven algorithm or argument exists. An open question is whether the function problem ANOTHER ROOM PARTITION is PPA-complete. One PPA-complete problem is a Sperner lemma for non-orientable 3-manifolds; see Gringi (2001). The total function problem CIRCUIT MAZE is also conjectured to be PPA-complete (Goldberg

(2011)). More knowledge on the relation between the two complexity classes PPA and PPAD is needed. An actual separation between the two classes would imply $P \neq NP$; see Gringi (2001).

Another question relating to the exchange algorithm in the oik setup is to classify its complexity. Our conjecture is that it is at least as hard, that is PSPACE-complete, as the LH-algorithm, however because of the lack of direction one needs to additionally keep track of the history of the previous step. Although the path can be exponentially long, we do not need to keep track of the entire path to make the next pivot.

To my knowledge, Edmonds and Sanita (2010) do not show that the exchange algorithm is exponential for every choice of the initial pivot. For example dropping vertex b in their Figure 1, results in dropping c next, upon which e is double and is dropped to pick up d, which is then dropped to finally pick up label b again, taking only 4 pivots. It is important to show exponential running time for every initial vertex. Otherwise, run the exchange algorithm on a Turing machine with multiple tapes, that is running the pivoting algorithm in *parallel* for each vertex, it terminates after polynomially many steps. Hence the result that is given by Morris's construction in Section 4.4 is the only class of 1-oiks, known to me, where the exchange algorithm takes exponential time for every possibility in the initial vertex. It would be interesting to provide a class of higher dimensional oiks where this property holds as well. Furthermore it would be of interest whether the special edge pairing for the Morris graph, which insured that each edge only needed to be "visited" twice can be generalised? That is, does there exists a "good" edge pairing which results in a short pivot path through the exponentially sized graph of skew-matchings? In case of existence, one needs to determine the complexity of computing a good edge pairing. Both are are open questions.

The hope of investigating the complexity of ANOTHER COMPLETELY LABELED GALE STRING was to show that this problem is PPAD-complete, which is unlikely due to our result showing that ANOTHER COMPLETELY LABELED GALE STRING is solvable in polynomial time. Although this result is negative, we hope that it stimulates research into combinatorially defined problems that are PPAD-complete and imply this property for 2-NASH, leading to a path driven and combinatorial proof of PPAD-completeness.

The computational intractability of 2-NASH poses the question of approximating a Nash equilibrium instead of computing it exactly. Various definitions for approximate

Nash equilibria exist. Generally it is a strategy profile such that no pure strategy of a player played with positive probability is in distance further away than some error $\varepsilon > 0$ from the best response payoff. This error can be additive or multiplicative, both in absolute and relative magnitude in the size of the game, see Daskalakis (2011) for a careful exposition. Daskalakis provides the first hardness result for the relative multiplicative case. He shows that it is PPAD-complete for any, even constant, $\varepsilon \in [0, 1)$ to compute an ε-Nash equilibrium in a bimatrix game. An interesting idea is to extend the approximation approach to the LH-algorithm, see Daskalakis, Mehta, and Papadimitriou (2006). It is desirable to give an error term, associated with the best pair of strategies found along the path, as a function of the number of steps taken.

Index of Symbols

Symbol	Description	Page
$\oplus$	symmetric difference between two sets	39
$[k]$	the set $\{1,...,k\}$	31
$\mathbf{0}$	column vector of all 0s	24
$\mathbf{1}$	column vector of all 1s	24
(A,B)	bimatrix game with payoff matrices for player 1 and 2	18
b^*	super vertex of a contracted blossom	39
d	dimension	24
$det(A)$	determinant of a $n \times n$ matrix	83
E	edge set	38
Et	Euler tour	39
Et_d	Euler tour of Morris graph with d vertices	70
e_i	unit vector, with component i equal to 1, and all others 0	24
F	a facet of a polytope	24
G	graph with edge and vertex set E and V	38
$G(d,n)$	set of Gale even bitstrings of length n with d ones	31
$L(i,d)$	length of pivot path for G_d with missing label i	71
M	perfect matching	39
$\mathcal{M}$	Turing machine	20
$perm(A)$	permanent of a $n \times n$ matrix	82
P, Q	best response polytopes P and Q	25
P^Δ	dual or polar of a polytope P	25
p	simple (directed) path in graph	39
$\pi(i,d)$	the pivot path for G_d with missing label i	71
$\mathcal{R}$	rooms of an oik	49
s	bitstring	31
S_n	symmetric group, n even	43

$sign()$	sign of a permutation, Gale string or a perfect matching	43
T	spanning tree of a graph	86
v	vertex	39
V	vertex set	38
$value()$	value of a permutation	83

References

Balthasar, A. V. (2009), Geometry and equilibria in bimatrix games. PhD Thesis, London School of Economics and Political Science.

Balthasar, A. V., and B. von Stengel (2010), Index and uniqueness of symmetric equilibria. In preparation.

Berge, C. (1957), Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America* **43**, 842–844.

Casetti, M. M., J. Merschen, and B. von Stengel (2010), Finding Gale strings. *Electronic Notes in Discrete Mathematics* **36**, 1065–1072

Cayley, A. (1852), On the theory of permutants. *Cambridge and Dublin Mathematical Journal* **VII**, 40-51.

Chen, X., and X. Deng (2006), Settling the complexity of 2-player Nash equilibrium. *Proc. 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 261–272.

Conitzer, V., and T. Sandholm (2003), Complexity results about Nash equilibria. *International Joint Conference on Artificial Intelligence*, 765–771.

Dantzig, G. B. (1963), *Linear Programming and Extensions*. Princeton University Press, Princeton.

Daskalakis, C. (2011), On the complexity of approximating a Nash equilibrium. *Invited to Transactions on Algorithms (Special Issue for SODA 2011)*, 1498–1517.

Daskalakis, C., P. W. Goldberg, and C. H. Papadimitriou (2006), The complexity of computing a Nash equilibrium. *Proc. 38th Annual ACM Symposium on Theory of Computing (STOC)*, 71–87.

Daskalakis, C., P. W. Goldberg, and C. H. Papadimitriou (2009), The complexity of computing a Nash equilibrium. *Communications of the ACM* **52 (2)**, 89–97.

Daskalakis, C., A. Mehta, and C. H. Papadimitriou (2006), A note on approximate Nash equilibria. *In Proceedings of Workshop on Internet & Network Economics (WINE)*, 297–306.

Daskalakis, C., and C. H. Papadimitriou (2005), Three-player games are hard. *Electronic Colloquium on Computational Complexity*, Report TR05–139.

Edmonds, J. (1965), Paths, trees, and flowers. *Canadian Journal of Mathematics* **17**, 449–467.

_____ (1967), Systems of distinct representatives and linear algebra. *Journal of Research of the National Bureau of Standards* **71B**, 241–245.

_____ (2007), Euler complexes. *http://www.imada.sdu.dk/ asp/edmonds.pdf*.

Edmonds, J., S. Gaubert, and V. Gurvich (2009), On Scarf and Sperner oiks. RUTCOR *Research Report* **18**, Rutgers University, New Jersey.

Edmonds J. and L. Sanitá (2010), On finding another room-partitioning of the vertices. *Electronic Notes in Discrete Mathematics* **36**, 1257–1264.

Fisher, M. E. (1961), Statistical mechanics of dimers on a plane lattice. *Physical Review* **124**, 1664–1672.

Gale, D. (1963), Neighborly and cyclic polytopes. In: *Convexity*, Proc. Symposia in Pure Mathematics, Vol. 7, ed. V. Klee, American Mathematical Society, Providence, Rhode Island, 225–232.

Garey, M. R., and D. S. Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, CA.

Galil, Z. (1986), Efficient algorithms for finding maximum matching in graphs. *Association for Computing Machinery Computing Surveys* **18**, 23–38.

Gilpin, A., J. Peña, and T. Sandholm (2008), First-order algorithm with $O(ln(1/\varepsilon))$ convergence for ε-equilibrium in two-person zero-sum games. *In Proceedings of the 23rd National Conference on Artificial Intelligence*, 75–82.

Goldberg, P. W. (2011), A survey of PPAD-completeness for computing Nash equilibria. *In Surveys in Combinatorics 2011* eds. Robin Chapman, Cambridge University Press, 51–82.

Goldberg, P. W., C. H. Papadimitriou, and R. Savani (2011), The complexity of the homotopy method, equilibrium selection, and Lemke-Howson solutions. *IEEE Symposium on Foundations of Computer Science (FOCS)*.

Graham, R.L., Groetschel M., and Lovász L. (1996), *Handbook of Combinatorics, Volumes 1 and 2*. Elsevier, Amsterdam, and MIT Press, Cambridge.

Grigni, M. (2001), A Sperner lemma complete for PPA. *Information Processing Letters* **77**, 255–259.

Hierholzer, C. (1873), Ueber die Moeglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechnung zu umfahren. *Mathematische Annalen* **6**, 30–32 .

Kasteleyn, P. W. (1967), Graph theory and crystal physics. *In Graph Theory and Theoretical Physics* eds. by F. Harary, Academic Press, New York, 43–110.

Kintali, S., L. J. Poplawski, R. Rajaraman, R. Sundaram, and S. Teng (2009), Reducibility among fractional stability problems. *Proceedings of Electronic Colloquium on Computational Complexity (ECCC)*, 41–41.

Klee, V., and G. J. Minty (1972), How good is the simplex algorithm? In: *Inequalities, III*, Proc. Third Sympos., UCLA, 1969, ed. O. Shisha, Academic Press, New York, 159–175.

LaPaugh, A., and C. Papadimitriou (1984), The even-path problem for graphs and digraphs. *Networks* **14**, 507–513.

Lemke, C. E., and S. J. Grotzinger (1976), On generalizing Shapleys index theory to labelled pseudomanifolds. *Journal of Mathematical Programming* **10**, 245–262.

Lemke, C. E., and J. T. Howson, Jr. (1964), Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics* **12**, 413–423.

Little, C. H. C. (1975), A characterization of convertible (0, 1)-matrices, *Journal of Combinatorial Theory, Series B,* **18**, 187–208.

Lovász, L., and M. D. Plummer (1986), *Matching Theory*. Akadémiai Kiadó, North Holland - Budapest.

McCuaig, W. (2004), Pólya's permanent problem. *The Electronic Journal of Combinatorics* **11**, #R79.

McCuaig, W., N. Robertson, P. D. Seymour, and R. Thomas (1997), Permanents, Pfaffian orientations, and even directed circuits (Extended abstract), *Proceedings of Symposium on the Theory of Computing*.

McKelvey, R. D., A. M. McLennan, and T. L. Turocy (2010), Gambit: Software Tools for Game Theory, Version 0.2010.09.01. Available at http://www.gambit-project.org.

McLennan, A. M., and R. Tourky (2009), Simple complexity from imitation games. *Games and Economic Behavior* **2**, 683–688.

Megiddo, N., and C. H. Papadimitriou (1991), On total functions, existence theorems and computational complexity (Note). *Theoretical Computer Science* **81**, 317–324.

Mihail, M., and P. Winkler (1992), On the number of Eulerian orientations of a graph. *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, 138–145.

Morris, W. D., Jr. (1994), Lemke paths on simple polytopes. *Mathematics of Operations Research* **19**, 780–789.

Muir, T. (1882), *A Treatise on the Theory of Determinants*. MacMillan and Co., London.

Nash, J. F. (1951), Non-cooperative games. *Annals of Mathematics* **54**, 286–295.

Papadimitriou, C. H. (1994a), *Computational Complexity*. Addison-Wesley, Reading, MA.

Papadimitriou, C. H. (1994b), On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences* **48**, 498–532.

Polya, G. (1913), Aufgabe 424. *Arch. Math. Phys. (3)* **20**, 271.

Robertson, N., P. D. Seymour, and R. Thomas (1999), Permanents, Pfaffian orientations, and even directed circuits. *Annals of Mathematics* **150**, 929–975.

Rote, G. (2001), Division-free algorithms for the determinant and the Pfaffian: algebraic and combinatorial approaches. *In: Computational Discrete Mathematics, in: Lecture Notes in Computer Science* **2122**, eds. H. Alt., Springer-Verlag, Berlin, Heidelberg, 119–135.

Savani, R. (2006), Finding Nash equilibria of bimatrix games. PhD thesis, London School of Economics and Political Science.

Savani, R., and B. von Stengel (2006), Hard-to-solve bimatrix games. *Econometrica* **74**, 397–429.

Savitch, W. J. (1970), Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences* **4**, 177–192,

Scarf, H. E. (1967), The core of an *n* person game. *Econometrica* **35**, 50–69

Shapley, L. S. (1974), A note on the Lemke–Howson algorithm. *Mathematical Programming Study* **1***: Pivoting and Extensions*, 175–189.

von Schemde, A., and B. von Stengel (2008), Strategic characterization of the index of an equilibrium. *Proceedings of the 1st International Symposium on Algorithmic Game Theory (SAGT)*, 242–254.

von Stengel, B. (1999), New maximal numbers of equilibria in bimatrix games. *Discrete and Computational Geometry* **21** , 557–568.

von Stengel, B. (2002), Computing equilibria for two-person games. Chapter 45, *Handbook of Game Theory, Vol. 3*, eds. R. J. Aumann and S. Hart, North-Holland, Amsterdam, 1723–1759.

_____ (2007), Equilibrium computation for two-player games in strategic and extensive form. *Chapter 3, Algorithmic Game Theory*, eds. N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, Cambridge Univ. Press, Cambridge, 53–78.

Thomas, R. (2006), A survey of Pfaffian orientations of graphs. *Proceedings of the International Congress of Mathematicians* **3**, 963–984.

Todd, M. J. (1976), Orientation in complementary pivot algorithms. *Mathematics of Operations Research* **1**, 54–66.

Vazirani, V. V., and M. Yannakakis (1989), Pfaffian orientations, 0-1 permanents, and even cycles in directed graphs. *Discrete Applied Mathematics* **25**, 179–190.

Yannakakis, M. (2009), Equilibria, fixed points, and complexity classes. *Computer Science Review* **3**, 71–85.

Ye, Y. (1997), *Interior Point Algorithms: Theory and Analysis*. John Wiley & Sons, New York.

Ziegler, G. M. (1995), *Lectures on Polytopes*. Springer, New York.