Exponential lower bounds for history-based simplex pivot rules on abstract cubes

Antonis Thomas*1

1Department of Computer Science, Institute of Theoretical Computer Science, ETH Zürich, 8092 Zürich, Switzerland

Abstract

The behavior of the simplex algorithm is a widely studied subject. Specifically, the question of the existence of a polynomial pivot rule for the simplex algorithm is of major importance. Here, we give exponential lower bounds for three history-based pivot rules. Those rules decide their next step based on memory of the past steps. In particular, we study Zadeh's least entered rule, Johnson's least-recently basic rule and Cunningham's least-recently considered (or round-robin) rule. We give exponential lower bounds on Acyclic Unique Sink Orientations (AUSO) of the abstract cube, for all of these pivot rules. For Johnson's rule our bound is the first superpolynomial one in any context; for Zadeh's it is the first one for AUSO. Those two are our main results.

1 Introduction

The existence of a polynomial time pivot rule for the simplex algorithm is a major open problem in the theory of optimization. Most known rules have superpolynomial lower bounds by now. For deterministic rules, in particular, it is the case that many of them admit exponential lower bounds. Klee and Minty with their seminal paper [17], already in 1972, gave an exponential lower bound for Dantzig’s original pivot rule. Their construction has been heavily studied ever since (for example [10],[4]) and inspired many later lower bounds.

In this paper, we are interested in a family of deterministic pivot rules known as history-based (or having memory). For those, superpolynomial lower bounds seemed to be elusive until recently. Arguably, the most famous history-based rule is due to Zadeh. Known as the least entered rule, it was described in 1980 with a technical report that was reprinted in 2009 [26]. This rule keeps a history of how many times each improving variable has been used and, at every step, chooses one that minimizes this history (a tie-breaking rule takes care of ties). The least entered rule was specifically designed to attack constructions similar to the Klee-Minty by using the improving directions in a balanced way (note that in this regard, it is similar to a random walk). With a letter to Klee in the 80s, Zadeh offered a $1000 prize to anyone who can prove polynomial upper or

*athomas@inf.ethz.ch
superpolynomial lower bounds for the least entered rule. This prize was claimed in 2011, by Friedmann [6], with a superpolynomial lower bound on actual Linear Programs (LP). No non-trivial upper bounds are known for this rule.

Another interesting rule was suggested by Cunningham [4], known as the least-recently considered rule. It fixes an initial ordering on all improving directions and then selects one in a round-robin fashion, starting from the last direction selected. The history here is to remember which was the last used improving direction. Furthermore, the least-recently basic rule, which Cunningham attributes to Johnson, was also first discussed in the same paper [4]. That rule selects the improving direction that left the basis least recently (in other words the direction whose opposite was selected least recently). For a detailed exposition on those and many other history-based pivot rules the interested reader should look at Aoshima et al. [1].

We provide exponential lower bounds, by means of Acyclic Unique Sink Orientations, for all three aforementioned history-based rules.

**Unique Sink Orientations** (USO) is an abstract framework that generalizes LP [11] (and other problems [25]). It was originally described by Stickney and Watson [24] and later revived by Szabó and Welzl [25]. Such abstract frameworks have received lots of attention since the discovery of the Random Facet pivot rule: Kalai [16] and, independently, Matoušek, Sharir and Welzl [20] proved subexponential upper bounds for this rule on LP-type problems. It became evident that their analysis made use only of combinatorial properties and, thus, it was possible for Gärtner to extend their upper bounds in a much more abstract setting [8].

The most well-studied such framework is that of USO (e.g. [19], [5], [12], also see below). Intuitively, a USO is an orientation of the hypercube graph such that every non-empty face has a unique sink (vertex with only incoming edges). The computational problem is to discover the unique global sink by performing vertex evaluations (each one reveals the orientation of the edges incident to the vertex). Commonly, acyclic USO (AUSO) constructions have served as lower bounds for pivot algorithms (e.g. [18], [23], [21], [15]) and our lower bounds are also manifested as AUSO.

**Prior work and open questions.** Aoshima et al. [1] explore the possibility that there exist AUSO on which history-based pivot rules take a Hamiltonian path. They prove, with the help of computers, that Zadeh’s pivot rule admits such Hamiltonian paths up to dimension 9 at least. On the contrary, they show that Johnson’s rule (among others) does not admit Hamiltonian paths and, so, they ask if it admits exponential paths on AUSO.

Recently, Avis and Friedmann [2] gave the first exponential lower bound for history-based rules. Namely, they prove an exponential lower bound for Cunningham’s rule on binary parity games (definitions in [2]). Their constructions translate immediately to linear programs and also AUSO, for which the lower bound is \( \Omega(2^{n/5}) \). The constructions of [2] are very complicated and, thus, the

---

1 The exact translation of binary Parity Games to AUSO is explained in [2]. Roughly, in their constructed binary parity game the first player has \( 5n' \) assigned vertices. This translates to a cube of dimension \( 5n' \), where the path that the algorithm will take is of length \( 2^{n'} \). Thus, for \( n \)-dimensional AUSO, that is a lower bound of the form \( \Omega(2^{n/5}) \).
authors ask if it is possible to prove exponential lower bounds for this rule, on AUSO, in a simpler manner.

Moreover, they compare their construction to the one for Zadeh’s rule [6]. The latter gives a family of non-binary parity games (which correspond to linear programs), where Zadeh’s rule takes a subexponential number of steps, of the form $2^{\Omega(\sqrt{n})}$ (where $n$ is the number of variables of the LP). Although binary parity games correspond directly to AUSO, the same is not known for non-binary ones. Hence, Avis and Friedmann ask [2] if superpolynomial lower bounds for Zadeh’s rule exist also on AUSO. In addition, Friedmann’s lower bound [6] is based on a tie-breaking rule which is artificial in the sense that it always works in favor of the lower bound designer. It is not described in the paper because, as the author writes, it is “not a natural one”. Thus, he raises the question [6] of whether it is possible to obtain a lower bound with a natural tie-breaking rule.

Finally, Avis and Friedmann write [2]: “More generally it is of interest to determine whether all of the history based rules mentioned in [1] have exponential behaviour on AUSO”.

Our results. In this paper, we report three lower bounds for the aforementioned rules, on AUSO. With Theorem 4, we give an exponential lower bound for Johnson’s rule. To the best of our knowledge, this is the first superpolynomial lower bound for this algorithm in any context. Moreover, we give an exponential lower bound for Zadeh’s rule, with Theorem 8. This has a number of advantages compared to the known construction: Firstly, it is exponential, whereas Friedmann’s lower bounds [6] are subexponential (also, those do not translate to AUSO). Secondly, our constructions are much simpler to describe. Finally, it is based on a tie-breaking rule that is essentially as simple as possible: a fixed ordered list. These two lower bounds constitute the main results of this paper.

With Theorem 3, we give an exponential lower bound for Cunningham’s rule. The advantage here is that the construction is significantly simpler (our proof can be fully described in a couple of pages, whereas the construction from [2] takes over 30); the lower bound also happens to be slightly improved. Theorem 3 serves as a warm-up to the main results by introducing the techniques and notation we use for our constructions.

Therefore, we answer to the positive all the questions described in the previous paragraph.

Our methods. The constructions in this paper are based on the building tools originally presented by Schurr and Szabó [22]; we do, however, introduce some novel ideas needed to deal with history-based pivot rules. Most known inductive lower bound constructions (e.g. [22, 23, 21, 15]) embed copies of the previous construction into the next one, in such a way that the algorithm gets trapped in the previous construction twice. For Zadeh’s rule this does not work: it balances the directions being used and it inevitably escapes the second trap (at the next inductive step). To overcome this, we build a trap that consists of a small number of copies, being connected in a careful way which ensures that the algorithm uses the improving directions in a balanced fashion: it follows the path of the previous construction, up to making additional “balancing moves” between different copies.
Lower bounds on AUSO. It is not clear if our constructions can be realized as LPs. However, the abstract setting allows for simpler proofs that are easy to communicate. We, thus, believe that such constructions are relevant for understanding the behavior of the pivot rules we study and the ideas could be used for the design of LP-based exponential lower bounds.

For example, the first subexponential lower bounds for Random Facet [18] (tight to the upper bound; see also [9]) and for Random Edge [21] (at every step chooses one improving direction at random) were both proved by AUSO constructions. Indeed, for these two rules, subexponential lower bounds have been later proved on actual LP [7]. The most recent lower bound on AUSO was by Hansen and Zwick in 2016 [15], where they improve the subexponential lower bound for Random Edge. Note that for this rule non-trivial exponential upper bounds are known in the general case [14] and under assumptions [13].

2 Preliminaries

Let $[n] = \{1, \ldots, n\}$ and $\pm[n] = \{-n, \ldots, -1, 1, \ldots, n\}$. Let $Q^n = 2^n$ be the set of vertices of the $n$-dimensional hypercube over coordinates in $[n]$. Often we write $Q^n$ (the superscript indicates the dimension). A vertex of the hypercube $v \in Q^n$ is denoted by the set of coordinates it contains. Generally, with $C \subseteq [n]$ we denote a set of coordinates.

Consider two vertices $v, u \in Q^n$. With $v \oplus u$ we denote the symmetric difference of the two sets. Now, let $C \subseteq 2^n$ and $v \in Q^n$. A face of the hypercube, $F(C, v)$, is defined as the set of vertices that are reached from $v$ over the coordinates defined by any subset of $C$, i.e. $F(C, v) = \{u \in Q^n | v \oplus u \subseteq C\}$. The dimension of the face is $|C|$. We call edges the faces of dimension 1, e.g. $F(\{j\}, v)$. For $k \leq n$ we call a face of dimension $k$ a $k$-face.

Let $\psi$ denote an orientation of the edges of the hypercube $Q^n$. Consider two vertices $v, u \in Q^n$ and a coordinate $j \in [n]$. The notation $v \xrightarrow{\psi} u$ (w.r.t $\psi$) means that $F(\{j\}, v) = \{v, u\}$ and that the corresponding edge is oriented from $v$ to $u$ in $\psi$. Sometimes we write $v \rightarrow u$, when the coordinate is irrelevant. An edge $v \xrightarrow{\psi} u$ is forward if $j \in u$ and otherwise we say it is backward. We use $v \sim w$ to denote (that there is) a directed path from $v$ to $w$.

We can now define the concept of direction; the algorithms that we study in this paper have memory of the directions that have been used so far. A direction is a signed coordinate. Let $c \in C$ be a coordinate; two different directions correspond to $c$, $+c$ and $-c$. At a vertex $v$ the direction $+c$ corresponds to a forward edge incident to $v$ and $-c$ to a backward edge. We say that a direction is available at vertex $v$ if the corresponding edge is outgoing. Thus, at each vertex if a coordinate is incoming then none of the directions is available and if a coordinate is outgoing then exactly one of the directions is available. Similarly to above, we write $v \xrightarrow{\psi} u$, for some direction $d$. Note that if we have $v \xrightarrow{+c} v'$ (similarly $-c$) then at $v'$ neither $+c$ nor $-c$ can be available. Generally, we denote with $D \subseteq \pm[n]$ a set of directions. Given a set of coordinates $C$, we say that $D$ is the set of directions that corresponds to $C$ to mean $D = \{-c, +c \mid c \in C\}$. Often, we use $d$ to denote a direction without specifying its sign.

Then, $\psi$ is a Unique Sink Orientation (USO) of $Q^n$ when every non-empty face has a unique sink. USO can be either cyclic or acyclic (for the latter we write AUSO). $n$-(A)USO means $(A)USO$ over $Q^n$. For a USO $\psi$, we
define $s_\psi$, the outmap function, in the spirit of \cite{25}. For every $v \in Q^n$, $s_\psi(v) = \{ j \in [n] | v \xrightarrow{\psi} (v \oplus \{ j \}) \}$, that is the set of coordinates on which $v$ has an outgoing edge. A sink of a face $F(C, v)$ is a vertex $u \in F(C, v)$, such that $s_\psi(u) \cap C = \emptyset$. The whole cube is a face of itself; thus, there is a unique vertex $v$, the \textit{global sink} with $s(v) = \emptyset$. In the rest of the paper, we write $s(v)$ to denote the outmap of $v$ (the exact orientation $\psi$ will be clear from the context). Moreover, let us call a USO \textit{uniform} when it is such that every edge is oriented towards the global sink. Of course, such a USO is acyclic.

The computational problem associated with a USO is to find the global sink. The computational model is the \textit{vertex oracle} model. We have access to an oracle such that when we give it a vertex $v$, it replies with the outmap $s(v)$ of $v$. This is the standard computational model in USO literature and all the lower and upper bounds are with respect to it.

We are now ready to state the Product and Reorientation lemmas (due to \cite{22}) which are the building tools for the lower bound constructions in this paper. The following constitutes the intuitive description of the Product lemma which is relevant to us: Consider an $n$-AUSO $A$ (oriented hypercube graph) and take $2^m$ copies of $A$. For every vertex $v \in A$, take an $m$-AUSO $A_v$. We call this the \textit{connecting frame} for $v$. Each copy of $A$ corresponds to a vertex of the frame. Then, each vertex $v \in A$ is connected according to $A_v$. The result is an $(n + m)$-AUSO. Formally:

\textbf{Lemma 1} (Product \cite{22}). Let $C$ be a set of coordinates, $C' \subseteq C$ and $C'' = C \setminus C'$. Let $\tilde{s}$ be a USO outmap on $Q^{C'}$. For each vertex $u \in Q^{C''}$ we have a USO outmap $s_u$ on $Q^{C''}$. Then, the orientation defined by the outmap $s(v) = \tilde{s}(v \cap C') \cup s_u(v \cap C'')$ on $Q^C$ is a USO. Furthermore, if $\tilde{s}$ and all $s_u$ are acyclic so is $s$.

The Reorientation lemma, which follows, can be intuitively explained this way: if we have a USO and there is a face, such that all the vertices in this face have the same outmap on the edges external to the face, then we can reorient this face according to any other USO.

\textbf{Lemma 2} (Reorientation. Corollary 6 in \cite{22}). Let $C$ be a set of coordinates, $C' \subseteq C$ and $C'' = C \setminus C'$. Let $s$ be a USO on $Q^C$ and let $F = F(C', u)$, for some $u \in Q^{C'}$ be a face of $Q^C$. If, for any two vertices $v, w \in F$, $s(v) \cap C' = s(w) \cap C''$ and $\tilde{s}$ is a USO on $Q^{C'}$, then the outmap $s'(v) = \tilde{s}(v \cap C') \cup (s(v) \cap C'')$ for $v \in F$ and $s'(v) = s(v)$ otherwise is a USO on $Q^C$.

\section{A warm-up: Cunningham’s Rule}

\textbf{Theorem 3}. There exist $n$-AUSO such that Cunningham’s rule, with a suitable starting vertex and list, takes a path of length at least $2n^{1/4}$.

In this section we will give a proof for the above theorem and introduce the general approach. The section starts with some general comments and definitions that are relevant to all our constructions. Firstly, they are inductive. Let $A_i$ be the $i$th step of the induction. The base case is $A_0$. We call $C_i$ a bundle of coordinates; that is the set of coordinates that was added at the $i$th step of induction (and $C_0$ are the coordinates of the base case). We also define
$C_i^+ = \bigcup_{k=0}^{i} C_k$. Then, $D_i$ denotes the set of directions that corresponds to $C_i$ and, similarly, $D_i^+$ denotes the set of directions that corresponds to $C_i^+$.

Let $v_0^i$ be the starting vertex for $A_i$. Consider that there is a token, which is initially on $v_0^i$, and at every step moves according to the direction that the given algorithm chooses. The path that the token takes from $v_0^i$ to the unique sink of $A_i$ is denoted with $P_i$; its length is denoted with $|P_i|$. In addition, let us denote with $s_i$ the sink of $A_i$.

To construct $A_{i+1}$ from $A_i$, we take $2^m$ copies of $A_i$ and connect their vertices with $m$-dimensional connecting frames, for some constant $m$ (Lemma 1). Afterwards, we perform one reorientation (Lemma 2), to install a simple balancing write 1. The idea is to prove that if we project the rest of the token’s path to the global vertices with $m$-dimensional connecting frames, for some constant $m$. The frames are taken back to the vertex that corresponds to the starting vertex for $A_i$. We refer to these faces in the text; for example, we consider rule. Consider that the algorithm runs on an $n$-AUSO over coordinate checking the directions on the list from $\mu$ of which direction was used last: if direction $L[k]$ was used at the last step then $\mu = k$. At the next step the algorithm will start checking the directions on the list from $L[\mu + 1]$ in a cyclic order (so if it reaches $L[2n]$ it continues from $L[1]$) and it chooses the first available one. Initially, $\mu = 2n$ so that the first direction that the algorithm checks is $L[1]$. An example of a run for this algorithm will be given below.

**The rule.** Firstly, let us formally define Cunningham’s least-recently considered rule. Consider that the algorithm runs on an $n$-AUSO. It has an ordered list $L$ that contains all $2n$ directions; let $L[k]$ indicate the $k$th direction on the list. There is a marker $\mu$ of which direction was used last: if direction $L[k]$ was used at the last step then $\mu = k$. At the next step the algorithm will start by checking the directions on the list from $L[\mu + 1]$ in a cyclic order (so if it reaches $L[2n]$ it continues from $L[1]$) and it chooses the first available one. Initially, $\mu = 2n$ so that the first direction that the algorithm checks is $L[1]$. An example of a run for this algorithm will be given below.

**The construction and relevant notations.** Let $A_0$ be the base case and $L_0$ be the list for it. For the proof of Theorem 3 the base case is not very important but, for the sake of consistency, let us make it a 4-AUSO over coordinate set $C_0 = \{c_1^0, c_2^0, c_3^0, c_4^0\}$. Also, let $L_0 = (+c_1^0, -c_2^0, +c_3^0, -c_4^0, +c_4^0, -c_3^0, -c_2^0, +c_2^0, -c_1^0)$. To construct $A_{i+1}$ from $A_i$ we take $2^i$ copies of $A_i$ which we connect with three different frames, in light of Lemma 1. The frames $F_1$ and $F_2$ are given in Figure 1, the frame $F_3$ is given in Figure 2. The new set of coordinates will be $C_{i+1} = \{c_{i+1}^1, c_{i+1}^2, c_{i+1}^3, c_{i+1}^4\}$. How the three frames connect the copies of $A_i$, will be explained below.

Let us define some notation in reference to the figures below. An AUSO (in the figures of this section 4-AUSO) is given as a collection of 2-faces on the first two coordinates. All coordinates are labeled. Each square represents a face on coordinates $C_i^+$. All of these faces, except $[B]$ are internally oriented according to $A_i$ (correspond to copies of $A_i$). The numbers are indicating in which order the token will visit them. We refer to these faces in the text; for example, we write $[1]$ (in reference to the corresponding figure; here, in reference to Figure 1) to mean the face $F(C_i^+, \{c_{i+1}^2\})$. Given a vertex $v$, we write $v \perp [1]$ to mean
the vertex \( v' \in 1 \), such that \( v \cap C_j^+ = v' \cap C_j^+ \) (the corresponding vertex in 1). Moreover, we write \( 1 \rightarrow 5 \) to mean a path from a vertex in 1 to the corresponding vertex in 5, using only directions from \( D_{i+1} \). In this case, the exact vertex will be clear from the context. The face \( H \) is the one that contains the balancing gadget, which is installed by use of Lemma 2. In this construction and the one of Section 4, \( H \) is a hypersink (has all edges external to the face \( F \)).

In the construction of Section 5 there is no hypersink. The notation of the figures that we just described will also be used in the next sections.

![Diagram](image)

**Figure 1:** The orientations \( F_1 \) and \( F_2 \), used as connecting frames, are given in this figure. The 4-dimensional frames are split in 2-faces of coordinates \( c_{i+1}^j \) and \( c_{i+1}^{j+1} \). In our depictions it is always the case that vertex 0 is at the bottom-left whereas the full vertex that contains all the coordinates is at the top-right (here). The arrows on the other coordinates indicate the orientation of all the edges on this coordinate except when noted differently. For example, in \( F_2 \) all edges on coordinate \( c_{i+1}^3 \) are oriented from left to right (forward) except the edge \( 9 \leftarrow 8 \). This notation is valid also for the next figures.

Based on Figure 2 below, we will give an example run of the algorithm. In that figure we have the connecting frame \( F_2 \); this 4-AUSO also serves as the base case \( A_0 \). Let the starting vertex be at 1, i.e. \( v_0^0 = \{c_0^2\} \). As a reminder, we restate that \( L_0 = (+c_0^1, -c_0^2, +c_0^3, -c_0^4, +c_0^5, -c_0^6, +c_0^7, +c_0^8, -c_0^9) \). Then, the algorithm will use \(+c_0^1, +c_0^3\) and go to 5 (notice that \(-c_0^2\) was not available). From 5 it will use \(-c_0^3, +c_0^5\) and go to \( B \). The rest of the directions in the cyclic order will not be available there. Finally, the algorithm will use \(+c_0^9\) once and the token will go to \( H \) which is the sink for \( A_0 \).

The list \( L_{i+1} \) is defined based on the list \( L_i \) in the following way (here \( \cdot \) represents concatenation):

\[
L_{i+1} = L_i \cdot (+c_{i+1}^1, -c_{i+1}^2, +c_{i+1}^3, -c_{i+1}^4, +c_{i+1}^5, -c_{i+1}^6, +c_{i+1}^7, +c_{i+1}^8, -c_{i+1}^9)
\]

This means that the directions from bundle \( D_k \) have priority over the ones from bundle \( D_{k'} \), if \( k < k' \). Let us denote by IN the set of directions \( D_i^+ \) and let us denote by OUT the set of directions from \( D_{i+1} \). The directions from \( OUT \) are last in the list \( L_{i+1} \). Let the starting vertex be \( v_{i+1}^0 = \{c_{i+1}^0, \ldots, c_{i+1}^9\} \).

We say that the directions from IN have been exhausted when the algorithm has already considered them and it is now the turn of directions from OUT to be considered. Similarly, the directions from OUT have been exhausted when
the algorithm has made a full cycle of the list and the next direction that will be considered is $L[1]$. Note that frame $F_3$ has the property (w.r.t. $L_{i+1}$) that when $IN$ is exhausted at either $1$ or $5$ the token will go to $B$.

Let us now describe how the different frames are used to connect the different copies of $A_i$. Firstly, the starting vertex $v_{i+1}^0$ is connected with frame $F_1$ and the sink vertex $s_i$ is connected with $F_3$. Note that the starting vertex is in $1$. Our goal is to force the algorithm to behave as follows: It uses directions from $IN$, only in faces $1$ and $5$, to follow path $P_i$ (in a projected way) and moves between these two faces, using directions from $OUT$, whenever $IN$ is exhausted.

Every time the token is in $1$ and $IN$ gets exhausted, it takes a path $1 \rightarrow 5$. There it arrives at a vertex where $OUT$ is exhausted. So, it uses a direction from $IN$. After one such move we change the connecting frame to $F_2$. This can be interpreted as an adversary argument and can be implemented by Lemma 1. Similarly, when the token is in $5$ and $IN$ gets exhausted, it takes a path $5 \rightarrow 1$. Like before, after that $OUT$ is exhausted and the algorithm will start considering the list from the beginning. After the first step on one of the directions from $IN$ we change the connecting frame to $F_1$.

Note that while moving between $1$ and $5$ the algorithm will take the path $P_i$ in a projected way. The adversary argument takes care only of vertices that are on this projected path. For the rest of the vertices the frame does not really matter, since they will not be on the path $P_{i+1}$ that the algorithm will take. We use the frame $F_3$ to connect those vertices.

The above procedure produces AUSO $A_{i+1}'$. The latter is acyclic because it has been produced by only using Lemma 1 and all the connecting frames are acyclic. That is not the final AUSO as it remains to install the balance-AUSO in the face $B$. For this we use Lemma 2 to reorient the face $B$ according to the balance-AUSO. The latter is the uniform AUSO that has its sink at $v_{i+1}^0$.

This concludes the construction and the result is $A_{i+1}$. It is easy to see that the reorientation does not introduce any cycles: By the previous discussion, only vertices in $B$ could be involved in cycles. Since the orientation in $B$ is acyclic, any possible cycle must involve coordinates from $C_{i+1}$. However, in all the connecting frames $B$ has only coordinate $c_{i+1}^1$ outgoing to the hypersink.

Figure 2: The frame $F_3$ which is also the base case 4-AUSO $A_0$. The faces $1$ and $5$ are explicitly labeled since the sink $s_i$ of $A_i$ will be found (for the first time; Step (3) in the step-by-step analysis) in one of them.
Thus, it is not possible for any vertex from $B$ to be involved in any cycles and $A_{i+1}$ is acyclic. Next we give a detailed step-by-step analysis of the behavior of the token on AUSO $A_{i+1}$.

**Step-by-step analysis.** The token is initially at the starting vertex $v_i^{i+1} = \{c^{i+1}_0, \ldots, c^{i+1}_s\}$. In detail:

1. The token is at 1. Directions from $IN$ will be used, since they have priority in $L_{i+1}$. After some steps they will be exhausted. The connecting frame for $C_{i+1}$ will be $F_1$. The token then takes a path $1 \rightarrow 5$. There, the $OUT$ directions will be exhausted. The correctness of this argument can be checked in Figure 1.

2. The token is now in 5. Again, directions from $IN$ will be used, since they have priority in $L_{i+1}$. After some steps they will be exhausted. The connecting frame for $C_{i+1}$ will be $F_2$. The token then takes a path $5 \rightarrow 10 \rightarrow 1$. There, the $OUT$ directions will be exhausted. The correctness of this argument can be checked in Figure 1.

3. The algorithm will keep looping between steps (1)-(2) until the token reaches a vertex $v_{s_i}$ such that $s(v_{s_i}) \cap C^{i}_r = \emptyset$ (equivalently, $v_{s_i} \cap C^{i}_r = s_i$). That is a vertex that corresponds to the sink of $A_i$.

4. The connecting frame for $v_{s_i}$ is $F_3$. From the arguments in (1) and (2) this vertex will be reached in either 1 or 5. Note that in both cases, the algorithm following $L_{i+1}$ will take the token to $B$ and the directions from $OUT$ will be exhausted there. The correctness of this argument can be checked in Figure 2.

5. The token now is at vertex $v_{s_i} \perp B$ and the algorithm will start checking at $L[1]$. The token takes a path $v_{s_i} \perp B \rightarrow v_0 \perp B$. The last vertex is the sink of $B$.

6. In the next step, the algorithm will use $+c^{i+1}_i$ to go to $H$. In there, the token is at vertex $v_0 \perp H$. The coordinates from $C_{i+1}$ will remain incoming from now on. Thus, the token will take a path in $H$ that is the same as $P_i$.

With the above analysis, we have proved that the path $P_{i+1}$ will have length that is larger than twice the length of path $P_i$. Therefore, we obtain the recursion $T(n+4) > 2T(n)$ which leads to the proof of Theorem 3.

## 4 Exponential lower bound for Johnson’s rule

**Theorem 4.** There exist $n$-AUSO such that Johnson’s rule, with a suitable starting vertex, takes a path of length at least $2^{n/4}$.

In this section we will prove the above theorem. Firstly, let us define Johnson’s least-recently basic rule. Consider that the algorithm runs on an $n$-AUSO. It maintains a history function $h$ which is defined on all $2n$ directions. Let $v$ be the current vertex. Intuitively, the algorithm keeps the following history: Say
direction $d$ was used at step $x$ and let $-d$ be the opposite of that direction. Then, at step $x$ we have $h(-d) = x$; this will stay intact until $-d$ is used. On the other hand, $h(d)$ will keep increasing to the current step until $-d$ is used.

Formally, for a direction $d$, $h(d)$ is the last step number when $|d| \in v$ if $d$ is positive and the last step number when $|d| \notin v$ if $d$ is negative. Here, $v$ is the vertex on the path of the algorithm that corresponds to this last step and $|d|$ denotes the coordinate that corresponds to direction $d$. Note that ties are possible. We assume that those are broken lexicographically. The algorithm chooses from the set of available directions, direction $d$ which minimizes $h(d)$.

**An example run.** We describe an example run of Johnson’s rule on the 4-AUSO $F_1$ that is given in Figure 3. Note that $F_1$ will also serve as the base case $A_0$ for the inductive construction (to be described after this example). In the table below the first column is the step number. The second column is the vertex at the beginning of the corresponding step. The third column is the direction chosen by the algorithm at that step. The rest of the columns show the history of each direction after the step was performed. Below we use $\pm k$ to mean direction $\pm c_k$. The starting vertex is 1.

| Step | vertex | $d$ | $h(+1)$ | $h(+2)$ | $h(+3)$ | $h(+4)$ | $h(-1)$ | $h(-2)$ | $h(-3)$ | $h(-4)$ |
|------|--------|-----|---------|---------|---------|---------|---------|---------|---------|---------|
| 1    | 1      | +1  | 1       | 0       | 0       | 0       | 1       | 1       | 1       | 1       |
| 2    | 2      | +2  | 2       | 2       | 0       | 0       | 1       | 2       | 2       | 2       |
| 3    | 3      | +3  | 3       | 3       | 3       | 0       | 1       | 2       | 3       | 3       |
| 4    | 3      | +4  | 4       | 4       | 4       | 4       | 1       | 2       | 3       | 4       |
| 5    | 3      | -3  | 5       | 5       | 5       | 5       | 1       | 2       | 3       | 4       |
| 6    | R      | -2  | 6       | 6       | 5       | 6       | 1       | 6       | 4       | 4       |
| 7    | H      | 7   | 6       | 5       | 7       | 1       | 7       | 7       | 4       | 4       |

The construction is inductive. Let $A_i$ denote the $i$th inductive step. The base case $A_0$ is the 4-dimensional AUSO $F_1$, shown in Figure 3. The initial set of coordinates is $C_0 = \{c^1_0, c^2_0, c^3_0, c^4_0\}$. The starting vertex is $v_0 = \emptyset$, which is at the vertex labeled 1 in the figure. As discussed in the previous paragraph, the algorithm will go over directions $(+c^1_0, +c^2_0, +c^3_0, +c^4_0, -c^1_0, -c^2_0)$ and find the sink at $\{c^1_0, c^2_0\}$. Let us now describe how to construct AUSO $A_{i+1}$ from AUSO $A_i$. Every inductive step adds 4 dimensions. As before, $C^{\pm}_j = \bigcup_{k=0}^{j} C_k$. Let the new bundle of coordinates be $C_{i+1} = \{c^1_{i+1}, c^2_{i+1}, c^3_{i+1}, c^4_{i+1}\}$. We take 16 copies of $A_i$ and connect them with the 4-AUSO $F_1$ and $F_2$ that appear in Figure 3. In this section, a reset-AUSO (thus, the $\mathbf{R}$ in the figure) will take the role of the balancing gadget.

Let us define what we mean by lexicographic order here: $+c^j_k$ comes before $-c^j_{k'}$ for any $k$ and $k'$ (positive before negative in the same bundle); $+c^j_k$ comes before $+c^j_{k'}$ and $-c^j_k$ comes before $-c^j_{k'}$ if $k < k'$ (in the same bundle smaller-index directions come first). Finally, $d^j_k$ comes before $d^j_{k'}$ for any $k$ and $k'$ and positive/negative sign, if $j < j'$ (bundles with smaller index come first).

The starting vertex will be $v_0 = \emptyset$ for every inductive step. Then, every positive direction $+c$ initially has $h(+c) = 0$ (until $+c$ is used by the algorithm). At step number 1, one of the positive directions will be used at which point every negative direction $-c$ has $h(-c) = 1$. 

10
Figure 3: The orientations $F_1$ and $F_2$, used as connecting frames, are given in this figure.

With this construction we want to force the algorithm to the following behavior: It starts at 1 using all the positive directions in lexicographic order. Then it is in 5, where it will use all the negative directions in lexicographic order. This will continue until the sink of $A_i$ has been reached. We will later show that this is the case.

The construction is considered as an adversary argument (similarly to Section 3). Firstly, the starting vertex of $A_i$ and the sink of $A_i$ both use $F_1$ as the connecting frame. Every time the algorithm is in 1 and uses a direction from $D_i^+$, we change (or keep) the connecting frame to $F_1$. Similarly, when the algorithm arrives in 5 and uses a direction from $D_i^+$, we change the connecting frame to $F_2$ except if the algorithm has reached the sink of $A_i$. This operation is consistent with Lemma 1: Every vertex is connected with the corresponding frame. The result of this operation is $A'_i$, which is not the final AUSO. The final step for the construction of $A_i$ is to use Lemma 2 to embed a reset-AUSO in the face $R$.

**Constructing the reset-AUSO.** This construction is also inductive. Let $R_i$ denote the $i$th inductive step. Then, $R_i$ is a 4i-AUSO (whereas $A_i$ is a 4($i$ + 1)-AUSO). Since $R_i$ is placed in a face of $A_i$, the coordinates it spans are the ones in $C_{i-1}^+$. The final step in the construction of $A_i$ is to embed the reset-AUSO $R_{i+1}$ in the face $R = F(C_i^+, \{c_{i+1}^1, c_{i+1}^2, c_{i+1}^4\})$ of $A_{i+1}$.

$R_0$ is just a 0-AUSO, i.e. a vertex; or equivalently, there is no reset-AUSO for $A_0$. Then, let $R_1$ be the base case; this 4-AUSO is depicted in Figure 4 below. Then, to construct $R_{i+1}$ from $R_i$ we take 16 copies of $R_i$ which we connect with two frames. For every vertex in $R_i$ which is not the sink, we use for connecting frame the uniform 4-AUSO which has its sink at $\{c_{i+1}^1, c_{i+1}^4\}$. For the sink of $R_i$ we use for a frame the 4-AUSO $R_1$. The intuition behind reset-AUSO $R_{i+1}$ is summed up by the following property: It has a path from vertex $\{c_{0}^1, c_{i}^4, \ldots, c_{i}^1, c_{i}^4\}$ to the vertex $\emptyset$ such that every vertex on this path has only one outgoing edge and the path goes through the negative directions in lexicographic order: $(-c_{0}^3, -c_{i}^3, \ldots, -c_{i}^3, -c_{i}^3)$.

The final step for the construction of $A_{i+1}$ is to use Lemma 2 to embed $R_{i+1}$ to the face $R$. This concludes the construction of $A_{i+1}$. Note that the last step
does not introduce any cycles: $R_{i+1}$ is acyclic and, thus, the only possible cycles would involve the edges on coordinates external to $R_i$ i.e. the coordinates from $C_{i+1}$. However, the hypervertex $[R]$ has only one outgoing edge on $-c^2_{i+1}$ which leads to the hypersink $[H]$; the latter does not have any outgoing edges on $C_{i+1}$ and, so, a cycle is not possible. We conclude that the constructed AUSO is acyclic.

The behavior of Johnson’s rule on the AUSO constructed as above will be described here. Firstly, we define the tools that we are going to use for this analysis.

Similarly to the previous section, consider a token $t$. That is a token that starts at the initial vertex $v_0$ and moves according to the directions that the algorithm chooses. With slight abuse of notation we also use $t$ to refer to the vertex where the token currently lies on. Moreover, we write $t_j$ to mean the set $t \cap C^+_j$; that is, the projection of the vertex $t$ to the set of coordinates $C^+_j$. Since $t_j \subseteq t$, we call $t_j$ a subtoken.

We say that a coordinate bundle $C_j$ is active when $s(t) \cap C_j \neq \emptyset$. Otherwise, we say that $C_j$ is inactive. Note that for both the 4-dimensional frames that we have used, the sink is at the same vertex. This implies that $C_j$ is inactive if and only if token $t$ is such that $t \cap C_j = \{c^1_j, c^4_j\}$. Moreover, we say that token $t$ is in $[1]$ to mean that $t \cap C_j = \emptyset$; $t$ is in $[5]$ when $t \cap C_j = C_j$ and similarly for the rest of the faces $[\cdot]$ from Figure 3.

For each subtoken $t_j$, we say that it has reached its sink when all bundles in $C^+_j$ are inactive. This means that $t_j = \{c^1_j, c^4_j, \ldots, c^1_j, c^4_j\}$. Resetting $C^+_j$ is a process that happens when subtoken $t_j$ is at its sink: Token $t$ moves from a vertex where $t \cap C^+_j = \{c^1_j, c^4_j, \ldots, c^1_j, c^4_j\}$ to a vertex where $t \cap C^+_j = \emptyset$. Moreover, we say that $C^+_j$ is resettable when:

- $t_j$ is at its sink.
- $h(-c^1_j) < h(-c^4_j) < h(-c^2_{j+1})$, for every $0 \leq j' \leq j$.

The first bullet in the definition above equivalently means that all bundles in $C^+_j$ are inactive. Resetting $C^+_j$ is a process that takes place when (and only when) token $t$ is in the reset-AUSO in $[R]_{j+1}$. For now assume that $C^+_j$ will be reset.
only when it is resettable; we will prove this later (with Lemma 7). Thus, \( t_j \) is on its sink when the resetting process starts. The second bullet of the definition of resettable ensures that token \( t \) will not go out of \( R_{j+1}^+ \) before \( C_j^+ \) has been reset. During the reset of \( C_j^+ \), token \( t \) will go over negative directions from \( D_j^+ \) in this order: \((−c_1, −c_2, ..., −c_j, −c_j^+)\). This is because of the construction of the reset-AUSO \( R_{j+1}^+ \); the algorithm has no other choice.

We are ready to state the following lemma. It dictates the first steps that the token will take, either at the beginning of the execution of the algorithm or after a reset. Since the algorithm is deterministic, this defines the path that the token will take to the sink.

**Lemma 5.** Let \( t \cap C_j^+ = \emptyset \). Then, all the positive directions from \( C_j^+ \) will be used in the lexicographic order: \((+c_0^1, +c_0^2, +c_1^2, +c_1^3, ..., +c_j^1, +c_j^2, +c_j^3, +c_j^+\).

**Proof.** The token \( t \) is at a vertex such that \( t \cap C_j^+ = \emptyset \). Note that the connecting frame is \( F_1 \) for all the bundles in \( C_j^+ \). There are two cases. The first case is when the algorithm is just starting at the initial vertex \( v_0 \). Since \( v_0 = \emptyset \), all positive directions have history 0 and they will be used in the lexicographic order.

The other case is after \( C_j^+ \) has been reset. By the definition of resetting, we have that \((−c_0^1, −c_0^2, ..., −c_j^1, −c_j^+\) have been lastly used. Thus, for every \( 0 \leq k \leq j \) we have that \( h(+c_k^1) < h(+c_k^2) < h(+c_k^3) < h(+c_k^+) \). In addition, \( h(+c_k^2) < h(+c_k^3) < h(+c_k^+) \) for all \( 0 \leq k < k' \leq j \). Since the connecting frame is \( F_1 \) for all the bundles in \( C_j^+ \), we have that direction \(+c_k^+\) is only available after \(+c_k^1\) has been used and direction \(+c_k^2\) is only available after \(+c_k^1\) and \(+c_k^2\) has been used. So, the direction \(+c_k^+\) is the first one that will be used. Then, \(+c_0^1\) becomes available and will be used, then \(+c_0^2\) becomes available and will be used and, finally, \(+c_0^3\) will be used. Then, \(+c_1^1\) will be used and so on and so forth. We conclude that all the positive directions from \( C_j^+ \) will be used in the lexicographic order.

The above lemma defines the path that token \( t \) will follow until subtoken \( t_j \) reaches its sink. For every bundle \( C_j \), the positive directions are used in lexicographic order \((+c_j^1, +c_j^2, +c_j^3, +c_j^+)\) and the token \( t \) goes to \( 5 \). After this, we have \( h(−c_j^1) < h(−c_j^2) < h(−c_j^3) < h(−c_j^+) < h(d) \) for any positive \( d \) from \( D_j \).

Then, some directions from \( D_{j−1}^+ \) will be used and the frame for \( C_j \) will change to \( F_2 \). When it is the turn of the negative directions from \( D_j \) to be used they will be used consecutively and in lexicographic order \((−c_j^1, −c_j^2, −c_j^3, −c_j^+)\); that is, assuming \( t_{j−1} \) has not reached its sink yet. Otherwise, the connecting frame for \( C_j \) would be \( F_1 \) and the directions \(−c_j^1\) and \(−c_j^2\) would not be available.

After this, \( t \) will be in \( 1 \) and, moreover, \( h(+c_j^1) < h(+c_j^2) < h(+c_j^3) < h(+c_j^+) < h(d) \) for any negative \( d \) from \( D_j \). There, after some direction from \( D_{j−1}^+ \) is used, the connecting frame for \( C_j \) will be \( F_1 \). When it is the turn of the positive directions from \( D_j \) to be used they will be used consecutively and in lexicographic order \((+c_j^1, +c_j^2, +c_j^3, +c_j^+)\) and the token \( t \) will go back to \( 5 \). We can conclude that \( t \) will keep moving from \( 1 \) to \( 5 \) and back until subtoken \( t_{j−1} \) reaches its sink. The next corollary follows from this discussion.

**Corollary 6.** Let \( C_{j+1} \) be active.
1. If $t$ is in $\mathbf{J}_{i+1}$, the positive directions from $D_{j+1}$ will be used (when it is their turn) consecutively as $(+c_{1}^{1}, +c_{2}^{2}, +c_{3}^{3}, +c_{4}^{4})$.

2. If $t$ is in $\mathbf{J}_{j+1}$ and subtoken $t_{j}$ has not reached its sink, then the negative directions from $D_{j+1}$ will, similarly, be used (when it is their turn) consecutively as $(-c_{1}^{1}, -c_{2}^{2}, -c_{3}^{3}, -c_{4}^{4})$.

It follows that directions from $D_{j}^{+}$ are only used when $t$ is in $\mathbf{J}_{i+1}$ or in $\mathbf{J}_{j+1}$.

The next lemma is the last ingredient needed for the proof of Theorem 4.

**Lemma 7.** Let $C_{j+1}$ be active. When $t_{j}$ reaches its sink, $C_{j}^{+}$ is resettable.

**Proof.** By definition, when $t_{j}$ reaches its sink all bundles in $C_{j}^{+}$ are inactive.

We prove the statement by induction on $j$. For the base case $j = 0$, we explicitly give the directions from $D_{1}^{+}$ in the order they are used by the algorithm until $t_{0}$ reaches its sink: $(+c_{1}^{1}, +c_{2}^{2}, +c_{3}^{3}, +c_{4}^{4})$. It follows that $C_{0}^{+} = C_{0}$ is resettable when $t_{0}$ reaches its sink (while $C_{1}$ is still active).

Now consider any $k$, $1 < k \leq j$. Let $C_{k+1}$ be active; we will prove that $C_{k}^{+}$ is resettable when $t_{k}$ reaches its sink. Consider the last step when $t$ was in $\mathbf{I}_{k}$. The algorithm will use directions $(+c_{1}^{k}, +c_{2}^{k}, +c_{3}^{k}, +c_{4}^{k})$, by Corollary 6 and token $t$ will go to $\mathbf{J}_{k}$. Note that this is regardless of whether subtoken $t_{k-1}$ has reached its sink. The result is that $h(-c_{k}^{1}) < h(-c_{k}^{2}) < h(-c_{k}^{3}) < h(-c_{k}^{4})$. The subtoken $t_{k-1}$ reaches its sink either when $t$ is in $\mathbf{I}_{k}$ or when $t$ is in $\mathbf{J}_{k}$, by Corollary 6.

Afterwards, the algorithm uses direction $-c_{k}^{3}$ and $t$ goes to $\mathbf{R}_{k}$ ($C_{k}$ is still active). Therein, $C_{k-1}^{+}$ will get reset since it is resettable by the inductive hypothesis. Following, the algorithm uses $-c_{k}^{2}$, $t$ goes to $\mathbf{H}_{k}$, and $C_{k}$ becomes inactive. So, we still have $h(-c_{k}^{1}) < h(-c_{k}^{2})$. In addition, since $C_{k-1}^{+}$ just got reset we have that $t \cap C_{k-1}^{+} = \emptyset$. By Lemma 5 all the positive directions from $D_{k-1}^{-}$ will be used. Since $C_{k+1}$ is still active, the positive directions from $D_{k+1}$ will be used at least once; this includes $+c_{j}^{j}$. However, $C_{k}$ is already inactive and no directions from $D_{k}$ will be used. Thus, we have $h(-c_{k}^{1}) < h(-c_{k}^{2}) < h(-c_{k}^{3})$. 

Now consider the path of the token $t$ from $v_{0}$ to the sink of $A_{i+1}$. This AUSO is of dimension $n = 4(i + 2)$. By Corollary 6 $t$ will be moving back and forth between $\mathbf{I}_{k}$ and $\mathbf{J}_{k}$ until $t_{i}$ reaches its sink. If we project the path that $t$ has taken until this point on coordinates from $C_{i}$ it will be the same as $P_{i}$. When $t_{i}$ reaches its sink, $C_{i}^{+}$ is resettable (by Lemma 7) and, when $t$ enters $\mathbf{R}_{i}$, $C_{i}^{+}$ will be reset. Following, $t$ enters $\mathbf{H}$ at vertex $v_{0}$.

Consider the path $P'$ that $t$ will follow from $v_{0}$ until $t_{i}$ reaches its sink. Since every vertex in $\mathbf{H}$ has only incoming edges on $C_{i+1}$, the coordinates that the path uses are from the set $C_{i}^{+}$. From Lemma 5 we know that when token $t$ is such that $t \cap C_{i}^{+} = \emptyset$, all the positive directions from $D_{i}^{+}$ will be used in the lexicographic order. Therefore, the path $P'$ will be the same as $P_{i}$ when $t_{i}$ reaches its sink, at which point $t$ will have reached the global sink of $A_{i+1}$.

14
Let $T(n)$ denote the length of the path that token $t$ will take from $v_0$ until it reaches the global sink on a $n$-AUSO. With the above analysis, we have shown that the recursion $T(n + 4) > 2T(n)$ holds. This recursion leads to the proof of Theorem 4.

5 Exponential lower bound for Zadeh’s Rule

Theorem 8. There exist $n$-AUSO such that Zadeh’s rule, with a suitable starting vertex and tie-breaking rule, takes a path of length at least $2^{n/6}$.

In this section we will prove the above theorem. Firstly, let us define formally Zadeh’s least entered rule. Consider that the algorithm runs on an $n$-AUSO. It maintains a history function $h$ which is defined on all $2^n$ directions. Given a direction $d$, $h(d)$ is the number of times the direction $d$ has been used. At the beginning $h(d) = 0$, for all $d$. At every step the algorithm picks one direction from the set of available ones that minimizes the history function. In addition, there is a tie-breaking rule: this is an ordering of the directions and is invoked only in case more than one have the minimum history size. As we already mentioned in Section 1, our lower bound construction will have the simplest possible tie-breaking rule, an ordered list which will be given explicitly. This is in contrast to the lower bounds by Friedmann [6].

Secondly, let us define some tools that we will use for the analysis of the algorithm. We have a balance function $b$, which is also defined on all the $2^n$ directions. Let $d_{\text{max}}$ be the most used direction; then, $b(d) = h(d_{\text{max}}) - h(d)$. This means that direction $d$ has been used $b(d)$ less times compared to $d_{\text{max}}$. We say that a direction $d$ is imbalanced when $b(d) > 0$ and that a set of directions $D$ is balanced when $b(d) = 0$, for all $d \in D$. We also define a balance function on any subset of directions: Given set $D$ we define $b(D,d)$ to be the balance of direction $d$ w.r.t. the directions from $D$, i.e. the defining coordinate is now $d_{\text{max}} \in D$.

Furthermore, we define the concept of saturation. This is with regards to the history and the current vertex in the algorithm run. Given a set of directions $D$ and a vertex $v$ we say that $v$ is $D$-saturated when for every $d \in D$ with $b(d) > 0$, the direction $d$ is not available for $v$. It follows that if vertex $v$ is $D$-balanced, then $v$ is also $D$-saturated.

The construction is inductive. Let $A_i$ denote the $i$th inductive step. The base case, $A_0$, is a 6-AUSO. This will be described later with Figure 7. Following, every inductive step adds 6 new dimensions. As before the bundle of coordinates $C_i$ is the one added at the $i$th step of induction (and $C_0$ is the bundle for the base case). Also, with $D_i$ we denote the directions that correspond to $C_i$ and, similarly, for $D_i^+$. Thus, the AUSO $A_i$ is $6(i+1)$-dimensional and the coordinates that describe it are in the set $C_i^+$. For each $A_i$, the starting vertex is $v_0^i = \{c_0^i, \ldots, c_7^i\}$.

Let us now describe how to construct AUSO $A_{i+1}$ from AUSO $A_i$. Let the new bundle of coordinates be $C_{i+1}$. We call $IN$ the set of directions $D_i^+$ and $OUT$ the set of directions $D_{i+1}$. At every inductive step the tie-breaking rule will be formed such that the directions from $IN$ have priority over the ones from $OUT$. Thus, directions $D_k$ have priority over the ones from $D_{k'}$, if $k < k'$. 

15
The starting vertex for $A_{i+1}$ is $v_0 = v_0^{i+1} = \{c_0^2, \ldots, c_{i+1}^2\}$. Similarly to the previous sections, the first part of the construction is interpreted as an adversary argument. To construct $A'_{i+1}$ (not the final construction) we take $2^6 = 64$ copies of $A_i$ and use three different 6-AUSO as connecting frames, utilizing Lemma 1. The frames are given in Figures 5 and 6. For each vertex on the path of the algorithm we choose the connecting frame according to the following rule:

1. We connect with $F_1$ vertices that are not IN-saturated;
2. We connect with $F_2$ vertices that are IN-saturated;
3. For the sink $s_i$ of $A_i$ we use $F_3$.

Note that the connecting frame is not important for vertices that are not on the path of the algorithm. For the sake of completeness let us connect these vertices using $F_3$ as the connecting frame. The latter is a 6-AUSO that has the same path $[1] \sim [12]$ as $F_1$, has its sink at $[12]$ and all other edges are forward. It will is described pictorially in Figure 6.

The result of this operation is $A'_{i+1}$. The latter is acyclic since it arises by applying only Lemma 1. To obtain our final construction it remains to perform one reorientation. Namely, the balance-AUSO will be embedded in the face $[B] = F(C_i^+, \{c_{i+1}^2, c_i^2, c_0^2\})$, shown in Figure 5. The balance-AUSO is the uniform 6$(i+1)$-AUSO which has its sink at the vertex $v_0^i$. Formally, the outmap of vertex $v = v_0 \perp [B]$ is such that $s(v) \cap C_i^+ = \emptyset$. The result of that reorientation will be the final AUSO $A_{i+1}$. Firstly, let us argue that $A_{i+1}$ is acyclic.

Lemma 9. $A_{i+1}$ is acyclic.

Proof. We already mentioned above that $A'_{i+1}$ is acyclic; it remains to argue that after the reorientation of $[B]$ do not introduce any cycles. Note that $[B]$ has only forward edges incident on the coordinates from $C_{i+1}$ in all three frames. From [B] there are paths to vertices in faces from the set $S_1 = \{[1], \ldots, [12]\}$; those are the only vertices reachable from $[B]$ that have incident backward edges on coordinates from $C_{i+1}$. However, in none of the frames there is path from any face in set $S_1$ to some face from the set $S_2 = \{[1], \ldots, [12]\} \cup F(C_i^+, \emptyset) \cup F(C_i^+, \{c_{i+1}^2, c_0^2\})$. The set $S_2$ contains all the faces which have vertices with paths to $[B]$ (in all three faces). Therefore, there is no cycle involving $[B]$ and, then, $A_{i+1}$ is acyclic.

Secondly, let us give the figure that describes the frames $F_1$ and $F_2$ (Figure 5).
Figure 5: Both orientations $F_1$ and $F_2$ are given in this figure. For simplicity, only the backward edges are explicitly oriented; every other edge is forward. The frame $F_1$ includes the dashed backward edges but not the dotted ones; $F_2$ includes the dotted backward edges but not the dashed ones. The solid backward edges are included in both frames.

In reference to Figure 5 let us present the intuitive idea of this lower bound: The token will walk (in a projected way) along path $P_i$, while walking between 1 and 12. Then, it will go back to the start of $P_i$ in the balance-AUSO $B$. Then, it will walk the path $P_i$ once again (in a projected way), while walking between 1 and 12.

The frame $F_3$, which is used to connect the vertices that correspond to the sink $s_i$ of $A_i$, is presented in the next figure (Figure 6). Note that the sink of the frame is at 12 which also places the global sink of $A_{i+1}$ in 12. The frame $F_3$ has two useful properties. The first is that it has only forward edges attached at 1. This will take the token from there to $B$ (w.r.t. $T_{i+1}$). The second is that there exists the same path $1 \leadsto 12$ as in $F_2$ (but the sink is at 12 for $F_3$).
Following, we give three crucial properties that will hold for our construction. The first one is about the base case $A_0$ (refer to Figure 7).

(i) There is at least one $(D_0)$-saturated vertex, other than the starting vertex $v_0$ and the sink $s_0$, in $A_0$. In addition, the sink $s_0$ of $A_0$ is at least two vertices away from the last vertex on the path $P_0$ that was $(D_0)$-saturated.

Property (i) will be utilized in the proof of Lemma 10. The other two properties hold for every inductive step $A_k$ of the construction $0 \leq k \leq i + 1$.

(ii) Let the token $t$ be at a vertex $v$ such that $v$ is $D^+_k$-saturated. Then the algorithm will use each direction from $D^+_k$ at most once and it will reach another $D^+_k$-saturated vertex.

(iii) When the token reaches the sink $s_k$ of $A_k$ there are exactly $4(k+1)$ negative imbalanced coordinates: $IM_k = \{-c^3_0, -c^4_0, -c^5_0, \ldots, -c^3_k, -c^4_k, -c^5_k, -c^6_k\}$.

For every $d \in IM_k$ we have $b(d) = 1$ and for every other $d$ we have $b(d) = 0$.

In the next figure we give the 6-AUSO for the base case $A_0$ and argue that Properties (i), (ii) and (iii) hold for it. The arguments will appear in the caption of Figure 7.
Figure 7: The base case $A_0$. Only the backward edges are explicitly drawn. The tie-breaking rule $T_0$ is essentially the same as for the inductive step. Specifically, $T_0 = (+1, -2, +3, -1, +4, -3, +5, -4, +6, -5, +2, -6)$. Here, a number $\pm k$ indicates the direction $\pm c_k^0$. The numbers in the figure indicate the path that the token will take: $1 \rightarrow 21$. Note that $12$ is $D_0$-saturated while the sink is at $21$ and, so, Property (i) is satisfied. The initial vertex $1$ is, of course, $D_0$-balanced and $D_0$-saturated. From there the algorithm uses $|D_0| - 1$ different directions to reach vertex $12$ which is $D_0$-saturated. From the latter the algorithm uses $|D_0| - 3$ different directions to reach vertex $21$ which is the sink of $A_0$ and, thus, $D_0$-saturated. Therefore, Property (ii) is satisfied for the base case. After the sink is reached at $21$, we have $b(-3) = b(-4) = b(-5) = b(-6) = 1$ and $b(d) = 0$ for all other $d \in D_0$. This satisfies Property (iii) for the base case.

In the step-by-step analysis, which comes later, we will argue that they also hold for every inductive step of the construction. Also note, with regards to Property (iii), that if the token takes the directions in $IM_k$ from the sink $s_k$, it will go back to the starting vertex $v_k^0$. Such a path is not available in any of the connecting frames; however, such a path is available in the balance-AUSO of $A_{k+1}$. We will now analyze the behavior of Zadeh’s rule on AUSO $A_{i+1}$. Firstly, let us define the tie-breaking ordered list $T_{i+1}$:

$$T_{i+1} = T_i \cdot (+1, -2, +3, -1, +4, -3, +5, -4, +6, -5, +2, -6).$$

Here, a number $\pm k$ indicates the direction $\pm c_k^{i+1}$. Furthermore, the next lemma
states that if the token is at a $D_{i+1}^+$-saturated vertex that does not have all edge on $C_i^+$ incoming, then there is an outgoing edge from $D_0$ to a neighboring vertex such that the latter is not IN-saturated.

**Lemma 10.** Let token $t$ be at a $D_{i+1}^+$-saturated vertex $v$, such that $s(v) \cap C_i^+ \neq \emptyset$. Then the next step of the token will be $v \xrightarrow{d} v'$ such that $d \in D_0$. Moreover, $v'$ is not IN-saturated and $s(v') \cap C_i^+ \neq \emptyset$.

**Proof.** Since vertex $v$ is $D_{i+1}^+$-saturated the next direction that will be chosen by the algorithm will be from the smallest index bundle that has an available direction at $v$. Note that it is actually not possible that $s(v) \cap C_0 = \emptyset$, $v$ is $D_{i+1}^+$ saturated and $s(v) \cap C_i^+ \neq \emptyset$ at the same time: if $v$ was such that $s(v) \cap C_0 = \emptyset$ then, by Property (i), the directions from $IM_0$ would not be balanced. By design the token would go to a balance-AUSO where the directions from $|IM_0|$ are available. We conclude that $s(v) \cap C_0 \neq \emptyset$.

Then, the next direction chosen by the algorithm at $v$ will be $d \in D_0$, $v \xrightarrow{d} v'$. After this move we will have that $b(d') > 0$ for every $d' \in D_{i+1}^+$, $d' \neq d$. Thus, $v'$ will not be IN-saturated. In addition, by Property (i), $v'$ will be such that $s(v) \cap C_0 \neq \emptyset$.

Moreover, we one last notation that will help us with the upcoming step-by-step analysis: Let us denote with $d_{\max}^{\text{OUT}}$ the direction that maximizes history over the $OUT$ directions; similarly, we define $d_{\max}^{\text{IN}}$.

**Step-by-step analysis.** We are ready to give a description for the behavior of the algorithm on $A_{i+1}$. The token is initially at the starting vertex $v_0 = v_0^{i+1} = \{c_0^2, \ldots, c_{i+1}^2\}$. In the rest we write $\pm k$ to mean $c_{i+1}^k$.

Note that in the analysis below we will also argue that Properties (ii) and (iii) are inductively satisfied. For the latter a very simple accounting of the imbalanced coordinates at the end of the run of the algorithm is enough: this will be provided at Step (10). For the former we basically have to show that when the token is at a $D_{i+1}^+$-saturated vertex then it will go through at most $|D_{i+1}^+| - 1$ many different directions to reach another $D_{i+1}^+$-saturated vertex. For the directions from $IN$ the inductive hypothesis will be employed. For the directions from $OUT$ we will explicitly do the accounting. This will be found at Steps (2), (5), (8) and (10) of the analysis.

(1) The token is at $v_0$. Directions from $IN$ will be used, since they have priority in $T_{i+1}$. After some steps, an IN-saturated vertex $v_i$ will be reached, by Property (ii) which holds for $A_i$ by the inductive hypothesis. At $v_i$, we have that $b(d_{\max}^{\text{OUT}}) = 1$. The connecting frame will be $F_2$. The directions from $OUT$ will be utilized and used in the order defined by $T_{i+1}$. The token will take a path $v_0^{i+1} \rightarrow 12$, where it will reach $v_s \downarrow 12$.

(2) When the token reaches $12$ we have: $b(-6) = 1$ and for every other direction $d \in OUT$, $b(d) = 0$; also, $b(d_{\max}^{\text{OUT}}) = 0$. The frame is $F_2$ and the dashed edge is not available: $v_s \downarrow 12 \leftarrow v_s \downarrow 1$. This means that $v_s \downarrow 12$ is $D_{i+1}^+$-saturated (it is $OUT$-saturated and $IN$-saturated). We conclude that Property (ii) also holds for the new inductive step up to this point (the
algorithm has used each direction from OUT at most once since the previous
$D_{i+1}^+$-saturated vertex).

Then, the algorithm will use one direction from IN. The next vertex $v$ is not IN-saturated, by Lemma 10. Thus, the connecting frame for $v$ is $F_1$. Direction $-6$ (which is the only direction that has $b(-6) = 2$ at this point) will be used and the token will go to $1$. Afterwards, $b(OUT, d) = 0$, for every $d \in OUT$, $b(d_{OUT}^+) = 1$ and $b(d_{max}^+) = 0$ (because some direction from IN has been used once more).

(3) The algorithm will keep looping between steps (1)-(2) until the token reaches a vertex $v_s$, such that $s(v_s) \cap C_1^+ = \emptyset$. That is a vertex that corresponds to the sink of $A_i$. This will be evaluated in $1$ by Lemma 10. At that vertex, we have that $b(d) = 1$, for every $d \in IM_i$, by Property (iii) which holds inductively.

(4) The connecting frame for $v_s$ is $F_3$. No direction from IN is available and, thus, the algorithm will choose $+1$, first, and then $+3$ (only positive directions are available). The token is now at vertex $v_s \downarrow \[1\]$. In $\[1\]$ we have a uniform orientation with the sink at vertex $v_0 \downarrow \[B\]$. The token will move over all the imbalanced directions from $IM_i$, towards the sink of $\[B\]$ in exactly $|IM_i|$ steps. After this, $b(d) = 0$ for every $d \in IN$ and the token is at $v_0 \downarrow \[B\]$. The connecting frame is $F_2$ (since $v_0 \downarrow \[B\]$ is IN-saturated because IN is balanced). For the directions from OUT we have that $b(+1) = b(+3) = 0$ and $b(d) = 1$ for every other $d \in OUT$.

(5) In the next step, the algorithm will choose $+4$ and then $+5$ and the token will go to vertex $v_0 \downarrow \[8\]$. From there on, backwards edges will become again available. The token takes a path $\[8\] \leftrightarrow \[12\]$, according to $T_{i+1}$, where it ends up at vertex $v_0 \downarrow \[12\]$. The connecting frame is $F_2$ and the dashed edge is not available: $v_0 \downarrow \[12\] \leftrightarrow v_0 \downarrow \[1\]$. This means that $v_0 \downarrow \[12\]$ is also OUT-saturated and, thus, $D_{i+1}^+$-saturated. More specifically, we have $b(-3) = b(-4) = b(-5) = b(-6) = 1$ and for every other $d \in OUT, b(d) = 0$. The algorithm has used strictly less than $|OUT|$ directions from OUT, each one different to the other, and Property (ii) is satisfied up to this point.

$\triangleright$ The token is currently at vertex $v = v_0 \downarrow \[12\]$, which is such that $v \cap C_i^+ = v_0$. Moreover, IN is balanced. Since the algorithm is deterministic, all the steps that the token will take using directions from IN will be consistent with the path $P_i$. This is because using directions from OUT according to the tie-breaking rule $T_{i+1}$ it will always be in a face from the set $\{1, \ldots, 12\}$ and all these faces are oriented according to $A_i$.

(6) In the next step, the algorithm will use one direction $d \in IN, v_0 \downarrow \[12\] \xrightarrow{d} v$. The next vertex $v$ is not IN-saturated, by Lemma 10 and, thus, the connecting frame is $F_1$. The algorithm will use direction $-3$ and the token will go to $\[1\]$. So, now $b(-4) = b(-5) = b(-6) = 1$ and for every other $d \in OUT, b(d) = 0$; so, $b(d_{max}^+) = 0$.

(7) The token is at $\[1\]$. Directions from IN will be used, since they have priority in $T_{i+1}$. From OUT only negative directions are imbalanced but
no negative direction is available at $1$. After some steps, an $IN$-saturated vertex will be reached, by Property (ii) which holds for $A_i$ by the inductive hypothesis. Let us call this vertex $v_s$. When this happens, $b(d_{\text{max}}^{\text{OUT}}) = 1$ and $b(\text{OUT}, -4) = b(\text{OUT}, -5) = b(\text{OUT}, -6) = 1$. The directions from $\text{OUT}$ will be utilized and used in the order defined by $T_{i+1}$. The token will take a path $1 \rightarrow 12$. This is possible because the connecting frame for $v_s$ is $F_2$ and the dotted edges are available.

(8) When the token reaches $12$ we have $b(-3) = b(-4) = b(-5) = b(-6) = 1$ and for every other $d \in \text{OUT}$, $b(d) = 0$. The frame is still $F_2$ and the dashed edge is not available: $v_s \perp 12 \leftarrow v_s \perp 1$. This means that $v_s \perp 12$ is also $\text{OUT}$-saturated and, thus, it is $D_{i+1}^+$-saturated. We conclude that up to this point Property (ii) also holds for the new inductive step (the algorithm has used each direction from $\text{OUT}$ exactly once since the previous $D_{i+1}^+$-saturated vertex).

Then, the algorithm will use one direction from $IN$. The next vertex $v$ is not $IN$-saturated, by Lemma 10. Thus, the connecting frame for $v$ is $F_1$. Direction $-3$ will be used and the token will go to $1$, $b(d_{\text{max}}^{\text{OUT}}) = 1$.

(9) The algorithm will keep looping between steps (7)-(8) until the token reaches a vertex $v_{s_i}$ such that $s(v_{s_i}) \cap C_i^+ = \emptyset$. That is a vertex that corresponds to the sink of $A_i$. It will be evaluated in $1$ by Lemma 10.

(10) The token is on vertex $v_{s_i} \perp 1$; hence, the connecting frame is $F_3$. By the same arguments as for step (7), when the token reaches $v_{s_i}$ we have $b(d_{\text{max}}^{\text{OUT}}) = 1$ and $b(\text{OUT}, -4) = b(\text{OUT}, -5) = b(\text{OUT}, -6) = 1$. The token will take a path $v_{s_i} \perp 1 \rightarrow v_{s_i} \perp 12$. The global sink will be at $v_{s_i} \perp 12$. After the sink has been reached the balance for $\text{OUT}$ will be the same as the beginning of step (8) above. Namely, $b(-3) = b(-4) = b(-5) = b(-6) = 1$ and for every other $d \in \text{OUT}$, $b(d) = 0$. Thus, Property (ii) will be satisfied inductively. Finally, the global sink is $D_{i+1}^+$-saturated and the algorithm has used each direction from $\text{OUT}$ exactly once since the previous $D_{i+1}^+$-saturated vertex. Therefore, Property (ii) is satisfied.

With the above analysis, we have proved that the path $P_{i+1}$ will have length that is larger than twice the length of path $P_i$. Therefore, we obtain the recursion $T(n + 6) > 2T(n)$ which leads to the proof of Theorem 8.

6 Conclusions

In this paper, we have constructed AUSO on which the three pivot rules of interest will take exponentially long paths. Several interesting problems remain open: First and foremost is settling if Zadeh’s and Johnson’s rules admit exponential lower bounds even on linear programs. Moreover, it remains open to decide if Zadeh’s rule admits Hamiltonian paths on AUSO, a direction suggested by Aoshima et al. Finally, we are interested in exponential lower bounds for all the history-based rules that are discussed in [1]. We believe that our methods can be used to prove exponential lower bounds on AUSO for all of these rules.
References

[1] Y. Aoshima, D. Avis, T. Deering, Y. Matsumoto, and S. Moriyama. On the existence of Hamiltonian paths for history based pivot rules on acyclic unique sink orientations of hypercubes. *Discrete Applied Mathematics*, 160(15):2104 – 2115, 2012. doi:10.1016/j.dam.2012.05.023

[2] D. Avis and O. Friedmann. An exponential lower bound for cunningham’s rule. *Mathematical Programming*, pages 1–35, 2016. doi:10.1007/s10107-016-1008-4

[3] J. Balogh and R. Pemantle. The Klee-Minty random edge chain moves with linear speed. *Random Structures & Algorithms*, 30(4):464–483, 2007. doi:10.1002/rsa.20127

[4] W. H. Cunningham. Theoretical properties of the network simplex method. *Mathematics of Operations Research*, 4(2):196–208, 1979. doi:10.1287/moor.4.2.196

[5] J. Foniok, B. Gärtner, L. Klaus, and M. Sprecher. Counting unique-sink orientations. *Discrete Applied Mathematics*, 163, Part 2:155 – 164, 2014. doi:10.1016/j.dam.2013.07.017

[6] O. Friedmann. A subexponential lower bound for Zadeh’s pivoting rule for solving linear programs and games. In *IPCO 2011*, pages 192–206, 2011. doi:10.1007/978-3-642-20807-2_16

[7] O. Friedmann, T. D. Hansen, and U. Zwick. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In *STOC 2011*, pages 283–292, 2011. doi:10.1145/1993636.1993675

[8] B. Gärtner. A subexponential algorithm for abstract optimization problems. *SIAM J. Comput.*, 24(5):1018–1035, 1995. doi:10.1137/S0097539793250287

[9] B. Gärtner. The random-facet simplex algorithm on combinatorial cubes. *Random Structures & Algorithms*, 20(3):353–381, 2002. doi:10.1002/rsa.10034

[10] B. Gärtner, M. Henk, and G. M. Ziegler. Randomized simplex algorithms on Klee-Minty cubes. *Combinatorica*, 18(3):349–372, 1998. doi:10.1007/BF00999827.

[11] B. Gärtner and I. Schurr. Linear programming and unique sink orientations. In *SODA 2006*, pages 749–757, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109639.

[12] B. Gärtner and A. Thomas. The complexity of recognizing unique sink orientations. In *STACS 2015*, pages 341–353, 2015. doi:10.4230/LIPIcs.STACS.2015.341

[13] B. Gärtner and A. Thomas. The niceness of unique sink orientations. In *APPROX/RANDOM 2016*, pages 30:1–30:14, 2016. doi:10.4230/LIPIcs APPROX-RANDOM.2016.30.
[14] T. D. Hansen, M. Paterson, and U. Zwick. Improved upper bounds for random-edge and random-jump on abstract cubes. In *SODA 2014*, pages 874–881, 2014. doi:10.1137/1.9781611973402.65

[15] T. D. Hansen and U. Zwick. Random-edge is slower than random-facet on abstract cubes. In *ICALP 2016*, pages 51:1–51:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.51

[16] G. Kalai. A subexponential randomized simplex algorithm (extended abstract). In *STOC 1992*, pages 475–482, 1992. doi:10.1145/129712.129759

[17] V. Klee and G. J. Minty. How good is the simplex algorithm? *Inequalities III*, pages 159–175, 1972.

[18] J. Matoušek. Lower bounds for a subexponential optimization algorithm. *Random Structures & Algorithms*, 5(4):591–607, 1994. doi:10.1002/rsa.3240050408

[19] J. Matoušek. The number of unique-sink orientations of the hypercube*. *Combinatorica*, 26(1):91–99, 2006. doi:10.1007/s00493-006-0007-0

[20] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, 1996. doi:10.1007/BF01940877

[21] J. Matoušek and T. Szabó. RANDOM EDGE can be exponential on abstract cubes. *Advances in Mathematics*, 204(1):262 – 277, 2006. doi:10.1016/j.aim.2005.05.021

[22] I. Schurr and T. Szabó. Finding the sink takes some time: An almost quadratic lower bound for finding the sink of unique sink oriented cubes. *Discrete & Computational Geometry*, 31(4):627–642, 2004. doi:10.1007/s00454-003-0813-8

[23] I. Schurr and T. Szabó. Jumping doesn’t help in abstract cubes. In *IPCO 2005*, pages 225–235, 2005. doi:10.1007/11496915_17

[24] A. Stickney and L. Watson. Digraph models of Bard-type algorithms for the linear complementarity problem. *Math. Oper. Res.*, 3(4):322–333, 1978. doi:10.1287/moor.3.4.322

[25] T. Szabó and E. Welzl. Unique sink orientations of cubes. In *FOCS 2001*, pages 547–555, 2001. doi:10.1109/SFCS.2001.959931

[26] N. Zadeh. What is the worst case behavior of the simplex algorithm. *Polyhedral computation*, 48:131–143, 2009.