2D-TUCKER is PPAD-complete

Dömötör Pálvölgyi

Ecole Polytechnique Fédérale de Lausanne, Switzerland,
`dom@cs.elte.hu`,
`http://www.cs.elte.hu/~dom`

Abstract. Tucker's lemma states that if we triangulate the unit disc centered at the origin and color the vertices with $\{1, -1, 2, -2\}$ in an antipodal way (if $|z| = 1$, then the sum of the colors of z and $-z$ is zero), then there must be an edge for which the sum of the colors of its endpoints is zero. But how hard is it to find such an edge? We show that if the triangulation is exponentially large and the coloring is determined by a deterministic Turing-machine, then this problem is **PPAD**-complete which implies that there is not too much hope for a polynomial algorithm.

1 Introduction

Papadimitriou defined in [3] the complexity class **PPAD** which is a special class of search problems. It contains the problems which are reducible to LEAFD, a total search problem defined as follows.

Definition 1 (LEAFD)**.** *We are given the description of a deterministic Turing machine M which for every input $v \in \{0,1\}^n$ returns an ordered pair from the set $\{0,1\}^n \cup \{no\}$ in time $poly(n)$. These will be denoted by $M_{in}(v)$ and $M_{out}(v)$. This defines a directed graph on $V = \{0,1\}^n$ such that $uv \in E$ if $M_{out}(u) = v$ and $M_{in}(u) = v$. This graph is a collection of directed cycles and paths. We also require that $M_{in}(0^n) = no$, meaning that 0^n is a leaf (or an isolated node). The parity argument implies that in the former case there must be another leaf. The input is 0^n (apart from the description of M), the output of the search problem is another leaf (or 0^n if it is an isolated node).*

Remark 2. *We can suppose that M is equipped with a standard built-in mechanism that checks its running time and if M would run too long, it halts and outputs* no. *It can also guarantee $M_{in}(0^n) = no$. It can be easily checked whether the description of M has this property and if not, then the output of the search problem can also be* violation. *In the problems that we will define later, we similarly allow the output to be* violation *if M violates one of the properties that we require.*

Remark 3. *This problem is* almost *in* **TFNP***, the class of total search problems verifiable in polynomial time. We do not want to define this class here (see [2]).*

The reason why it is not in the class with this definition is that the verification time will depend on the running time of M, so it is not bounded by some fixed polynomial like it is in the case of SAT. *An alternative definition would be to define a class called* LEAFD-C *where the running time of M would be bounded by n^c or to define M as a boolean circuit. It is not the goal of this paper to go deeper in this problem.*

For the definition of reduction among search problems, we direct the reader to the original paper of Papadimitriou [3]. It was also shown there that the search versions of many well-known theorems that use some kind of parity argument belong to **PPAD**, moreover, many are also complete for this class. The following analogue of Sperner's lemma was shown to be **PPAD**-complete by Chen and Deng [1]. (When we write $x \in \{0,1\}^n$, we also mean the number in base two that it represents.)

Definition 4 (2D-SPERNER)**.** *We are given the description of a deterministic Turing machine M which for every input $(u,v) \in \{0,1\}^{2n}$ such that $u+v \leq 2^n$ returns either 1, 2 or 3 in time poly(n). Furthermore, $M(0,0)=1$, $M(2^n,0)=2$, $M(0,2^n)=3$, for all $i<2^n$ $M(0,i) \neq 3$, $M(i,0) \neq 2$ and for all $i+j=2^n$ $M(i,j) \neq 1$. The output (whose existence is guaranteed by Sperner's lemma) is $(u,v) \in \{0,1\}^{2n}$ for which $M(u,v)$, $M(u+1,v)$ and $M(u,v+1)$ are all different.*

One can similarly define 3D-SPERNER and other higher dimensional analogues. It is also possible to define a continuous version, which can be denoted by 2D-BROUWER, the interested reader is again directed to [3] where it is also shown that all these variants are equivalent to LEAFD and thus are **PPAD**-complete.

Definition 5 (2D-TUCKER)**.** *We are given the description of a deterministic Turing machine M which for every input $(u,v) \in \{0,1\}^{2n}$ returns either 1, -1, 2 or -2 in time poly(n). Furthermore, for all i $M(0,i) = -M(2^n, 2^n - i)$ and $M(i,0) = -M(2^n - i, 2^n)$. The output (whose existence is guaranteed by Tucker's lemma) is $(u,v) \in \{0,1\}^{2n}$ and $(u',v') \in \{0,1\}^{2n}$ for which $|u-u'| \leq 1$, $|v-v'| \leq 1$ and $M(u,v) = -M(u',v')$.*

Remark 6. *Tucker's lemma is often stated in a slightly different way, more similar to Sperner's, and it requires the square to be triangulated. The above search problem is clearly easier than the triangulated one, so when we prove a hardness result about* 2D-TUCKER, *that also implies the hardness of the triangulated version, so our results hold for both cases.*

The respective higher dimensional and continuous versions are denoted by 3D-TUCKER and 2D-BORSUK-ULAM, the interested reader is again directed to [3] where it is shown that the higher dimensional version is **PPAD**-complete and 2D-BORSUK-ULAM is equivalent to 2D-TUCKER. The **PPAD**-completeness of 2D-TUCKER was posed as an open problem both in [3] and in [1]. In this note we prove this result.

Theorem 7. 2D-TUCKER *is* **PPAD***-complete.*

2 Reduction of LEAFD to 2D-TUCKER

It was shown in [3] that 2D-TUCKER $\in$ **PPAD**, to prove hardness, we will reduce LEAFD to it. The reduction is surprisingly easy and only uses technics similar to the ones appearing already in [1].

We will call the vertices of the grid *points* and the vertices of the graph generated by M simply *vertices*. We say that two points are *neighbors* if their distance is $\leq \sqrt{2}$ (meaning there is a little square that has both as its vertex). We call two points *negated* if the sum of their colors is zero.

The goal is, that given any M that generates an input for LEAFD, we want to produce a coloring c of the points of the $20 \cdot 2^{2n} \times 20 \cdot 2^{2n}$ grid with colors $\pm\{1,2\}$ such that if one finds two negated neighbors, then we can find a leaf in the graph generated by M.

The idea is that for every vertex we reserve a part of the grid and if there are two negated neighbors in a reserved part, that will imply that the vertex to which this part belongs to is a leaf (there cannot be negated neighbors outside the parts reserved for vertices). Most part of the square is filled with 1's, the edges are represented by tubes of $-2, -1, 2$ going from one reserved part to the other, and these tubes are disjoint (if two tubes would cross, we slightly modify them in the vicinity of the crossing so as they evade each other). Unfortunately it is quite ugly to give a precise description of this construction by words, we advice the reader to consult the Figures which might be sufficient even without reading the text to understand the whole reduction.

For a vertex v_i (where the indices are an arbitrary enumeration of the 2^n vertices with v_0 being 0^n) we reserve a part close to the left side of the square, $V_i = [8,\dots,10] \times [20i2^n + 10, \dots, 20(i+1)2^n - 10]$. For different i's, these parts are disjoint and are above each other. We also reserve a part for every possible edge of the graph. For the possible v_iv_j edge we reserve a part that connects the lower half of V_i and the upper half of V_j via a $\sqsupset$ shape[1], $E_i = [11,\dots,20i2^n + 10+10j] \times [20i2^n+10+10j,\dots,20i2^n+10+10j+2] \cup [20i2^n+10+10j,\dots,20i2^n+10+10j+2] \times [20i2^n+10+10j,\dots,20j2^n+10\cdot 2^n+10+10i+2] \cup [11,\dots,20i2^n+10+10j+2] \times [20j2^n+10\cdot 2^n+10+10i,\dots,20j2^n+10\cdot 2^n+10+10i+2]$. These regions are mainly disjoint, every intersection $E_i \cap E_j$ is a little square, far from the other edges. If v_iv_j is an edge, then we fill out this tube of thickness 3 with $-2, -1, 2$, with the -1's being in the middle, the -2's being in the bottom when leaving v_i and in the top when entering v_j, the 2's being in the top when leaving v_i and in the bottom when entering v_j. (We deal with the intersections of filled out tubes later). Remember that most of the square is filled out with 1's, so if a point does not belong to a part reserved to an edge or vertex, then its color is 1. This way we do not create any negated neighbors outside of the parts reserved for vertices, since the boundaries of the tubes are always ± 2's. Inside V_i, if v_hv_i and v_iv_j are both edges, we fill out the vertical tube of thickness 3 leading from where the tube of the edge from v_h enters down to where the tube of the edge to

[1] we suggest to skip the following ugly description and just read the properties in the next sentence

v_j starts $([8,\dots,10] \times [20i2^n + 10 + 10j, \dots, 20i2^n + 10 \cdot 2^n + 10 + 10h + 2])$ with $-2, -1, 2$ such that the -1's are in the middle, the -2's are to the left and the 2's are to the right. This way again do not create any negated neighbors. If v_i is a leaf, then leave it filled out with 1's (which gives negated neighbors) except for v_0. To v_0 we "drive in" the boundary of the square, we set $c(m,m) = 2$, $c(0,0) = c(0,1) = -2$, for all $1 < i$ $c(0,i) = c(i,m) = -1$, for $0 < i < m$ $c(i, m-1) = 2$, for $2 < i < m$ $c(1,i) = 2$ and continue this tube to inside v_0 and from there to the start of the tube of the edge to its only neighbor.

We have almost solved the problem, the only thing left that we must handle is if two filled out tubes cross. In this case we can simply modify the tubes in the vicinity of their crossing such that we do not create negated neighbors. If for example the edge ab would cross cd, then we modify the tubes such that we obtain an ad and a bc edge (see Figures). Of course these will not really be tubes leading from V_a to V_d and from V_b to V_c because we are handling several crossings, but that does not matter for us. We only want to preserve the conditions that the colors are easy to determine and that there are no negated neighbors outside the parts reserved for vertices.

Now the color of any point can be determined by a finite number of computations of M (we can easily decide from the coordinates of any point whether it belongs to a part reserved for a vertex, an edge, to a crossing or to to the remaining part of the grid). If we find two negated neighbors, they must be in a part reserved for a vertex that is a leaf in the original graph. This finishes the reduction.

3 Remarks and acknowledgment

The same argument works to solve 2D-SPERNER which slightly simplifies the proof of [1].

An interesting question would be to determine the complexity of the so-called octahedral Tucker's lemma (here the dimension would be a part of the input in unary), which might tell something about the complexity of necklace splitting among two thieves with a lot of different kinds of beads. Since this theorem is not so widely known and can be stated in a purely combinatorial way, we state it here.

Lemma 8. *(Octahedral Tucker's lemma) If for any set-pair* $A, B \subset [n], A \cap B = \emptyset, A \cup B \neq \emptyset$ *we have a* $\lambda(A,B) \in \pm[n-1]$ *color, such that* $\lambda(A,B) = -\lambda(B,A)$*, then there are two set-pairs,* (A_1, B_1) *and* (A_2, B_2) *such that* $A_1 \subset A_2$*,* $B_1 \subset B_2$ *and* $\lambda(A_1, B_1) = -\lambda(A_2, B_2)$*.*

I would like to thank Jarek Byrka for discussions and reading the first version of this paper.

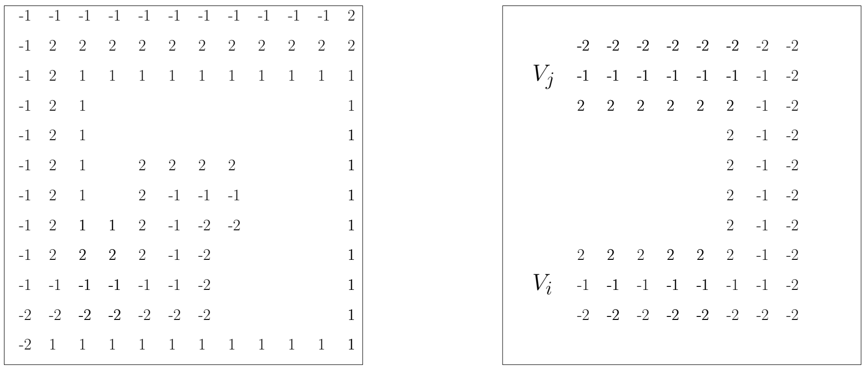

Fig. 1. The boundary "going" into the leaf v_0 and An edge v_iv_j

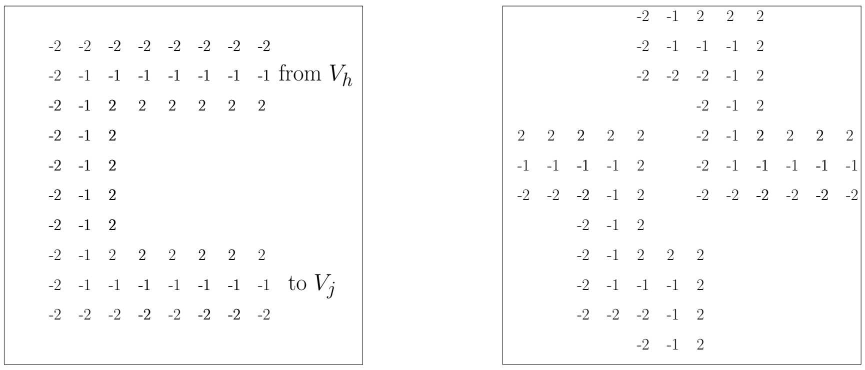

Fig. 2. The part V_i with edge from v_h and to v_j and Handling a crossing

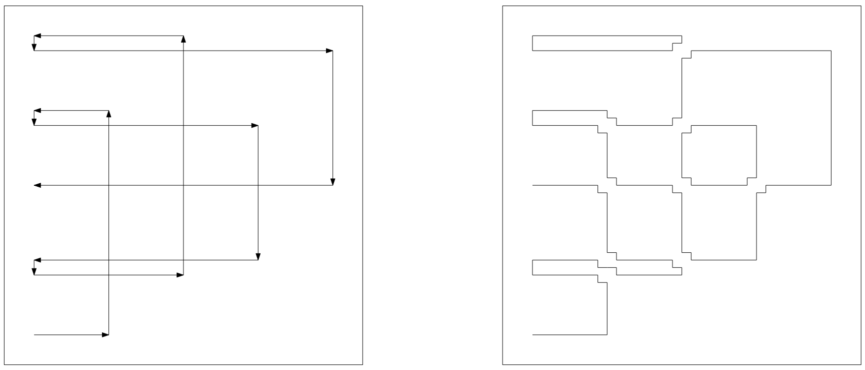

Fig. 3. The graph of the path $v_0v_3v_1v_4v_2$ before and after handling the crossings

References

1. Chen, X., Deng, X.: On the complexity of 2D discrete fixed point problem. In: 33rd International Colloquium on Automata, Languages and Programming, pp. 489–500 (2006).
2. N. Megiddo and C. H. Papadimitriou. On total functions, existence theorems, and computational complexity, Theoretical Computer Science 81(2):317-324, 1991.
3. C. Papadimitriou, On the complexity of the parity argument and other inefficient proofs of existence. J. Comput. System Sci. 48 (1994), pp. 498–532.