J Comb Optim (2012) 24:52–64
DOI 10.1007/s10878-010-9318-6

Online maximum directed cut

Amotz Bar-Noy · Michael Lampis

Published online: 2 April 2010

Abstract We investigate a natural online version of the well-known MAXIMUM DIRECTED CUT problem on DAGs. We propose a deterministic algorithm and show that it achieves a competitive ratio of $\frac{3\sqrt{3}}{2} \approx 2.5981$. We then give a lower bound argument to show that no deterministic algorithm can achieve a ratio of $\frac{3\sqrt{3}}{2} - \epsilon$ for any $\epsilon > 0$ thus showing that our algorithm is essentially optimal. Then, we extend our technique to improve upon the analysis of an old result: we show that greedily derandomizing the trivial randomized algorithm for MAXDICUT in general graphs improves the competitive ratio from 4 to 3, and also provide a tight example.

Keywords MAXDICUT · Online algorithms · DAG

1 Introduction

The MAXIMUM CUT and MAXIMUM DIRECTED CUT problems are among the most famous and widely studied problems in computer science and during the past few decades countless papers have been devoted to them. Their objective is to partition the vertices of an edge-weighted graph so that the weight of the edges going from one side of the partition to the other is maximized. Both problems have attracted considerable interest from the research community both because of their theoretical importance but also because of their numerous applications in fields such as network design, VLSI design and statistical physics (see for example Poljak and Tuza 1995).

A. Bar-Noy · M. Lampis (✉)
Doctoral Program in Computer Science, Graduate Center, City University of New York,
365 5th Avenue, New York, NY 10016, USA
e-mail: mlampis@gc.cuny.edu

A. Bar-Noy
e-mail: amotz@sci.brooklyn.cuny.edu

MAXIMUM CUT was one of the 21 original problems shown to be NP-complete in Karp's seminal paper (Karp 1972). Since then, many results have appeared in the literature dealing with restrictions of the problem; for example in Papadimitriou and Yannakakis (1991) it is shown that the problem remains NP-hard for graphs of maximum degree 3, while in Hadlock (1975) it is shown that it is solvable in polynomial time for planar graphs. The approximability of both problems has also been well-studied. It has long been known that MAXCUT and MAXDICUT can be approximated within factors of 2 and 4 respectively with a trivial randomized algorithm which randomly places each vertex on either side of the partition with equal probability (see e.g. Alon and Spencer 2004). The celebrated paper of Goemans and Williamson (1995) achieves an approximation ratio of 1.1383 for MAXCUT using semi-definite programming and improving on its results a ratio of 1.165 is achieved for MAXDICUT in Feige and Goemans (1995). The ratio for MAXCUT is shown to be best possible under the Unique Games Conjecture in Khot et al. (2004). In addition to these results, several other results using combinatorial algorithms are known for special cases of MAXCUT (see for example Bazgan and Tuza 2008).

MAXDICUT is the less studied of the two problems, especially from the point of view of combinatorial algorithms. One exception is the paper by Halperin and Zwick (2001) which takes a combinatorial approach to the problem and gives a 2-approximation algorithm via a reduction to matching and a 2.22-approximation algorithm which runs in linear time. Few other combinatorial results are known for this problem. In fact, the restriction of MAXDICUT to DAGs was only recently shown to be NP and APX-hard in Lampis et al. (2008).

To the best of our knowledge, neither problem has been studied before in an online setting and the only algorithms applicable in this case are the folklore trivial randomized algorithms. In this paper we propose the study of MAXDICUT as an online problem motivated by several reasons: first is the general observation that studying a problem in an online setting can often lead us to discover some unknown combinatorial structure which we can then exploit, either in an online or an offline setting. This is especially true in the case of MAXDICUT since the vast majority of known results rely on complex tools such as semi-definite programming, while ignoring the more combinatorial aspect of the problem.

Our main motivation though is that, for MAXDICUT in particular, a very natural class of greedy heuristics is very likely to be used in practice: algorithms which make a single pass over the input and decide on which side of the partition to place each vertex relying on information from previous vertices and local considerations, such as the total weight of incoming and outgoing edges. Thus, analyzing the performance of such greedy heuristics is an interesting problem and one of our main aims in this paper. Posing this as an online problem in an appropriate model will also allow us to argue about lower bounds which apply to this class of algorithms.

It should be clarified at this point what we mean by an online setting, since one could potentially define countless online variations of MAXDICUT. The model we propose is one where the vertices are revealed one by one. Along with each vertex, all edges incident to it whose other endpoint has already been revealed are also revealed. This description is very natural. However, if this is all the information the adversary has to reveal, it is easy to force any deterministic algorithm to have unbounded competitive ratio thus making the problem uninteresting (more on this in Sect. 2). Also,

any reasonable (offline) greedy heuristic for the problem would take other information into account as well, such as the total degree of the vertex, so this online setting does not allow us to use the offline heuristics for which we want to prove upper and lower bounds. This is why we add an extra restriction: we force the adversary to reveal along with each vertex the *total* weight of its incoming and its outgoing edges. We believe that this model strikes a good balance: it only gives the algorithm a little information about the future, just enough to make the problem reasonable. It also fits well with our motivation from greedy single-pass algorithms, since in such cases one would likely make decisions based on past decisions and local information, such as the degree of a vertex.

Furthermore, motivated by the result of Lampis et al. (2008) mentioned above, we investigate the problem restricted to DAGs. The problem's own structure leads us to impose one further restriction on the adversary: we require the vertices to be revealed in an order consistent with the partial ordering implied by the DAG itself. This can be an interresting restriction consistent with our above motivation since this is probably the most natural order in which a single-pass greedy algorithm would examine the vertices of the DAG.

Our main results regarding the problem on DAGs are an algorithm with competitive ratio $\frac{3\sqrt{3}}{2} \approx 2.5981$ and an essentially matching lower bound, which also applies in the case of general graphs. Thus, it follows from our results that $\frac{3\sqrt{3}}{2}$ is the best ratio achievable for this problem. What is also interesting is that the intuition gained from our analysis on DAGs is then applied in the general case of the problem to give a deterministic 3-competitive algorithm. In fact, this can be interpreted as an improvement in the analysis of an old result: our algorithm can be seen as a derandomization of the trivial randomized algorithm and we show that this derandomization improves the competitive ratio from 4 to 3. To the best of our knowledge, this result was not known before, even concerning the analysis of this derandomized algorithm in the offline case (for example in Goemans and Williamson 1995 it is mentioned that the best previously known algorithm for MAXDICUT has a ratio of 4), and it is interesting to consider its contrast with the undirected case where derandomizing the trivial algorithm does not improve its guarantee.

The rest of this paper is structured as follows: in Sect. 2 we give the precise definition of the problem and the online setting we will consider. In Sect. 3 we give the online algorithm for DAGs, while in Sect. 4 we give the lower bound. In Sect. 5 we give the 3-competitive algorithm for general graphs and finally in Sect. 6 we give some discussion and directions to further work.

2 Definitions and preliminaries

In the remainder we consider only directed graphs, so we will use the terms graph and digraph interchangeably. Let us give the definition of MAXDICUT.

Definition 1 Given a digraph $G(V, E)$ and a weight function on the edges $w : E \to \mathbb{N}$, MAXDICUT is the problem of finding a partition of V into two sets V_0 and V_1 so that the weight of the edge set $C = \{(u, v) \mid u \in V_0, v \in V_1\}$ is maximized. That is, the objective is to maximize $\sum_{e \in C} w(e)$.

The restriction of the problem where all edges have the same weight is often called cardinality or unweighted MAXDICUT. In this paper we focus on the weighted version. Because we focus on the weighted problem when we refer to the in-degree (resp. out-degree) of a vertex, we mean the total weight of its incoming (resp. outgoing) edges. Also when we refer to the size of a cut, we mean the total weight of the edges in the cut.

When a vertex u is placed in V_0 (resp. V_1) we will often say that 0 (resp. 1) was assigned to u. We denote by $w_{\text{in}}(u)$ the total weight of incoming edges at vertex u and by $w_{\text{out}}(u)$ the total weight of outgoing edges. By $w^0_{\text{in}}(u)$ (resp. $w^1_{\text{in}}(u)$) we denote the incoming edges of u whose other endpoint has been assigned 0 (resp. 1). Similarly for $w^0_{\text{out}}(u)$ and $w^1_{\text{out}}(u)$. In a context where the solution produced by an algorithm is compared with another (optimal) solution, when we use this notation we refer to the algorithm's choices, unless otherwise specified. That is, $w^0_{\text{in}}(u)$ means the total weight of edges coming to u from vertices to which *the algorithm* assigned 0. Some extra shorthand notation relevant in the case of our algorithm for general graphs is defined in Sect. 5.

The online model

Definition 2 In the *online* MAXDICUT problem, an adversary chooses a graph $G(V, E)$ and an ordering of the vertices of V. Then, for each vertex u in this order the adversary reveals to the algorithm $w_{\text{in}}(u)$, $w_{\text{out}}(u)$ and the weights of edges connecting u to previously revealed vertices. The algorithm then makes an irrevocable decision to assign 0 or 1 to u and the process continues, until all of G has been revealed.

Perhaps the detail of this definition which needs justification is why we demand that the adversary reveal the total in-degree and out-degree each vertex will have in the *final* graph G, thus revealing some information about the future. Aside from practical considerations (it could be argued that this is a more realistic model for some applications) the main reason is that not revealing any such information makes the problem impossible. Consider for example the setting where the adversary does not reveal such information. So, when the first vertex is given the algorithm must assign a value. If it picks 0 the adversary is free to claim later that this vertex is a sink in G with many heavy edges going into it, which are lost from the solution. Similarly, the adversary can respond to a 1 by revealing that the vertex is a source. This simple trick would be enough to make the competitive ratio of any deterministic algorithm tend to ∞. That is why revealing $w_{\text{in}}(u)$ and $w_{\text{out}}(u)$ are necessary.

In the case of DAGs we restrict the setting a little bit more. Recall that the edges of a DAG imply a partial ordering on its vertices. We demand that the adversary must reveal the vertices of G in a way consistent with this partial ordering. This eliminates the need to reveal $w_{\text{in}}(u)$, since when this restriction is followed the tails of all edges incoming to u must be revealed before u, therefore the algorithm already has enough information to calculate $w_{\text{in}}(u)$.

The motivation behind this restriction is once again single-pass algorithms. In the case of DAGs it is quite likely that one would scan through the graph in the order

Algorithm 1 Weighted comparison online algorithm

When a vertex u is revealed, compare $w_{\text{out}}(u)$ to $cw^0_{\text{in}}(u)$.

- If $w_{\text{out}}(u) > c \cdot w^0_{\text{in}}(u)$ then assign 0 to u.
- Otherwise, assign 1 to u.

dictated by the directed edges. Moreover, this restriction seems natural and may be interesting in its own right. Of course it should also be noted that, as our results point out, this restriction does not have a huge impact on the problem since we show a lower bound for this special case which is not very far from the competitiveness of our algorithm even in the general case.

3 An online algorithm for MaxDiCut on DAGs

In this section we describe an algorithm for MaxDiCut on DAGs which can be applied in the online setting described in Sect. 2. Recall once again that in our setting the vertices are revealed one by one in the ordering implied by the DAG and we are also informed of the out-degree of each vertex.

We present an algorithm (Algorithm 1) based on a weighted comparisons idea, leaving a parameter $c > 1$ to be fixed later to a value that will achieve the best competitive ratio. After this optimization we will show a ratio of $\frac{3\sqrt{3}}{2} < 2.5981$.

It is not hard to understand the motivation behind this algorithm, if one considers the naive greedy algorithm which at every vertex compares $w^0_{\text{in}}(u)$, which is the payoff that would be obtained by assigning 1, with $w_{\text{out}}(u)$, which is the potential payoff of assigning 0. An adversary could easily fool this naive algorithm into assigning a long string of consecutive 0s in a path, making its competitive ratio tend to ∞. However, in the algorithm we propose, we only choose to assign 0 to a vertex when tempted by a payoff much larger (c times larger) than what could be obtained by assigning 1. Even though the adversary can still fool the algorithm into assigning a long string of consecutive 0s, it will have to offer exponentially increasing edge weights to do so. Eventually, a vertex will be encountered where the algorithm assigns 1, and the profit obtained by doing so will (at least partially) outweigh the loss of edges that occured up to that point.

Let us now provide a formal analysis of the algorithm's performance to establish its competitive ratio. Suppose we are given a graph $G(V, E)$, and let (V_0, V_1) be the partition of V decided by the algorithm. Let $E_{ij} = \{(u, v) \in E \mid u \in V_i, v \in V_j\}$. Thus, the cut produced by the algorithm has weight $\text{SOL} = \sum_{e \in E_{01}} w(e)$.

Consider the two subgraphs of G, $G_0(V_0, E_{00})$ and $G_1(V, E_{01} \cup E_{10} \cup E_{11})$. Let OPT be the size of an optimal cut of G and OPT_0 (resp. OPT_1) the size of an optimal cut of G_0 (resp. G_1).

Lemma 1 $\text{OPT} \leq \text{OPT}_0 + \text{OPT}_1$.

We will compare SOL independently to OPT_0 and OPT_1. First, let us compare it to OPT_1.

Lemma 2 $c\text{SOL} \geq \text{OPT}_1$.

Proof Suppose that the vertices in V are numbered $1, \ldots, n$, in the order in which they were revealed. Starting with the optimal solution, we will gradually apply changes to make it equal to the solution produced by the algorithm, at each step "bribing" the optimal solution to change its assignment. In the end we will bound the amount we used in bribing to prove the lemma.

Consider a sequence of cuts OPT_1^k of the graph G_1 where $0 \leq k \leq n$, n being the number of vertices. OPT_1^k is defined as the assignment that is identical to SOL for vertices $1, \ldots, k$ and identical to OPT_1 for the remaining vertices. Thus, $\text{OPT}_1^0 \equiv \text{OPT}_1$ and $\text{OPT}_1^n \equiv \text{SOL}$.

We can now prove the invariant $\text{OPT}_1^k + (c-1)\sum_{u \in V_1 \cap \{1,\ldots,k\}} w_{\text{in}}^0(u) \leq \text{OPT}_1^{k+1} + (c-1)\sum_{u \in V_1 \cap \{1,\ldots,k+1\}} w_{\text{in}}^0(u)$. If vertex $k+1$ is in V_0, then we need to prove that $\text{OPT}_1^k \leq \text{OPT}_1^{k+1}$. But vertex $k+1$ has no incoming edges from another vertex in V_0 (recall that we are talking about G_1). Also, because OPT_1^k agrees with SOL on the first k vertices, any edge incoming to $k+1$ must have its other endpoint assigned 1 by OPT_1^k, therefore, it can not be included in the cut. Thus, assigning 0 to vertex $k+1$ will not decrease the size of OPT_1^k and the inequality follows. Now, if vertex $k+1$ is in V_1, we need to prove that $\text{OPT}_1^{k+1} + (c-1)w_{\text{in}}^0(k+1) \geq \text{OPT}_1^k$. If OPT_1 assigns 1 to $k+1$ then $\text{OPT}_1^{k+1} \equiv \text{OPT}_1^k$ and the inequality follows trivially. If, on the other hand, OPT_1 assigns 0 to $k+1$, then changing this to 1 will contribute $w_{\text{in}}^0(k+1)$ but might remove up to $w_{\text{out}}(k+1)$ from the cut. Thus, bribing the solution at this point with an amount of $w_{\text{out}}(k+1) - w_{\text{in}}^0(k+1)$ is enough to make it change its assignment. But $w_{\text{out}}(k+1) - w_{\text{in}}^0(k+1) \leq (c-1)w_{\text{in}}^0(k+1)$ because the algorithm assigned 1 to $k+1$ and the inequality follows.

Now observe that by using the above sequence of inequalities we have shown that $\text{OPT}_1^0 \leq \text{OPT}_1^n + (c-1)\sum_{u \in V_1} w_{\text{in}}^0(u)$, or equivalently $\text{OPT}_1 \leq \text{SOL} + (c-1)\text{SOL}$. □

Now, we must compare OPT_0 and SOL. First we need to show an auxiliary lemma.

Lemma 3 $\text{SOL} \geq (c-1)\sum_{e \in E_{00}} w(e)$.

Proof For every vertex $u \in V_0$ we have $cw_{\text{in}}^0(u) < w_{\text{out}}(u)$. Summing over all vertices in V_0 we have $c\sum_{e \in E_{00}} w(e) < \sum_{e \in E_{00}} w(e) + \sum_{e \in E_{01}} w(e)$, from which the lemma follows. □

Lemma 4 $\text{OPT}_0 \leq \frac{c}{c^2-1}\text{SOL}$.

Proof Consider a vertex $u \in V_0$ to which OPT_0 assigns 1. Its outgoing edges are lost from the cut. Since u was assigned 0 by the algorithm we know that $w_{\text{out}}(u) > cw_{\text{in}}^0(u) \Rightarrow w_{\text{out}}^0(u) > cw_{\text{in}}^0(u) - w_{\text{out}}^1(u)$. Recall that in G_0, the edges from V_0 to V_1 are not included, therefore $w_{\text{out}}^0(u)$ is the amount lost. Now, OPT_0 is upper bounded by the total edge weight in E_{00} minus the total weight of outgoing edges from vertices to which it assigned 1. If we denote the set of vertices to which 1

was assigned by OPT_0 as S_1 we have $\mathrm{OPT}_0 \leq \sum_{e\in E_{00}} w(e) - \sum_{u\in S_1} w^0_{\mathrm{out}}(u) < \sum_{e\in E_{00}} w(e) - c\sum_{u\in S_1} w^0_{\mathrm{in}}(u) + \sum_{u\in S_1} w^1_{\mathrm{out}}(u)$. But $\sum_{u\in S_1} w^0_{\mathrm{in}}(u) \geq \mathrm{OPT}_0$. Also, $\sum_{u\in S_1} w^1_{\mathrm{out}}(u) \leq \sum_{u\in V_0} w^1_{\mathrm{out}}(u) = \mathrm{SOL}$. Therefore, we have that $(c+1)\mathrm{OPT}_0 < \sum_{e\in E_{00}} w(e) + \mathrm{SOL}$. The result follows using Lemma 3. □

Theorem 1 *Algorithm 1 is* $(c + \frac{c}{c^2-1})$*-competitive.*

Proof Combining the results of Lemmata 1, 2 and 4. □

Taking the derivative of $\frac{c}{c^2-1} + c$ we find that a minimum is obtained for $c = \sqrt{3}$. In that case the competitive ratio achieved is $\frac{3\sqrt{3}}{2}$.

The fact that the above analysis is tight for $c = \sqrt{3}$ will be confirmed in Sect. 4 where it will be shown that this is in fact the best ratio achievable by any algorithm. But before that, we could also give a simple example to show that our analysis is tight for any c. First, let us point out that when looking for such an example we may assume without loss of generality that in cases where $w^0_{\mathrm{in}}(u) = cw_{\mathrm{out}}(u)$ the algorithm assigns to u an arbitrary value chosen by the adversary. This is because the adversary could easily adjust $w_{\mathrm{out}}(u)$ by $\pm\epsilon$ without affecting the size of the optimal solution significantly.

Consider a directed path with $2n$ edges and the vertices labeled $0, 1, \ldots, 2n$. We give the edges weights $w((i, i+1)) = c^i$. The algorithm assigns 0 to vertex 0 and then without loss of generality assigns 0 to all vertices until $2n-2$. Then it assigns 1 to $2n-1$ and $2n$. The produced solution has weight $w((2n-2, 2n-1)) = c^{2n-2} = (c^2)^{n-1}$. The optimal solution has weight $c^{2n-1} + c^{2n-3} + c^{2n-5} + \cdots + c = c\frac{(c^2)^n - 1}{c^2-1} = \frac{c^3}{c^2-1}(c^2)^{n-1} - \frac{c}{c^2-1}$.

4 A $3\sqrt{3}/2 - \epsilon$ lower bound

In this section we show that the competitive ratio of $3\sqrt{3}/2$ achieved by the algorithm of Sect. 3 is essentially the best possible. We show this by giving a strategy which the adversary can follow to force any deterministic algorithm's competitive ratio arbitrarily close to $3\sqrt{3}/2$.

The construction we use is very simple: it is a directed path, similar to the tight examples of the previous section. However, now the edge weights are picked so that any algorithm can be defeated. The adversary's strategy is still to fool the algorithm into assigning a long string of 0s thus worsening the competitive ratio. But now the edge weights will eventually converge. We make use of the following simple observation:

Lemma 5 *If for some vertex* $w^0_{\mathrm{in}}(u) \geq w_{\mathrm{out}}(u)$ *then it is optimal to assign* 1 *to that vertex.*

The proof is trivial: assigning 1 will immediately yield a payoff greater than the potential payoff which could be obtained by assigning 0.

The adversary must find a sequence of edge weights for the path, call them $w_0, w_1, \ldots$, such that assigning a long string of 0s is inevitable for any algorithm with a good competitive ratio. If the adversary can also make the sequence of weights decrease at some point without violating this principle the previous observation will complete the proof of a lower bound.

The above methodology is illustrated in the following lemma.

Lemma 6 *For a given number $\lambda > 1$, if for the sequence defined by $w_0 = 1, w_1 = \lambda, w_2 = \lambda^2 - 1$ and $w_i = \lambda(w_{i-1} - w_{i-3})$ there exists k such that $w_{k+1} \leq w_k$, then no deterministic online algorithm can achieve a ratio less than λ for* MaxDiCut *on DAGs.*

Proof Suppose that for some i we have $w_i \leq w_{i-1}$. The adversary will follow this strategy: the graph used is a directed path with edge weights $w_0, w_1, \ldots$. While the algorithm has not yet assigned a 1 keep revealing vertices and set edge weights according to the sequence w_i. When the algorithm assigns 1 or when the edge with weight w_i is revealed, terminate the instance, that is reveal only the other endpoint of the single outstanding edge and inform the algorithm that this is the last vertex.

For this strategy to demonstrate a lower bound of λ it is sufficient to have that for all k $w_k + w_{k-2} + \cdots \geq \lambda w_{k-1}$. The left-hand side of this equation is the optimal solution if we terminate after edge k (we will denote this by OPT_k), while w_{k-1} is the algorithm's solution in that case (observe that we terminate on the algorithm's first 1 so the algorithm always picks exactly one edge in the cut). Therefore, if this inequality holds for all k the competitive ratio will be at least λ independent of the point where the instance was terminated.

Clearly, $\mathrm{OPT}_1 = w_1 = \lambda w_0$ and $\mathrm{OPT}_2 = w_2 + w_0 = \lambda w_1$. Now, we can use induction and we have $\mathrm{OPT}_k = w_k + \mathrm{OPT}_{k-2} = \lambda(w_{k-1} - w_{k-3}) + \mathrm{OPT}_{k-2} \geq \lambda(w_{k-1} - w_{k-3}) + \lambda w_{k-3} = \lambda w_{k-1}$. Therefore, if the instance terminates at any point, the competitive ratio is at least λ. Lemma 5 guarantees that the instance will be terminated at i at the latest for any reasonable algorithm. □

The question is for which λ the sequence w_i decreases at some point and for which it is always increasing. In order to answer this we would like to obtain a closed form of w_i. One way to obtain such a form is to solve the corresponding equation $x^3 = \lambda(x^2 - 1)$. If the roots of this equation are distinct, say r_1, r_2, r_3 we get that $w_i = A_1 r_1^i + A_2 r_2^i + A_3 r_3^i$ for some constants A_1, A_2, A_3 determined by w_0, w_1, w_2.

Lemma 7 *If $\lambda < \frac{3\sqrt{3}}{2}$ the equation $x^3 = \lambda(x^2 - 1)$ has three distinct roots. One of them is real and belongs in $(-1, 0)$ and the other two are complex.*

Proof Take the discriminant of the equation $\Delta = 4\lambda^4 - 27\lambda^2$. If $\lambda < \frac{3\sqrt{3}}{2}$ then $\Delta < 0$ and the equation has two complex and one real root. Take the function $f(x) = x^3 - \lambda x^2 + \lambda$. We have $f(0) = \lambda > 0$ and $f(-1) = -1 < 0$ therefore the real root of the equation is in $(-1, 0)$. □

Lemma 8 *If $\lambda < \frac{3\sqrt{3}}{2}$ there exists an i such that for the sequence w defined in Lemma 6 we have $w_i \leq w_{i-1}$.*

Proof Let r be the real root and z_1, z_2 the complex roots of the equation of Lemma 7. z_1 and z_2 must be conjugates, i.e. $z_1 + z_2$ is a real number. We can rewrite them as $z_1 = |z|e^{\mathbf{i}\theta}$ and $z_2 = |z|e^{-\mathbf{i}\theta}$. Also note that, as $\lambda \to \frac{3\sqrt{3}}{2}^-$ we have that $\theta \to 0$, in other words, as λ approaches its critical value the two complex roots tend closer to becoming one double root.

A closed form for the sequence w will be of the form $w_n = A_1 z_1^n + A_2 z_2^n + A_3 r^n$. To calculate the constants A_1, A_2, A_3 we could use the known values of w_0, w_1, w_2. However, since we only need to prove that w decreases at some point, this is not necessary. It suffices to observe the following facts:

1. The term $A_3 r^n$ tends to 0 as n tends to infinity, because we know that $|r| < 1$.
2. Since $\forall n$, w_n is a real number, it must be the case that A_1 and A_2 are also conjugates. We can rewrite them as $A_1 = |A|e^{\mathbf{i}\phi}$ and $A_2 = |A|e^{-\mathbf{i}\phi}$.
3. Therefore, we have $w_n = |A| \cdot |z|^n (e^{\mathbf{i}(n\theta+\phi)} + e^{-\mathbf{i}(n\theta+\phi)}) + o(1)$.

Using the above we have $w_n = 2|A| \cdot |z|^n (\cos(n\theta + \phi)) + o(1)$, which is not an always increasing function. □

Using Lemmata 6 and 8 we have the following result.

Theorem 2 *No deterministic algorithm can achieve a competitive ratio of $\frac{3\sqrt{3}}{2} - \epsilon$ for any $\epsilon > 0$ for the* MaxDiCut *problem on DAGs.*

5 General graphs

In this section we show that a natural extension of the online algorithm for DAGs of Sect. 3 results in a 3-competitive online algorithm for the case of general digraphs.

Before we go on, let us first introduce some additional notation, which will be needed in this case. Recall that in Sect. 3 we used the notation $w_{\text{in}}(u)$, $w_{\text{out}}(u)$ to refer to the total weight of edges incoming and outgoing from u respectively. However, the online model we assumed in that case guaranteed that at the time when a vertex u was revealed, the tails of its incoming edges had already been revealed (and therefore assigned a value). Similarly, we knew that none of the heads of edges coming out of u had been revealed yet at the time u was given.

In the online model for general graphs this is not necessarily the case. However, we now make the assumption that whenever the adversary reveals a vertex, both its total in-degree and its total out-degree are revealed. Therefore, we introduce a shorter notation as follows: for a given vertex u and a specific assignment to all the vertices revealed before u we will denote by $C_1(u)$ the certain payoff which can be achieved by assigning 1 to this vertex. In other words, $C_1(u)$ is the total weight of edges incoming to u with their tails already revealed and assigned 0. Similarly, we denote by $C_0(u)$ the certain payoff of assigning 0, which is the total weight of edges coming out of u whose heads have already been assigned 1. We denote by $P_0(u)$, $P_1(u)$ the

Algorithm 2 Doubling online algorithm for general graphs

When u is revealed calculate $C_0(u)$, $C_1(u)$, $P_0(u)$, $P_1(u)$.

- If $C_0(u) + \frac{P_0(u)}{2} > C_1(u) + \frac{P_1(u)}{2}$ assign 0 to u.
- Otherwise, assign 1 to u.

maximum additional payoff which can be achieved by assigning 0 or 1 respectively to u. That is, $P_0(u)$ is the total weight of edges outgoing from u to vertices not yet revealed, and $P_1(u)$ is the total weight of edges incoming to u from such vertices.

Our proposed algorithm is Algorithm 2. It is clear that using this notation we would have in the online model for DAGs of Sect. 3 that for all u, $P_0(u) = w_{\text{out}}(u)$, $P_1(u) = 0$, $C_1(u) = w^0_{\text{in}}(u)$, $C_0(u) = 0$. This new notation will ease the analysis of our algorithm for general graphs.

Following our previous remark that in the model for DAGs we have for all u, $C_0(u) = P_1(u) = 0$ it is easy to see that, if restricted to DAGs, Algorithm 2 is exactly Algorithm 1 with $c = 2$ (which implies that it would have a ratio of $8/3$ for that special case). Moreover, the intuition is essentially the same: give twice as much weight to a certain payoff as you give to an uncertain one. The question is, what is this algorithm's competitive ratio for general graphs?

First, let us point out that Algorithm 2 is at most 4-competitive. This can be seen if we compare its performance to the trivial randomized algorithm.

Theorem 3 *Let* SOL *be the solution produced by Algorithm 2 for a graph* $G(V, E)$. *Then* $\text{SOL} \geq \frac{|E|}{4}$.

Proof Consider the randomized algorithm which assigns 0 or 1 to every vertex of G with probability $1/2$. The expected size of the solution produced by this algorithm is exactly $\frac{|E|}{4}$. Now, derandomize this algorithm using the method of conditional expectations. As each vertex u is revealed pick the assignment that will maximize the expected size of the produced solution if the remaining vertices were assigned at random with probability $1/2$. Obviously, only edges incident on u are affected, so maximize the expected total weight of edges incident to u that will be included in the cut. Assigning 0 includes a total weight of C_0 with probability 1 and a weight of P_0 with probability $1/2$, while the rest of the edges incident on u are lost. Similarly for assigning 1.

Therefore, Algorithm 2 picks for each vertex the assignment which would maximize the expected cut if we went on randomly for the remaining vertices. This implies that the algorithm's solution is lower bounded by the expected size of the solution produced by the randomized algorithm, which is $\frac{|E|}{4}$. □

A competitive ratio of 4 immediately follows from Theorem 3. We will now show that this can actually be improved to a ratio of 3.

Theorem 4 *Algorithm 2 is* 3*-competitive.*

Proof Let SOL be the cut produced by the algorithm and OPT an optimal cut. Once again we will gradually change OPT to SOL by bribing the adversary to change the assignment for vertices on which OPT and SOL disagree. Then we will bound the amount of bribing needed.

Number the vertices $1, 2, \ldots, n$ in the order in which they were revealed. Let $\mathrm{OPT}_0, \mathrm{OPT}_1, \ldots, \mathrm{OPT}_n$ be a sequence of cuts defined as follows: OPT_i gives the same assignment as SOL to the first i vertices and the same assignment as OPT to the others. Therefore, $\mathrm{OPT}_0 \equiv \mathrm{OPT}$ and $\mathrm{OPT}_n \equiv \mathrm{SOL}$.

Suppose that OPT and SOL agree on the assignment of some vertex i. Then $\mathrm{OPT}_{i-1} \equiv \mathrm{OPT}_i$.

Suppose that for some vertex i the optimal cut assigns 0 while the algorithm assigned 1. Consider then the cut OPT_{i-1}. From the edges incident on i the cut OPT_{i-1} gets at most $C_0(i) + P_0(i)$. The cut OPT_i gets at least $C_1(i)$. The cuts OPT_{i-1} and OPT_i differ only in vertex i therefore we have $\mathrm{OPT}_{i-1} \leq \mathrm{OPT}_i + C_0(i) + P_0(i) - C_1(i)$. But the algorithm assigned 1 to i therefore, $C_0(i) + \frac{P_0(i)}{2} \leq C_1(i) + \frac{P_1(i)}{2} \Rightarrow C_0(i) - C_1(i) \leq \frac{P_1(i)-P_0(i)}{2}$. Using this we get that $\mathrm{OPT}_{i-1} \leq \mathrm{OPT}_i + \frac{P_0(i)+P_1(i)}{2}$.

Using similar arguments we can prove that in the symmetric case where the optimal cut assigns 1 and the algorithm assigns 0 we again have $\mathrm{OPT}_{i-1} \leq \mathrm{OPT}_i + \frac{P_0(i)+P_1(i)}{2}$. Therefore, we have $\mathrm{OPT} = \mathrm{OPT}_0 \leq \mathrm{OPT}_n + \sum_{i=1}^{n} \frac{P_0(i)+P_1(i)}{2}$.

However, for a vertex i we have that $P_0(i) + P_1(i)$ is the total weight of edges whose first endpoint to be revealed is i. Therefore, taking the sum of $P_0(i) + P_1(i)$ for all vertices gives us exactly $|E|$, since then every edge's weight is counted exactly once (on its endpoint which was revealed first). Using also Theorem 3 we have $\mathrm{OPT} \leq \mathrm{SOL} + \frac{|E|}{2} \leq 3\mathrm{SOL}$ □

Even though it has been known for decades that the trivial randomized algorithm is 4-competitive, to the best of our knowledge this is the first time that it has been examined whether its greedy derandomization actually offers an improved competitive ratio. We prove that it does, even in an online setting. What makes this result more interesting is that the situation is different in the undirected case. There, the trivial randomized algorithm is 2-competitive but the derandomized version is also 2-competitive and this is tight. (For example consider the unweighted graph $K_{2,n}$ and add an edge between the two vertices of the small part. Now, if the two vertices of the small part are examined first they will be placed on different sides of the partition and the solution will have $n + 1$ edges in the end while the optimal is $2n$.)

It should also be clear that, even though Algorithm 2 is a derandomization of the trivial algorithm, Theorem 4 does not imply that the randomized algorithm is also 3-competitive (it can be seen that it is in fact tightly 4-competitive by taking a bipartite graph $G(V_1, V_2, E)$ with all edges oriented from V_1 to V_2, so that $\mathrm{OPT} = |E|$). Quite interestingly, this is a case where derandomizing really helps improve the guarantee on the algorithm's performance.

Finally, let us point out that there is a simple example which demonstrates that the analysis of Algorithm 2 is tight.

Theorem 5 *There exists a graph $G(V, E)$ for which the results of Theorems 3 and 4 are tight.*

Proof First, observe that without loss of generality we can assume that when for some vertex $C_0(u) + \frac{P_0(u)}{2} = C_1(u) + \frac{P_1(u)}{2}$ then the algorithm assigns to u some arbitrary value chosen by the adversary. This is because simply by adding a small ϵ to P_0 or P_1 the adversary can make the algorithm behave in either way.

Now, the graph G we will use is a directed path on four vertices P_4. We label the vertices $1, 2, 3, 4$ and set $w((1,2)) = w((2,3)) = 1$ and $w((3,4)) = 2$. The order in which the vertices are presented is $2, 1, 3, 4$. When 2 is revealed, without loss of generality, the algorithm assigns 0. So, the edge $(1, 2)$ is lost independent of the choice for 1. When 3 is revealed we have $C_0(3) = P_1(3) = 0$ and $C_1(3) = \frac{P_0(3)}{2}$ so again without loss of generality the algorithm assigns 1 and the edge $(3, 4)$ is lost. Now SOL $= 1$ but OPT $= 3$ and $|E| = 4$. □

6 Conclusions and further work

In this paper we introduced a natural online setting for the study of MAXDICUT and its restriction to DAGs. We completely solved the problem in DAGs for deterministic algorithms by providing an algorithm and an essentially matching lower bound. In addition, and perhaps more interestingly, we showed that the intuition gained from this problem can help in the general case, by improving a folklore result concerning the approximation ratio of the basic greedy algorithm for general graphs.

Many other topics are worth considering. For the specific problem on DAGs it would be interesting to consider randomized algorithms against an oblivious adversary. This would likely defeat our lower bound but it is not clear what would be a better algorithm. For the general case, there is a small gap between the competitiveness of our algorithm and the lower bound for DAGs. Could this gap be closed? Finally, it would be interesting to see if and how any of the ideas of this paper can be applied in the undirected case of the problem.

Acknowledgement We would like to thank Valia Mitsou for stimulating discussion and for proofreading an earlier draft of this paper.

References

Alon N, Spencer JH (2004) The probabilistic method. Wiley, New York

Bazgan C, Tuza Z (2008) Combinatorial 5/6-approximation of max cut in graphs of maximum degree 3. J Discrete Algorithms 6(3):510–519

Feige U, Goemans M (1995) Approximating the value of two power proof systems, with applications to MAX 2SAT and MAX DICUT. In: Proceedings of the third Israel symposium on the theory of computing and systems, pp 182–189

Goemans MX, Williamson DP (1995) Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J ACM 42(6):1115–1145

Hadlock F (1975) Finding a maximum cut of a planar graph in polynomial time. SIAM J Comput 4:221

Halperin E, Zwick U (2001) Combinatorial approximation algorithms for the maximum directed cut problem. In: Proceedings of the twelfth annual ACM-SIAM symposium on discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, pp 1–7

Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW (eds) Complexity of computer computations. Plenum, New York, pp 85–103

Khot S, Kindler G, Mossel E, O'Donnell R (2004) Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? In: Foundations of computer science, 2004. Proceedings. 45th annual IEEE symposium, pp 146–154

Lampis M, Kaouri G, Mitsou V (2008) On the algorithmic effectiveness of digraph decompositions and complexity measures. In: Hong S-H, Nagamochi H, Fukunaga T (eds) ISAAC. Lecture notes in computer science, vol 5369. Springer, Berlin, pp 220–231

Papadimitriou CH, Yannakakis M (1991) Optimization, approximation, and complexity classes. J Comput Syst Sci 43(3):425–440

Poljak S, Tuza Z (1995) Maximum cuts and large bipartite subgraphs. In: Combinatorial optimization. Papers from the DIMACS special year, pp 181–224