# Finding Gale Strings

Marta M. Casetti, Julian Merschen, Bernhard von Stengel

*Department of Mathematics, London School of Economics, Houghton Street,London WC2A 2AE, U.K.*

**Abstract**

The classical algorithm for finding an Nash Equilibrium, the Lemke-Howson algorithm, exhibits exponential running time for a special class of bimatrix games generated by dual cyclical polytopes. It was thus conjectured that solving these games via the Lemke-Howson algorithm was PPAD-complete, which would give an elegant and traceable proof that NASH is PPAD-complete. For this special class of games the Nash Equilibria can be fully described by Gale strings. We introduce the problems GALE and ANOTHER GALE STRING and show that both are polynomially solvable: this implies that finding all NE for these games can be solved in polynomial time, disproving the above conjecture.

*Keywords:* Nash Equilibria, Lemke-Howson, Gale Evenness Strings, Complexity, Perfect Matching, Polynomial Time Algorithm

## 1. Introduction

After Nash proved the existence of mixed equilibria in finite strategic games in in 1951 [? ] recent research has addressed the complexity of the problem NASH, that is how hard is it to find an Nash equilibria of a bimatrix game. Finding all Nash equilibria in a bimatrix games is NP-complete, see [? ], but only recently it was shown that finding one Nash equilibrium belongs to the interesting complexity class PPAD, see Daskalakis, Goldberg and Papadimitriou [? ] and Chen and Deng [? ]. The complexity class PPAD includes function problems which are known to have a solution such as END OF THE LINE and BROUWER [? ]. Intuitively, it is the class of problems in which, given a source of a directed path,

*Email address:* `m.m.casetti@lse.ac.uk`, `j.merschen@lse.ac.uk`, `stengel@maths.lse.ac.uk` (Marta M. Casetti, Julian Merschen, Bernhard von Stengel)

we are asked to find a sink or a nonstandard source. The PPAD-completeness of NASH wats quite an unexpected result since a NE of a zero-sum game can be efficiently found by solving a minmax problem [? ] via a linear program in polynomial time [? ].

The classical algorithm to find an Nash equilibrium (NE), the Lemke-Howson (LH) algorithm [? ] finds a NE by pivoting along the edges of the best response polytopes, derived from a bimatrix game, until it terminates at a NE. This algorithm computes a NE efficiently in most games but Savani and von Stengel [? ] showed that it displays exponential running time for a special class of games, derived from dual cyclic polytopes. These polytopes can be obtained by taking the convex hull of the points on a moment curve. The resulting best response polytopes can then be fully described by an $n$-string of labels $l$, where each label corresponds to a facet of the polytope. A vertex of the polytope is represented by a bit string $s$ if and only if $s$ fulfills the *Gale evenness condition* (GEC), stating that a maximal substring of only 1's in $s$ must have even length, see [? ]. A one in the bit string indicates whether a vertex is on the facet associated with the label from $l$.

A NE in the bimatrix games corresponds to a vertex in the best response polytope which is *fully labeled*, that is every facet of $l$ is labeled exactly once. As vertices of the polytope can be described by bit strings which fulfill the GEC, we only need to consider finding a fully labeled string corresponding to a NE of the bimatrix game. Bit strings which satisfy the GEC and fully label the corresponding label string are referred to as *Gale strings*. The decision problem GALE answers the questions whether a string of labels has an associated Gale string. Clearly, there exist a Gale string for the string of labels derived from the best response polytopes of a bimatrix game, a NE of the game.

For this subclass of games, instead of finding a NE by pivoting in the best reply polytopes we only need to consider a string of labels and find a NE via a version of the LH algorithm, the Lemke-Howson for Gale (LHG) algorithm. Similarly to the LH algorithm the LHG algorithm starts with a trivial Gale strings, the artificial equilibrium, and finds another distinct Gale string, a NE of the game.

We refer to the problem of finding another Gale string $s$ for the string of labels $l$, provided we are given one Gale string for $l$ as ANOTHER GALE STRING. For the subclass of games derived from dual cyclic polytopes the LHG-algorithm displays exponential running time for some special examples [? ]. So finding another Gale string can take exponentially many steps via the LHG-algorithm. Due to the exponential example it was conjectured that the problem of ANOTHER GALE STRING is PPAD-complete. This would in turn have provided an elegant and traceable proof stating that NASH is PPAD-complete, proved by [**?** ][**?** ] by reducing BROUWER to an approximate version of NASH.

In this paper we prove that GALE and ANOTHER GALE STRING are polynomially decidable and solvable. We prove the fist part by reducing GALE to PERFECT MATCHING, which was shown to be in P by Edmonds [**?** ]. The idea of the reduction is to translate a string of labels $l$ into a graph $G$: the vertices in the graph correspond exactly to the labels in $l$ and an edge is placed between two vertices if the labels are neighbors in $l$. This construction of $G$ assures that the graph has a perfect matching if and only if there exists a Gale string $s$ corresponding to $l$. Using Edmonds's beautiful "Paths, Trees, and Flowers" algorithm [**?** ] we can find one perfect matching in $G$ and thus one Gale string, in case the graph omits one. By extending the construction of the graph to multigraphs we show ANOTHER GALE STRING is polynomially solvable, making it a member of P. This implies that computing all NE in bimatrix games generated by dual cyclic polytopes can solved in polynomial time. So far these are the only known games where the LH algorithm exhibits exponential running time.

Furthermore, we show via the LHG algorithm ANOTHER GALE STRING is a member of PPAD, although at this point is very unlikely to be PPAD-complete – since it would imply that PPAD = P.

## 2. Gale Strings and Nash Equilibria

In this section we introduce the problems GALE and ANOTHER GALE STRING and connect them to algorithmic game theory. Given an $n$-string of labels, $l \in \{1,\ldots,d\}^n$ where $d$ is even and $d < n$, an associated bit string $s \in \{0,1\}^n$ is called a *Gale string* (or Gale bit string) if it satisfies:

1. *Gale Evenness Condition* (GEC): all maximal substrings of 1's in $s$ with indices considered modulo $n$ have even length;
2. *Fully Labeled Condition* (FLC): $\{l(i)|s(i) = 1\} = \{1,\ldots,d\}$.

where we set $kn \mod n = n$ for all integers $k$.

The first condition states that a substring of $s$ of the form $01\cdots10$ has even length, so that 0110, 011110, etc., are allowed, but not 010, 01110, and so on. In other words, a maximal substring of 1's, called a *run*, must have an even number of 1's. By considering the indices of $s$ modulo $n$ we allow odd maximal substrings of 1's at both ends of $s$ by "gluing" them together to form an even substring. The second condition states that each label is marked with exactly one 1 in the associated bit string $s$, thus fully labeling $l$. Let $G(l)$ be the set of Gale strings corresponding to an $n$-string of labels $l$.

**Example** For the string of labels $l = 1123143$ the set of Gale strings is $G(l) = \{0110011, 0011110\}$. The string of labels $l = 123432$ has associated Gale strings $s_1 = 111100$, $s_2 = 110110$, $s_3 = 100111$ and $s_4 = 101101$ (note the "glued ends" in the last two). Conversely, for the string of labels $l = 121314$, $G(l)$ is empty.

A natural question is: given an $n$-string of labels $l$ does it have an associated Gale string? That is, is $G(l)$ non empty? We refer to this as the GALE decision problem, defined as follows.

GALE
**Input**: $n$-string $l \in \{1, \dots, d\}^n$ where $d$ is even and $d < n$.
**Question**: Does $l$ have a Gale string $s$?

A second question, motivated from algorithmic game theory, is, given a string of labels $l$ and Gale bit string $s$, how hard is it to find another Gale bit string?

ANOTHER GALE STRING
**Input**: $n$-string $l \in \{1, \dots, d\}^n$ where $d$ is even and $d \leq n$, a Gale bit string $s$ associated with $l$.
**Output**: Another Gale string $s' \neq s$ associated with $l$.

Finding another Gale string is motivated by computing an NE via the classic Lemke-Howson (LH). This algorithm is efficient for most bimatrix games, but for a special class of games derived from dual cyclic polytopes it needs exponentially many steps to find an NE. These games can be described by $n$-strings of labels and the problem of finding a NE is exactly the problem of ANOTHER GALE STRING. We now describe the relationship between these games and Gale strings, for more detail see [**?** ].

The special class of bimatrix games with $d$ strategies for each player are derived from from dual cyclic polytopes in dimension $d$ with $2d$ facets. These polytopes can be obtained by taking the convex hull of $2d$ points on the moment curve. Re-interpreting and resizing this polytope gives the dual cyclic polytope for player I

$$P'' = \left\{ z \in \mathbb{R}^d | c_i^T z \leq 1 \leq i \leq 2d \right\},$$

where $c_i \in \mathbb{R}^d$. The vertices of $P''$ are exactly bit strings which correspond to the $2d$-string of labels $l$, where the labels are taken from the set $\{1,...,d\}$, and which satisfy the GEC. A bit $s(i)$ equal to 1 indicated that the vertex is on the $l(i)$th facet of the polytope. The best reply polytope $P$ for player I is derived by a suitable affine transformation which rearranges the labels in $P''$ such that the first $d$ labels correspond to the pure strategies of player I. In similar fashion we derive the best reply polytope $Q$ for player II where the second $d$ labels correspond to the pure strategies of player II. The other labels of the two strings describe the game. Concatenating the two label strings we obtain a $4d$-string of labels $l$, where the first $d$ and last $d$ labels correspond to the pure strategies of player I and II, respectively.

Nash equilibria of a bimatrix games are exactly Gale strings of $l$, apart from the bit string with $d$ 1's on both ends. This special bit string corresponds to a trivial "artificial" Nash equilibrium which assigns zero probability to all pure strategies of both players. This Gale string acts as the starting point of the LH and and Lemke-Howson for Gale (LHG) algorithm. The problem ANOTHER GALE STRING is therefore motivated from the problem of finding a non-trivial NE from the artificial Gale string. In the next section we describe the Lemke-Howson for Gale pivoting algorithm and analyze its complexity.

## 3. The Lemke-Howson for Gale Algorithm

In this section we will prove that ANOTHER GALE STRING is a problem in the complexity class PPAD (Polynomial Parity Argument, Directed version), first introduced by Papadimitriou in 1994 [**?** ] together with the closely related class PPA (Polynomial Parity Argument). We refer to Papadimitriou's article for the original formal definition of PPA and PPAD: for our scope we will consider PPAD as the class of problems reducible to
END OF THE LINE
**Input**: Two circuits $S$ and $P$ with $n$ input bits and $n$ output bits such that $P(0^n) = 0^n \neq S(0^n)$.
**Output**: An input $x$ such that $P(S(x)) \neq x$ or $S(P(x)) \neq x \neq 0^n$

that is, the class of problems reducible to "given a directed graph in which each node has indegree and outdegree of at most one and an unbalanced node, find another unbalanced node". PPA will then be the class of problems reducible to "given a graph of degree at most 2 and a node with odd degree, find another node of odd degree". Note that although PPAD $\subseteq$ PPA it is not known whether this relationship holds with equality.

Our first theorem states

**Theorem 3.1.** ANOTHER GALE STRING *is in PPA.*

To prove the theorem we will introduce an algorithm, a variation on the Lemke-Howson algorithm that will pivot from one Gale string (that will correspond to a Nash equilibrium) to another. We start by explaining what we mean by a pivot. Suppose we are given an $n$-string $l$, a corresponding Gale string $s$ and a position $i \in \{1,\dots,n\}$ which satisfies $s(i) = 1$. We first set $s(i) = 0$, i.e. we *drop* the label $l(i)$. By the GEC, either to the left or the right of $s(i)$, there is an odd run of 1's. We choose the side of $s(i)$ adjacent to where the odd run is and invert the first 0 to the left or right of the run, hence *picking* up a new label. We refer to this as *pivoting* the label $l(i)$, as we pivot the 1 associated with $l(i)$ across the odd run. Clearly the GEC is still satisfied after a pivot, however $l$ may not be fully labeled anymore as one label may be duplicate and thus one label missing. We call a bit string which satisfies the GEC and where exactly one label is duplicate an *almost Gale* string. That is, an almost Gale string $s$ satisfies the GEC; furthermore the number of 1's in $s$ is equal to $d$ and $|\{l(i)|s(i) = 1\}| = d-1$.

A pivot for an almost Gale string is defined similarly as dropping a duplicate label, assuring that the resulting string is again an almost Gale string or Gale string. In a Gale string we may drop any label $l(i)$ which satisfies $s(i) = 1$. We refer to these as *free* labels.

Note that a pivot is reversible. Suppose we just picked up a label $l(i)$ by dropping a label $l(j)$. Clearly, $s(i)$ must be at the end of a run and $s(j)$ is equal to 0 and adjacent to the opposite side of the run where $l(i)$ was picked up. By pivoting $l(i)$ we must pick $l(j)$, making the pivot reversible. We now describe the Lemke-Howson for Gale algorithm,

As there are only a finite number of possible bit strings for each label string and if cycling is not possible the algorithm must terminate by finding another Gale string in a finite number of steps. Cycling is not possible due to the following observations. Suppose the algorithm returns to a bit assignment of $s$ 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 GEC as only one of the adjacent runs of the dropped label is odd. Returning to initial position is only possible by reversing the initial pivot which is not allowed. The only free choice we have is at the beginning of the

**Algorithm 1**: Lemke-Howson for Gale Algorithm

```
input : A $n$-string $l \in \{1,\dots,d\}^n$ where $d$ is even and $d < n$, a Gale string
        $s$ associated with $l$.
output: A Gale string $s' \neq s$ associated with $l$.
1 pivot a free label of s;
2 while (s is an almost Gale string) do
3     pivot the duplicate label, not picked up by the previous pivot;
4 end
5 return $s' = s$
```

algorithm where we drop one free label. From then on the process of the algorithm is uniquely determined, thus terminating in a finite number of steps at another Gale string. As this initial choice is somewhat arbitrary the resulting Gale strings and NE for different first steps can vary.

By analysing the path created by the LHG algorithm we show that ANOTHER GALE STRING is a member of PPA.

The steps taken by the LHG algorithm generate an undirected path which starts at the initial Gale string and which ends at another Gale string, thus making LHG a member of PPA.

We will now refine theorem 3.1 to show that

**Theorem 3.2.** ANOTHER GALE STRING *is in PPAD.*

In order to prove that LHG is a member of PPAD we need to add an orientation, so that a directed path is generated by the algorithm. That is at any point of the algorithm we should be able to decide only from considering the current configuration of $s$ what the previous pivot was and which of the duplicate labels we have to pivot next. We now describe how to embed an orientation for each configuration.

Consider the substring $l'$ of $l$ where a label $l(i)$ is present in $l'$ if $s(i) = 1$, e.g. for $l = 123432$ and $s = 011110$ the substring $l'$ is 2343. If $l'$ is obtained from a Gale string, the *sign* of $l$ is positive (negative) if the number of inversions in $l'$ is even (odd). This describes the orientation of the input and thus the start of the algorithm.

In case $s$ is almost Gale (we are somewhere in the middle of the path) $l'$ has a duplicate label. To obtain the sign of $l'$ for this case, let $l'_1$ and $l'_2$ be obtained from $l'$ by substituting the missing label for the different duplicate labels in $l'_1$

and $l'_2$, respectively. The sign of $l'_1$ and $l'_2$ is positive (negative) if the number of inversions in $l'_1$ and $l'_2$ is even (odd). It is important to note that $l'_1$ and $l'_2$ have always opposite orientations, since each can be obtained from the other by exchanging two elements – the duplicate label and the missing label of $l$.

**Example** Given the string of labels $l = 123432$ the Gale bit string $s_1 = 111100$ has positive sign as $l'_1 = 1234$ has no inversions. For $s_2 = 110110$ and $s_3 = 100111$, $l$ has negative sign since there is one inversion in $l'_2 = 1243$ and three inversions in $l'_3 = 1432$. Finally, for $s_4 = 101101$, $l$ has positive sign since there are two inversions in $l'_4 = 1342$. For the almost Gale string $s = 011110$ the associated string of labels is $l' = 2342$. This has positive or negative sign depending on whether we consider $l'_1 = 2341$ or $l'_2 = 1342$.

A pivot from a Gale or an almost-Gale string, where the "old" duplicate label is substituted by the missing label, changes the sign of the new string. This is because the pivoting 1 will "jump" over an odd number of 1's. We modify the LHG algorithm, see algorithm **??**, to incorporate a direction for the resulting path, visible at each step, resulting in a change of sign between each step. Note that our orientation of the path recalls the orientation of Todd (see [**?** ]) for primoids and duoids, which in turn is a generalization of Shapley's (see [**?** ]) "sense of direction" on the paths given by the LH algorithm.

**Algorithm 2**: LHG Directed Algorithm

```
input : A n-string $l \in \{1,\dots,d\}^n$ where d is even and $d < n$, a Gale string
        s associated with l.
output: A Gale string $s' \neq s$ associated with l.

1 pivot a free label of s;
2 while (s is an almost Gale string) do
3     if ($sign(l'_1) = sign(s)$) then
4         pivot the label in $l'_1$, substituted by the missing label;
5     else
6         pivot the other duplicate label;
7 end
8 return $s' = s$
```

**Example** We consider again $l = 123432$. If we start from $s = 111100$, with positive sign, by pivoting $l(1) = 1$ we obtain $s = 011110$. Now we have two possible signs: negative for $l'_1 = 2341$ and positive for $l'_2 = 2143$. For the former the duplicate label is the “new” label, for the latter the “old” label. We pivot the old label, $l(3)$, which yields the Gale string $s = 110110$ (negative sign) and the algorithm terminates with a solution.

In general, starting with a Gale string with positive sign we obtain the following path:

$$\begin{tikzcd}
+\,(\text{Gale string}) \arrow[r, "first\ pivot"] & -\,(\text{almost-Gale, “new label”}) \arrow[dl, "orientation"] \\
+\,(\text{almost-Gale, “old label”}) \arrow[r, "pivot"] & \dots \arrow[dl, "orientation"] \\
\dots \arrow[r, "pivot"] & -\,(\text{another Gale})
\end{tikzcd}$$

When starting with a string with negative sign the signs are simply the opposite and equivalently, the direction of the path is inverted. Therefore, the endpoints of the algorithm always have opposite orientation. As already stated by Shapley [**?** ], since the starting point of the LH algorithm has positive orientation, we know that using the LH algorithm we will always find Nash equilibria of negative sign.

Note that, in general, if we construct a graph with possible Gale strings as vertices and connect the Gale string that can be derived from each other, following the LHG algorithm this graph is bipartite into strings of positive and negative sign. This shows that the number of NE found by the LHG-algorithm is always odd, with one “negative sign” equilibrium more – see also [**?** ].

**Example** Let us consider $l = 12342314$. The possible Gale strings are derived from each other by dropping different labels as follows:

$$\begin{tikzcd}
11110000 \arrow[r, leftrightarrow, "1"] \arrow[dr, leftrightarrow] \arrow[ddr, leftrightarrow] & 00011110 \arrow[dl, leftrightarrow, "2,4"'] \arrow[ddl, leftrightarrow] \\
01100011 \arrow[r, leftrightarrow, "2"] \arrow[dr, leftrightarrow, "4"] & 11100001 \\
00001111 \arrow[r, leftrightarrow, "1,2"'] & 10001101
\end{tikzcd}$$

Note that the graph is a bipartite graph – but it is not a complete bipartite graph: to reach 1110001 from 0000111 we must apply the algorithm three times. To our knowledge it is an open question if the graph has to be connected.

This brings us back to the connection with Game Theory: a result of PPAD-completeness on ANOTHER GALE STRING would yield a traceable proof via the LHG algorithm to the result that NASH is PPAD-complete, whereas [**?** ] and [**?** ] use an approximated version of the Nash equilibrium. A good reason why finding a second Gale string was believed to be PPAD-complete arised from a bimatrix game, generated from a dual cyclic polytopes, where the LHG-algorithm takes exponentially many steps to find an NE, see [**?** ]. The question remains open but, as we will see, it is very unlikely that the answer is positive. In the next chapter we show that GALE and ANOTHER GALE STRING are both members of P.

## 4. The Complexity of GALE and FINDING ANOTHER GALE STRING

We show that GALE is decidable in polynomial time, and thus a member of P, via a reduction to PERFECT MATCHING, the decision problem defined as

PERFECT MATCHING
**Input**: Graph $G = (V,E)$.
**Question**: Does there exist a set $M \subseteq E$ of pairwise non-adjacent edges such every vertex $v \in V$ is incident to exactly one edge of $M$?

Clearly, a graph can only have a perfect matching if the number of vertices is even. Edmonds [**?** ] first introduced a polynomial algorithm which finds a maximal matching in a graph, provided it has one, in polynomial time. We can use this algorithm to decide PERFECT MATCHING in polynomial time. Before we give a polynomial reduction from GALE to PERFECT MATCHING, which shows that GALE is polynomially decidable, let us note two interesting properties of Gale strings.

**Lemma 4.1.** *Let $l = l(1)l(2)\cdots l(n)$ be a string as in an arbitrary instance of* GALE *and let $l(i) = l(i+1)$ for some $i \mod n$. Let $l'$ be the $(n-1)$-string obtained by deleting $l(i)$, that is $l' = l(1)\cdots l(i-1)l(i+1)\cdots l(n)$. Then there is a a one-to-one correspondence between the Gale strings associated with $l$ and those associated with $l'$.*

**Proof** Assume w.l.o.g. that $i = 1$, so that $l' = l(2)\cdots l(n)$. Let $s = s(1)\cdots s(n)$ and $s' = s'(1)\cdots s'(n-1)$ be the Gale strings associated with $l$ and $l'$ respectively.

Suppose we know $s$. Note that only one $s(i)$ for $i = 1,2$ is equal to 1. If either $s(1)$ or $s(2)$ is equal to 1 take $s' = 1s(3)\cdots s(n)$. If both are equal to 0 take

$s' = 0s(3)\cdots s(n)$. In both cases, the label $l(1) = l(2)$ is still labeled – by $s'(1)$ in the former and elsewhere in latter. As we eliminate only a 0, the GEC is still satisfied.

Conversely, suppose we know $s'$. If $s'(1)$ is equal to 0, take $s = 0s'(1)\cdots s'(n-1) = 00s'(2)\cdots s'(n-1)$. The GEC still holds, because we only add a zero next to another zero. If $s'(1)$ is equal to 1, it is part of an even run of ones; set $s = 01s'(2)\cdots s'(n-1)$ or $s = 10s'(2)\cdots s'(n-1)$ according to which one retains the GEC. This is always possible because we "break" the even run in two even runs by introducing a 0. In both cases it is guaranteed that $l(1) = l(2)$ is labeled exactly once, as in $s'$. This concludes the proof of the lemma.

From this point forward we will only consider strings of labels $l$ with $l(i \mod n) \neq l((i+1) \mod n)$. Next we show that can disregards strings of labels with its ends "glued together" as long as we consider an augmented string, obtained by adding a constant number of labels to the original string.

**Proposition 4.2.** *The Gale strings associated with the string $l = l(1)\cdots l(n)$ with indices taken modulo n are in one-to-one correspondence with the ones associated with $l' = al(1)\cdots l(n)bxabxy$ with indices taken* not *modulo n, where $a,b,x,y \notin \{1,\ldots,d\}$.*

**Proof** Let $s \in G(l)$ and $s' \in G(l)$. Since $y$ appears only in one position, the corresponding position in the bit string $s'$ must be 1. By parity the bit preceding is also equal to 1. Any other bit in $s'$ corresponding to an occurrence of $x$ in $l'$ must be 0. Again by parity the bits associated with $a$ and $b$, $s'(2)$ and $s'(n+3)$ are both equal to 0 and $s'(n+5)$ and $s'(n+6)$ are both equal to 1, or vice versa. Now, all the strings $s$ in which the ends are not "glued together" are in one-to-one correspondence with all the $s'$ in the former case, whereas the the strings $s$ in which both ends have an odd number of 1's are in one-to-one correspondence with all the $s'$ in the latter. This concludes the proof.

**Example** The string $l = 1234132$ has four corresponding Gale strings: $s_1 = 1111000$, $s_2 = 0111100$, $s_3 = 0001111$, and $s_4 = 1011001$. The corresponding string of labels without wrap-around is $l' = a1234132bxabxy$ with Gale strings $s'_1 = 0\mathbf{1111000}001111$, $s'_2 = 0\mathbf{0111100}001111$, $s'_3 = 0\mathbf{0001111}001111$, and $s'_4 = 1\mathbf{1011001}100011$.

We now state our theorem.

**Theorem 4.3.** GALE *is polynomially decidable.*

**Proof** We transform GALE to PERFECT MATCHING as follows. Let $l$ be the $n$-string in the set $\{1,\dots,d\}^n$ as in an arbitrary instance of GALE. We must construct the graph $G=(V,E)$ such that $G$ has a perfect matching if and only if $l$ has a Gale string. Specify the graph $G=(V,E)$ by setting $V=\{1,\dots,d\}$ and $E=\{(u,v)|u=l(i)$ and $v=l((i+1) \mod n)$ for $i=1,\dots,n\}$, where we set $kn \mod n = n$ for all integer $k$. In other words there exists an edge between two vertices in the graph exactly when they are neighbors somewhere in the string $l$ with ends "glued together". Clearly the graph is polynomial in the size of the instance of GALE.

Now suppose that $G$ has a perfect matching. Then there exist a set of pairwise non-adjacent edges $M \subseteq E$ such every vertex $v \in V$ is incident to exactly one edge of $M$. Given $M$, construct $s$ as follows: if $(u,v) \in M$ then $uv$ are neighboring labels in $l$, by the construction of the graph. As there might be multiple occurrences of $uv$ in $l$, find one such pair and set the corresponding bits in $s$ to 1. As we are not considering strings with substrings $uu$, there are no loops $(u,u)$ in the graph or in the matching and therefore no duplicate labels. Since we set only adjacent pairs of bits to 11, clearly the GEC is satisfied. The string is fully labeled since the matching is perfect and therefore every vertex is incident to exactly one edge in $M$. If the corresponding graph does not have a perfect matching then it is impossible to construct a Gale String $s$. This concludes the proof.

The function problem associated with GALE is

GALE STRING
**Input**: $n$-string $l \in \{1,\dots,d\}^n$ where $d$ is even and $d < n$.
**Output**: An associated Gale string $s$ or $s = 0\cdots0$ if such a string does not exist.

A solution to GALE STRING can be computed in polynomial time from the perfect matching (in case one exists) of the corresponding graph, see the proof of Theorem **??**.

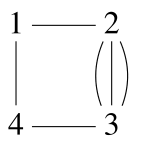
**Corollary 4.4.** GALE STRING *is polynomially solvable.*

Interestingly, while PERFECT MATCHING is in P counting the number of perfect matchings of even a bipartite graph is ♯P-complete [**?** ]. This implies that counting all Gale strings associated with a string of labels is also ♯P-complete.

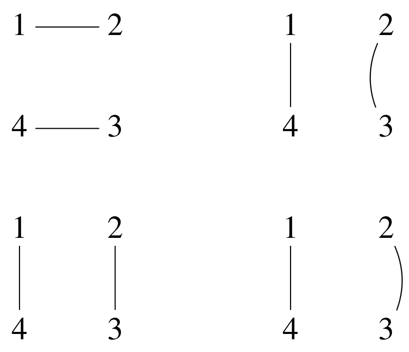
## 5. Finding a second Gale String is in P

We have seen that one approach to find a second Gale string, via the LHG algorithm, exhibits exponential running time for certain games. We now show that a second Gale string can be found in polynomial time. Using the transformation as in Theorem **??** and Edmonds's algorithms, given a string $l$ of labels we can find either at least one element $s \in G(l)$ or that $G(l)$ is empty. In order to find another member of $G(l)$ we need to alter the transformation such that the corresponding graph allows multiple edges if there are repetitions of neighboring labels in $l$.

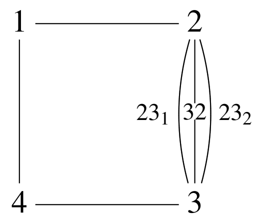
**Example** For $l = 123234$ the corresponding multigraph has three distinct edges between vertex 2 and 3 allowing for four possible distinct perfect matchings. These correspond to the four Gale strings $s_1 = 100111$, $s_2 = 101101$, $s_3 = 111001$ and $s_4 = 110011$. That is, from



we get



Furthermore, we can label each multiple edge $(u,v)$ according to the position of the substring $uv$ in $l$, i.e. $uv_1$ is the first occurrence in $l$.



Note that the multigraph corresponding to $l$ is a Euler graph, that is each vertex has an even degree. Furthermore $l$ gives an Eulerian orientation of its corresponding Euler graph. We now give a polynomial time algorithm for solving the problem

ANOTHER GALE STRING
**Input**: $n$-string $l \in \{1, \dots, d\}^n$ where $d$ is even and $d \leq n$, a Gale string $s$ associated with $l$.
**Output**: Another Gale string $s' \neq s$ associated with $l$.

**Theorem 5.1.** ANOTHER GALE STRING *is polynomially solvable.*

**Proof** Construct a multigraph $G = (V, E)$ associated with $l$ as above, and consider the matching $M = \{m_1, \dots, m_{\frac{d}{2}}\}$ associated with $s$. Consider $G_1 = (V, E \setminus \{m_1\})$, and via the Edmonds's algorithm evaluate if $G_1$ has perfect matching. If so, take $s'$ to be the string associated with this matching. It is a Gale string because the matching is perfect, and it is not equal to $s$ because $m_1$ does not belong to its associated matching. If $G_1$ does not have a perfect matching, consider $G_2 = (V, E \setminus \{m_2\})$, and so on. Since we know that at least two Gale strings exist, the algorithm will stop after at most $\frac{d}{2}$ steps. This concludes the proof.

As finding all NE in bimatrix games generated by dual cyclic polytopes can be solved by Edmonds's algorithm this lemma follows.

**Lemma 5.2.** *Computing all NE in bimatrix games generated by dual cyclic polytopes can solved in polynomial time.*

It is interesting to note that this class of games is the only known class of games for which the LH-algorithm exhibits exponential running time.

[1] C. Xi, X.Deng, (2006). Settling the Complexity of Two-Player Nash Equilibrium. *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS) 2006*, pp 261–272.

[2] C. Daskalakis, P. W. Goldberg and C. H. Papadimitriou, (2006). The complexity of computing a Nash equilibrium. *Annual ACM Symposium on Theory of Computing 2006*, pp. 71–78.

[3] C. Daskalakis, P. W. Goldberg and C. H. Papadimitriou, (2009). The complexity of computing a Nash equilibrium. Commun. ACM 52, pp 89–97.

[4] J. Edmonds (1965), Paths, Trees, and Flowers. Canad. J. Math. 17, pp. 449–467.

[5] D. Gale (1963), Neighborly and cyclic polytopes. In: Convexity, Proc. Symposia in Pure Math., Vol. 7, ed. V. Klee, American Math. Soc., Providence, Rhode Island, pp. 225–232.

[6] I. Gilboa and E. Zemel. Nash and correlated equilibria: Some complexity considerations. Games and Economic Behavior, 1989.

[7] L. G. Khachiyan, (1979). A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20(1), pp. 191–194.

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

[9] J. Nash, (1951). Noncooperative games. *Annals of Mathematics, 54*, pp. 289–295.

[10] C. H. Papadimitriou (1994), On the Complexity of the Parity Argument and Other Inefficient Proofs of Existence. In Journal of Computer and System Sciences 48, pp. 498–532.

[11] R. Savani, B. von Stengel (2006), Hard-to-Solve Bimatrix Games. Econometrica 74, pp. 397–429.

[12] L. S. Shapley (1974), A Note on the Lemke-Howson Algorithm. Mathematical Programming Study, 1, pp. 175–189.

[13] M. J. Todd (1976), Orientation in Complementary Pivot Algorithms. Mathematics of Operations Research, vol. 1, no. 1, pp. 54–66.

[14] L. G. Valiant (1979), The complexity of computing the permanent. Theoretical Computer Science 8, pp. 89–201.

[15] J. von Neumann, (1928). Zur Theorie der Gesellschaftspiele, *Mathematische Annalen*, 100, pp. 295–320.