TWMS J. App. and Eng. Math. V.16, N.4, 2026, pp. 443-456

# $S_4$ SEPARATION AND P-PARTITION IN ALL-PATH AND DETOUR CONVEXITIES

V. HAPONENKO[1,2*], §

Abstract. In this work, we consider problems of $S_4$ and $p$-convex partition separations with respect to the all-path and the detour convexities. We give characterizations of $p$-all-path convex and $p$-detour convex graphs. With respect to all-path convexity $S_2$, $S_3$, and $S_4$ separable graphs are characterized. Also, we present necessary and sufficient conditions for two sets to be $S_4$ separable, for both convexities. Moreover, we prove that in all-path convexity the time complexity of those problems is linear, and it is NP-hard for detour convexity. Finally, we give an algorithm for determining whether two sets in graph are $S_4$ separable with respect to all-path convexity.

Keywords: all-path convexity, graph convexity, detour convexity, convex separation, $p$-partition.

AMS Subject Classification: 05C85, 52A01, 05C05, 05C38.

## 1. Introduction

The theoretical background for graph convexity was set up in [10]. There, for some set $X$, a convexity $\mathcal{C}$ was defined as a family of subsets of $X$ such that $\mathcal{C}$ includes $X$ and $\emptyset$, and is closed under intersections and nested unions. Then, an ordered pair $(X, \mathcal{C})$ is called a convexity space. From there, graph convexity rises by substituting a set $X$ with a graph $G$. Most studied graph convexities include so-called interval convexities such as geodesic and monophonic convexities [3]. Here, the convexity $\mathcal{C}$ is induced by sets that contain the respective interval between every pair of its vertices. For geodesic convexity, the interval is a collection of all shortest paths between vertices. Meanwhile, monophonic convexity is generated by induced paths.

Our study focuses on all-path and detour convexities. The former is induced by the interval that contains all simple paths between the elements. Detour interval, in contrast to geodesic one, includes all the longest simple paths. A similar interval with the same

[1] Department of Mathematics, National University of Kyiv-Mohyla Academy, Skovorody str. 2, 04070 Kyiv, Ukraine.
[2] Computer Science Department, Kyiv School of Economics, Mykoly Shpaka str. 3, 03113 Kyiv, Ukraine.
e-mail: vladyslav.haponenko@gmail.com; ORCID: https://orcid.org/0009-0001-1064-2512.
* Corresponding author.
§ Manuscript received: March 06, 2025; accepted: July 30, 2025.
TWMS Journal of Applied and Engineering Mathematics, Vol.16, No.4; 

name was introduced in [4], but with additional property of being induced. This way, the longest induced path equals the shortest one for a pair of adjacent vertices which also prevents it from being a metric interval. Thus, in [5] this property was rejected which latter finally established in the study about detour convexity [6].

In the context of mentioned convexities, we study closely related problems of *p-partition* and $S_4$ *separation*. A graph $G$ is said to be $p$-convex with respect to some convexity if it can be partitioned into $p$ convex disjoint sets. Next, two disjoint subsets $A$ and $B$ of $V(G)$ are $S_4$ separable in $G$ if they belong to different halfspaces. That can also be restated in the terms of 2-partition of $G$, so the study of one naturally involves the other. We can see this in the work [8] where the problem of $p$-partition in monophonic convexity was shown to be NP-complete in general case, except 2-partition, which latter was proven to be polynomial in [2]. The problem of half-space separation in geodesic convexity is also NP-complete which was covered in [7]. Also [12] shows that $p$-partition is NP-complete as well for convexity of the induced path of order three or $P_3$.

The paper is structured as follows. In section 2 we give basic definitions for graphs along with the notions of convexity and separation in section 2.2. Next, in section 3.1 theoretic results on all-path halfspace separation and general separability are given. This is followed by section 3.2 where the new algorithm for all-path halfspace separation is introduced. Finally, in section 3.3 we introduce theoretic results for $p$-partition and halfspace separation in the context of detour convexity.

## 2. Definitions and preliminary results

2.1. **Basics Definitions.** In this work, we consider only finite connected undirected graphs. A *graph* $G$ is an ordered pair of sets $(V, E)$ which are the *vertex set* and the *edge set* of $G$ respectively. To distinguish them from some other sets, they are usually written as $V(G)$ and $E(G)$. Further, edges $\{a, b\} \in E(G)$ are denoted as $ab$. Two vertices $a, b \in V(G)$ are *adjacent* given that $ab \in E(G)$. The *path* $P_{ab}$ between vertices $a$ and $b$ of length $n$ is an ordered subset of the vertex set $V(G) \supset P_{ab} = \{v_1, \ldots, v_i, \ldots, v_n\}$, where $i \in N$, $v_1 = a$, $v_n = b$ and $v_i v_{i-1} \in E(G)$ for all $v_i, v_{i-1} \in P_{ab}$ with $2 \leq i \leq n$. When some path $P$ is a *simple path*, $v_i = v_j$ implies that $i = j$ for $v_i, v_j \in P$. A graph $G$ is *connected* when between every two vertices of $G$ exists a path. From this, *the shortest path* between different vertices $a$ and $b$ is a simple path $P_{ab}$ with the smallest number of vertices. It also induces a metric $d(a, b) = |P_{ab}|$ where $P_{ab}$ is the shortest path. On the contrary, a simple path $P_{ab}$ with the largest number of vertices is called *the longest path*. In the literature it is also referred to as *detour*. It also induces a metric which is sometimes denoted by $d^*(a, b)$ or $D(a, b)$. For a vertex $v \in V(G)$ a set of its adjacent vertices $N(v) = \{u : vu \in E(G)\}$ is the *open neighborhood* of $v$. Whereas the *closed neighborhood* $N[v]$ also contains $v$ or $N[v] = N(v) \cup \{v\}$. A vertex $v$ is a *leaf* when it is adjacent to only one vertex or $|N(v)| = 1$. A *closed neighborhood of a set* $S$ is a union of neighborhoods of its vertices $N[S] = \bigcup_{v \in S} N(v)$, and *the open neighborhood of the set* $S$ is its closed neighborhood without itself $N(S) = N[S] \setminus S$. A pair of disjoint sets $A$ and $B$ are *adjacent* if $N(A) \cap B \neq \emptyset$.

A graph $G$ is *complete* if every pair of its vertices is adjacent. A complete graph with $|V(G)| = n$ is denoted by $K_n$. A graph $H$ is a *subgraph* of $G$ when $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For some subset $A \subseteq V(G)$, edges with endpoints in $A$ are denoted by $E_G(A)$. The subgraph $H$ with vertex set $V(H) = A$ and edge set $E(H) = E_G(A)$ is called the *induced subgraph* by $A$. We denote it by $G[A]$. A graph $G$ is called *biconnected* when there is no vertex $v \in V(G)$ such that a graph $G' = (V(G) \setminus \{v\}, E(G))$ is disconnected. A subset $S \subseteq V(G)$ is a *biconnected component* of $G$ if its induced subgraph $G[S]$ is

biconnected. A *block* is a maximal by inclusion biconnected component of $G$. A graph $G$ is called a *block graph* when each of its blocks induces a complete subgraph. Next, we define a *bridgeless component* as a maximal by inclusion connected union of blocks without bridges (which are $K_2$). For example, in a tree graph every vertex is a bridgeless component since it has only bridge blocks.

2.2. **Convexity and Separation.** We use the definition of *convexity* introduced in [10]. There, for some set $X$ its family of subsets $\mathcal{C}$ is called a convexity if it satisfies three conditions:

(1) $\emptyset, X \in \mathcal{C}$;
(2) $\mathcal{C}$ is closed under intersections;
(3) $\mathcal{C}$ is closed under nested unions.

Then, a pair $(X, \mathcal{C})$ is called a *convexity space*, and elements of $\mathcal{C}$ are referred to as convex sets. In a convexity space $(X, \mathcal{C})$ a *convex hull* of a subset $A \subseteq X$ is the smallest convex set containing $A$. As mentioned before, the most studied convexities are *interval convexities*. An *interval operator* on $X$ is a map $I : X \times X \to 2^X$ that is symmetric and includes its endpoints:

(1) $a, b \in I(a, b)$ for all $a, b \in X$;
(2) $I(a, b) = I(b, a)$ for all $a, b \in X$.

For an interval operator $I$ we can define a family $\mathcal{C}$ of subsets of $X$ that every $S \in C$ also contains an interval between every pair of its vertices or $\forall a, b \in S, I(a, b) \in S$. Such family $\mathcal{C}$ satisfies conditions of convexity and forms a convexity space $(X, \mathcal{C})$. Here it is said that the convexity $\mathcal{C}$ is induced by $I$. In the research, we consider all-path and deour convexities.

*All-path convexity* is induced by the all-path interval operator $I$, where $I(a, b)$ is a family of all simple paths between $a, b \in V(G)$. In this paper, we will rely on the next characterization of all-path convex sets.

**Theorem 2.1.** [9, Theorem 3.1.1] *Let $G$ be a connected graph, $A \subseteq V(G)$ and $|A| \geq 2$. The set $A$ is AP-convex if and only if the induced subgraph $G[A]$ is a connected union of blocks.*

For the *detour convexity* $I(a, b)$ is a family of all longest simple paths between the vertices. In our results, an important role play *detour extreme* vertices. A vertex $v \in V(G)$ is detour extreme if it is an initial or terminal vertex of any detour in $G$ containing the vertex $v$. Restating the definition, $v$ can not be between any vertices on their detour interval. The further statement could be treated as the characterization of a detour extreme vertex.

**Proposition 2.1.** [5, Observation 2.3] *A vertex $v$ is detour extreme vertex if and only if $V(G) \setminus \{v\}$ is a detour convex set in $G$*

From the Theorem 2.1 it is clear that any all-path convex set is also detour convex. In the context of this relation we propose this modification of Proposition 2.1:

**Proposition 2.2.** *If $u \in V(G)$ is a detour extreme vertex of induced subgraph $G[S]$ where $S$ is a union of blocks of some graph $G$, then $u$ is a detour extreme vertex of $G$.*

*Proof.* Proving by contradiction, let us assume that $u$ is not a detour extreme in $G$. Then, there exists a pair $a, b \in V(G)$ and a detour path $P_{a,b}$ which passes through $u$, i.e. $P_{a,b} = \{a, \ldots, v_i, u, v_{i+2}, \ldots, b\}$. Since $S \cap P_{a,b} \neq \emptyset$, we are able to take the biggest by inclusion subpath $P'_{a,b}$ of $P_{a,b}$ that lays in $S$. Let $P'_{a,b} = \{v_i, \ldots, u, \ldots, v_j\}$ where $i \leq j$ and $P'_{a,b} \subseteq S$. But then, $P'_{a,b}$ has to be a detour of $v_i v_j$ in $G[S]$ because $P_{a,b}$ is not an $ab$ detour otherwise. Thus, a contradiction with the fact that $u$ is detour extreme in $G[S]$. □

For a convexity space $(X, \mathcal{C})$, a convex set $H \subseteq X$ is a *halfspace* if $X \setminus H$ is convex. A pair of disjoint subsets $A, B$ are said to be separable by halfspaces if there exists a halfspace $H$ that $A \subseteq H$ and $H \cap B = \emptyset$. For some convexity space $(X, \mathcal{C})$, the following separation axioms are considered:

(1) $S_2$ separation: every pair of distinct vertices of $X$ are separable by halfspaces;
(2) $S_3$ separation: any convex set $A$ and a point $b \notin A$ are separable by halfspaces;
(3) $S_4$ separation: every pair of disjoint convex sets are separable by halfspaces.

This induces a decision problem of determining whether some pair of disjoint sets $A, B$ are separable for some graph $G$. The problem of *p-convex partition* generalises finding halfspaces. Here, a partition on $p$ convex sets of $V(G)$ is to be found. This problem could be seen as finding a halfspace $H$ and considering the problem for the induced subgraphs $G[H]$ and $G[V \setminus H]$. Additionally, we consider modified separation axioms where not only convex sets are separated by halfspaces. Further, they are referred to as strong separation axioms and could be written as follows:

(1) $S_3$ strong separation: any set $A$ and a point $b \notin A$ are separable by halfspaces;
(2) $S_4$ strong separation: every pair of disjoint sets are separable by halfspaces.

## 3. Main results

3.1. **All-path convexity.** Our first theorem gives a necessary and sufficient condition of existence of all-path convex halfspace. This fact also will be a cornerstone of our algorithm for deciding on $S_4$ separability of two disjoint sets. Disjoint adjacent blocks is a core concept for all-path convex halfspaces. As an example of such adjacency, blocks $B_1$ and $B_2$ in Figure 1 could be referred to. Note that this type of adjacency between $B_1$ and $B_2$ could also be reached via any complete graph in between the blocks, but in this case, all-path convex halfspaces division would not be possible which we prove further.

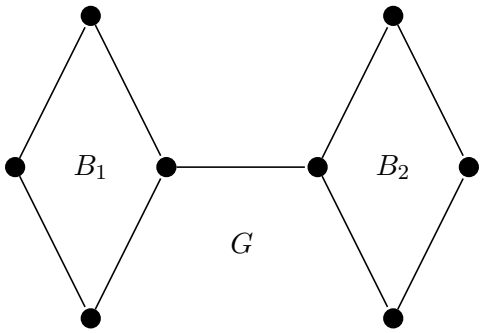


Figure 1. Two disjoint adjacent blocks of $G$.

**Theorem 3.1.** *A graph $G$ has all-path convex halfspaces if and only if $G$ has two adjacent and disjoint blocks.*

*Proof.* Let us consider two complementary halfspaces $A$ and $B$. Since they are all-path convex, $A$ and $B$ are blocks or connected union of blocks. From the connectedness of $G$, there is a vertex $a \in A$ such that $N(a) \cap B \neq \emptyset$, thus there exists $b \in B$ such that $b \in N(a)$. Moreover, $a$ and $b$ are the only vertices with such properties, because otherwise $A$ or $B$ is not a block or connected union of blocks. Therefore, respective blocks of $a$ and $b$ are adjacent but disjoint.

Let $A_1$ and $A_2$ be two disjoint adjacent blocks. Then, $A_1$ and $A_2$ are connected by a bridge block and every other block $B$ in $G$ is closer only to either $A_1$ or $A_2$. Such division

induces two subsets like $A_i \cup B_i$, where $B_i$ is a union of blocks that are closer to $A_i$. Since the union $(A_1 \cup B_1) \cup ((A_2 \cup B_2)) = V(G)$, each of the two unions is all-path convex as the connected union of blocks (see Theorem 2.1), and moreover, they are complementary which makes them the halfspaces. □

**Corollary 3.1.** *All-path halfspaces are connected by a single edge which is a bridge block.*

Note how for a block $B$ in a graph $G$ the condition of being connected by bridges to blocks in its complement $V(G)\setminus B$ ensures all-path convexity in $G$ of connected components of induced subgraph $G[V(G) \setminus B]$. On the contrary, consider a bridgeless non-biconnected graph $G$. There any pair of adjacent blocks shares a cut vertex; thus, the complement of each one of them is not all path convex in $G$. In this way, bridgeless components of $G$ become minimal by inclusion subsets of $G$ which complement has all-path convex connected components.

In the following results, we characterize graphs which are $S_2$, $S_3$, and $S_4$ separable with respect to the all-path convexity.

**Proposition 3.1.** *A graph $G$ is $S_2$ (equivalently, $S_3$, or $S_4$) all-path separable if and only if $G$ is a tree.*

*Proof.* **Necessity.** By contradiction, assume a connected graph $G$ is not a tree. Hence, there exists a cycle $C \subseteq G$. Consider a pair of adjacent vertices $a, b \in V(C)$. From the fact that $a$ and $b$ are contained in a biconnected component of $G$, any halfspace $H$ that $a \in H$ and $b \notin H$ is connected to the other halfspace by more than one edge, which is a contradiction by Corollary 3.1. Thus, $S_2$ separability requires $G$ to be a tree, but since all singletons are all-path convex, this is also true for $S_3$ and $S_4$ separability.

**Sufficiency.** For a tree $G$ consider a pair of two disjoint connected sets $A, B \subseteq V(G)$ with $|A| \geq 1$ and $|B| \geq 1$. They are convex by the fact that $A$ and $B$ are subtrees of $G$. More over, there exist $a \in A$ and $b \in B$ that $|P_{a,b}| = min\{|P(x, y)| : x \in A, y \in B\}$. Note that $P_{a,b} \setminus \{a, b\}$ do not intersect either $A$ or $B$. Furthermore, by deleting any edge between adjacent vertices of $P_{ab}$ we get subtrees $T_1$ and $T_2$ of $G$ that include $A$ and $B$ respectively and whose vertex sets are the all-path halfspaces. □

**Proposition 3.2.** *A graph $G$ is $S_3$ all-path strongly separable if and only if $G$ is $K_2$.*

*Proof.* **Necessity.** On the contrary, suppose connected graph $G$ is not a $K_2$ graph. Then, $|V(G)| \geq 3$ and there exist three different vertices $a, b, c \in V(G)$ where $b \in P_{ac}$ for some simple path $P_{ac}$. Thus, the set $A = \{a, c\}$ and the vertex $b$ are not separated by all-path convex halfspaces, since any halfspace $H \supset A$ does not contain $P_{ac}$ because $b \in V(G) \setminus H$. A contradiction. **Sufficiency** is obvious. □

**Corollary 3.2.** *A graph $G$ is $S_4$ all-path strongly separable if and only if $G$ is $K_2$.*

The next result shows that any graph $G$, given it has $k$ bridges, is $p$-all-path convex for all $p \leq k+1$. For example, a path-graph with 3 vertices is 2-all-path convex and 3-all-path convex at the same time.

**Lemma 3.1.** *A graph $G$ can be partitioned into at most $k+1$ all-path convex sets, where $k$ is the number of bridges in $G$.*

*Proof.* Since elements of the partition preserve all-path convexity of connected components of their complement, by Corollary 3.1, they are precisely the bridgeless components of $G$. They could be determined by bridge deletion. By deleting a bridge edge, we increase the number of connected components of the graph by 1. By deleting all bridges we are left with $k + 1$ connected components where $k$ is the number of bridges. This proves the Lemma. □

3.2. **All-path algorithms.** Algorithms in this section are based on the classic graph search algorithm DFS or depth-first search. DFS gives a way to traverse the graph in linear time. Despite our implementation being recursive, the time complexity of the algorithms is easy to count thanks to the finite nature of our graphs of interest. Basically, for any $G$ the DFS starts from an arbitrary vertex of the graph and traverses along the edges to further vertices memorizing visited elements of $V(G)$.

For a connected finite simple graph $G$ and its arbitrary vertex $v \in V(G)$ recursive implementation of DFS could be represented by following pseudocode of Algorithm 1.

**Algorithm 1** DFS

```
1: function DFS(v)
2:     visited[v] ← True                          ▷ Happen |V| times
3:     for u ∈ N(v) do                            ▷ Happen 2|E| times
4:         if u is not visited then               ▷ Happen 2|E| times
5:             DFS(u)
6:         end if
7:     end for
8: end function
```

Instruction in line 2 marks a vertex as visited; thus, triggering once for each vertex. Code in lines 3 and 4 is responsible for a traversing cycle where each edge connected to the priorly visited vertex is taken and checked on being traversed. Given that each instruction has a constant time complexity, the overall time complexity is linear relatively $|E|$ and $|V|$ or $T(G) = |V| + 4|E|$.

Another essential part of our results is the linear time algorithm for finding all blocks and cut vertices of the graph by Tarjan from [11]. The algorithm is linear relatively $|V| + |E|$ which will be referred to as $T_{Tarjan}(G)$.

**Lemma 3.2.** *Deciding whether $G$ is $p$-all-path convex can be done in linear time.*

*Proof.* The mentioned Tarjan algorithm finds all cut vertices and blocks of a graph $G$ in linear time. Let $B_G$ be a set of all blocks of $G$. From Lemma 3.1, the property of being $p$-all-path convex depends only on a number of bridges in $G$. Thus, it is sufficient to check whether the number of elements of $B_G$ with cardinality 2 is bigger than $p-1$ or $|\{S : S \in B_G, |S| = 2\}| \geq p-1$. This also could be done in linear time which completes the proof. □

To prove the next lemma we introduce a two-part implementation of the algorithm in pseudocode (Algorithm 2 and Algorithm 3).

**Lemma 3.3.** *Deciding whether two disjoint sets are separable with respect to all-path convexity can be done in linear time.*

**The Idea** of the first part of the algorithm is to traverse the given graph $G$ in order to find its bridgeless components. During this process, we also check whether each such component intersects sets $A$ and $B$ (the sets we check on being $S_4$ separable). Note that the algorithm is applicable to both separability and strong separability. Thus, we build a tree graph $T = (V, E, S)$ where elements of $V(T)$ represent bridgeless components of $G$ and there is an edge between them in $T$ in case they are connected in $G$. The set $S(T)$ is a family of vectors $(a, b)_v : a, b \in \{0, 1\}$ for each $v \in V(T)$. There, elements $a$ and $b$ equal 1 when the respective bridgeless component intersects $A$ or $B$ respectively.

In the second part of the algorithm, we work with the tree of bridgeless components $T$. Here the idea is based on the fact that each pair of vertices in the tree is connected by a single shortest path. Hence, after validating that each bridgeless component (element of $T$) is associated either with $A$ or $B$, or neither of them, we find the union of shortest paths that connect pairs of components associated with $A$ and $B$. Then, the sets are possible to $S_4$ separate when the mentioned unions are disjoint.

**Preparation** of the procedure includes verifying $A$ and $B$ on being disjoint and running the Tarjan algorithm. The latter is needed for obtaining a family of bridge blocks of $G$. Also, the algorithm is supposed to process the simple undirected finite graphs.

**The first part** (Algorithm 2) of the algorithm consists of two functions: *triggerDFS* and *DFS*. Arguments of *triggerDFS* are:

- *startVertex* — is an arbitrary vertex of $G$;
- *bridges* — is a collection of bridge blocks of $G$;
- $A$ — a set of vertices $A \subseteq V(G)$;
- $B$ — a set of vertices $B \subseteq V(G)$.

The function returns a tree $T$ of bridgeless components of $G$. The lines from 2 to 4 are responsible for the initialization of the collection of *visited* vertices and the tree $T$. The collection *queue* is intended to store future vertices of $T$. This collection is a queue data structure where the earliest added element is the first one to be removed via instruction *queue.pop()*. This way we ensure the right connectivity between elements of $T$. The *while* cycle controls traversal through $G$. Here we obtain a vertex $v$ of $T$, its vector $s \in S(T)$, and trigger the (DFS) function.

The *DFS* function traverses the graph. Arguments of *DFS* are:

- $v$ — a traversed vertex;
- $a$ — a vertex of $T$ which connected component in $G$ is being traversed;
- $A$ — a set of vertices $A \subseteq V(G)$;
- $B$ — a set of vertices $B \subseteq V(G)$.

Here we check whether the component intersects $A$ or $B$ at lines $15 - 20$ and mark the vertex as visited at line 21. If the traversed edge is a bridge we prevent further traversal and add the other vertex to $V(T)$ at lines $24 - 26$, or continue the exploration of $G$ otherwise at line 28. This logic ensures the complete exploration of a bridgeless component before moving next to its neighbors. Figure 2 gives an example of how Algorithm 2 transforms the graph $G$ with a starting vertex $a$.

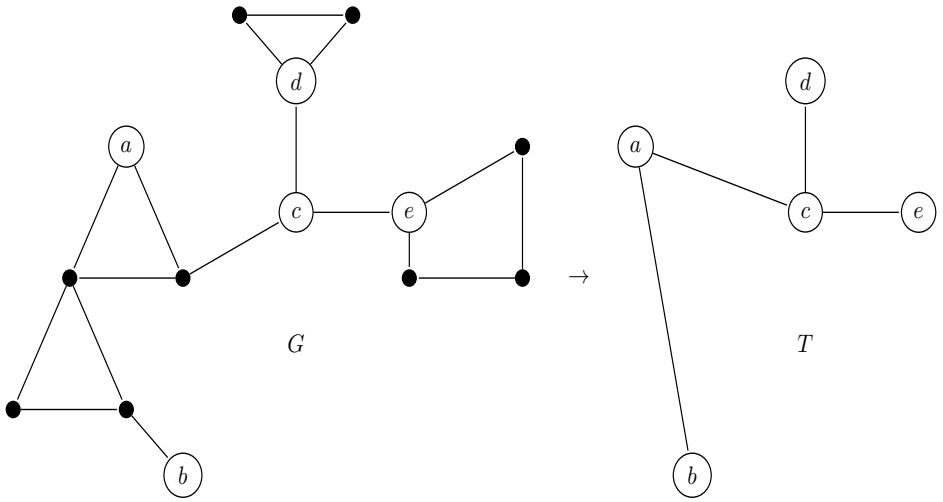


FIGURE 2. Result of Algorith 2 for a graph $G$ and starting vertex $a$

The **time complexity** is computed as follows. Consider *triggerDFS* function. The first three lines of it contain three constant time initiations of variables. The *while* cycle has four $O(1)$ variable initiations with a function call. It triggers once for each bridge edge and an extra time for the *startVertex*. Here the worst case is when $G$ is a tree, thus the number of bridges equals $|V| - 1$. Hence, $T(triggerDFS) = 3 + |bridges| + 1 + 5(|bridges| + 1) \leq 3 + |V| + 5|V| = 3 + 6|V|$.

Next, consider *DFS* function. It triggers once for each vertex. Thus, in the worst-case scenario each of them goes through five constant time instructions at lines $15 - 21$. This already contributes $5|V|$ to the time complexity. Instructions in lines $22 - 23$ trigger twice for each edge with the *if* statement further triggering once for each edge with, at maximum, two $O(1)$ instructions after. This results in the time complexity $T(DFS) = 5|V| + 2|E| + 2|E| + 2|E|$. Therefore, the **time complexity of the first part** of the algorithm is $T(Algirithm\ 2) = 3 + 11|V| + 6|E|$.

**The second part** (Algorithm 3) of the algorithm also has two functions: *triggerDFS* and *DFS*. There, arguments of *triggerDFS* are:

- *startA* — is an arbitrary vertex of $T$ which represents a bridgeless component that intersects $A$;
- *startB* — is an arbitrary vertex of $T$ which represents a bridgeless component that intersects $B$;
- $T$ — the tree of bridgeless components created by the first part of the algorithm.

It returns the boolean variable which is the decision about all-path $S_4$ separability of sets $A$ and $B$. Here, the first *if* statement in line 2 validates that the sum of bridgeless components, that intersect $A$, or $B$, or neither of them, equals the cardinality of $V(T)$. This ensures the absence of bridgeless components that intersect both $A$ and $B$. At lines $5 - 6$, we initiate variables that count how many bridgeless components are associated with the respective set we managed to reach during the traversal. As mentioned in the **idea** of the algorithm, here we basically find the subgraphs of $T$ of components that are associated with some of the two sets. By finding the subgraph of components that are associated with the set $A$ at line 8, we mark all of its vertices as traversed (visited) which prevents them from being traversed again during the equivalent actions toward the set $B$ at line 9. If some of the components that intersect $B$ are not reached, then the sets are

**Algorithm 2** Algorithm for building the $T_G(A, B)$

```
1: function triggerDFS(startVertex, bridges, A, B)
2:     visited ← ∅
3:     queue ← [startVertex]
4:     T ← (∅, ∅, ∅ )                          ▷ Initiate the empty tree
5:     while queue is not empty do              ▷ Happens |bridges| + 1 times
6:         foundA ← 0
7:         foundB ← 0
8:         a ← queue.pop()                      ▷ Take the earliest element from queue
9:         DFS(a, a, A, B)
10:        S(T)[a] ← (foundA, foundB)           ▷ Assosiate the vector with the vertex
11:    end while
12: return T
13: end function
14: function DFS(v, a, A,B)                     ▷ Triggers once for each v ∈ V
15:    if v ∈ A then
16:        foundA ← 1
17:    end if
18:    if v ∈ B then
19:        foundB ← 1
20:    end if
21:    visited[v] ← True
22:    for b ∈ N(v) do                          ▷ Happens 2|E| times
23:        if b ∉ visited then                  ▷ Happens 2|E| times
24:            if vb ∈ bridges then             ▷ Happens |E| times
25:                E(T) ← ab                    ▷ Adds vertices to E(T) and V(T)
26:                queue.add(b)                 ▷ Adds the vertex to the queue
27:            else
28:                DFS(b, a, A, B)
29:            end if
30:        end if
31:    end for
32: end function
```

not separable. This results in returning the *False* statement in line 10. Going forward, some of the components associated with $A$ could also be unreachable when a component associated with $B$ is between a pair of components that intersect $A$. This part is done during the traversal of $T$ in *DFS* function.

Here *DFS* function traverses $T$. Arguments of *DFS* are:

- $v$ — a traversed vertex;
- *beforeV* — previously traversed vertex;
- *enterVector* — element of $S(T)$ of components that are associated with the set of interest ($A$ or $B$);
- *notEnterVector* — element of $S(T)$ of components that are **not** associated with the set of interest ($B$ or $A$);
- *counter* — the counter associated with the set of interest (initiated at lines 5 or 6 of *triggerDFS*).

Such duality of the last three arguments is the result of *DFS* being used for finding the vertices for both $A$ and $B$. At line 13 the *if* statement validates whether a component $v$ that intersects the set of interest is being traversed. If so, it is marked as visited and the counter is increased. Then, at line 17 among its neighbors $N(v)$ we traverse towards ones that are not visited or not associated with the set different from the one of interest. Note that unassociated vertices are marked as visited at line 21 only when an associated one is encountered further at line 19. We need this logic to mark as visited only such vertices that lay between pairs of components that are associated with the set of interest.

The **time complexity** is computed as follows. In the worst-case scenario, the tree $T$ has the vertex set $|V(T)| = |V(G)|$ when $G$ is a tree. Thus $|E(T)| = |V(G)| - 1$. This not only aligns together the time complexity formulas of two parts of the algorithm but also is mandatory for measuring the worst possible time complexity. First, consider *triggerDFS* function of Algorithm 3. The *if* statement at the start counts the number of bridgeless components of $G$ that intersect $A$ or $B$. We do not provide the explicit algorithm for this, but obviously, it could be done by traversing the graph once with a linear time complexity or $O(|V| + |E|)$. Further, we refer to this fact as to $T_{if}(G)$. Next, it is followed by three constant time variable initiations and two function calls. At line 10 we check whether two equalities hold and draw a boolean value from this. It results in the time complexity of $T(\mathit{triggerDFS}) = T_{if}(G) + 8$.

Consider the function *DFS* of Algorithm 3. There, the first *if* statement consists of two constant time instructions with two $O(1)$ procedures after. The *if* statement at line 18 has six constant time actions with further function call and another *if* statement. Note that in this particular algorithm some vertices may not be marked as visited once traversed. For example, when $T$ is a chain with leaves associated with $A$ and $B$, $|V| - 2$ vertices are traversed twice. Thus, by assuming that *DFS* works twice for each vertex, the time complexity of the algorithm is $T(\mathit{DFS}) = 2(4|V| + 2|V| - 2 + 12|V| - 12 + 3|V| - 3) = 42|V| - 34$.

The **overall time complexity** of deciding whether two disjoint sets $A$ and $B$ of graph $G$ are separable by all-path convex halfspaces is $T(G) = T_{Tarjan}(G) + 3 + 11|V| + 6|E| + T_{if}(G) + 8 + 42|V| - 34 = T_{Tarjan}(G) + T_{if}(G) + 53|V| + 6|E| - 23$ which is linear relatively $|V| + |E|$.

**Algorithm 3** Algorithm deciding whether $A$ and $B$ are separable by all-path convex halfspaces

```
1: function triggerDFS(startA, startB, T)
2:     if |v ∈ T : 1 = x_1 ∈ S(T)[v]| + |v ∈ T : 1 = x_2 ∈ S(T)[v]| + |v ∈ T : S(T)[v] = (0 0)| ≠ |V(T)| then
3:         return False
4:     end if
5:     aCounter ← 0
6:     bCounter ← 0
7:     visited ← ∅
8:     DFS(startA, startA, (1 0), (0 1), aCounter)
9:     DFS(startB, startB, (0 1), (1 0), bCounter)
10:    return |v ∈ T : 1 = x_1 ∈ S(T)[v]| = aCounter and |v ∈ T : 1 = x_2 ∈ S(T)[v]| = bCounter
11: end function
12: function DFS(v, beforeV, enterVector, notEnterVector,couter)
13:    if S(T)[v] × enterVector = 1 then
14:        counter = counter +1
15:        visited[v] ← True
16:    end if
17:    for b ∈ N(v) do                                   ▷ Happens 2|V| − 2 times
18:        if b ≠ beforeV and S(T)[b]× notEnterVector= 0 and b ∉ visited then
19:            DFS(b, v, enterVector, notEnterVector, counter)
20:            if b ∈ visited then
21:                visited[v] ← True
22:            end if
23:        end if
24:    end for
25: end function
```

3.3. **Detour convexity.** Given the fact that every all-path convex set is detour convex, all-path convex halfspaces are also detour convex. Nevertheless, the converse statement is true only in graphs each non-bridge block of which does not have a detour extreme vertex in self-induced subgraph.

**Theorem 3.2.** *A biconnected graph $G$ does not have non-singleton detour halfspaces.*

*Proof.* Let us prove it by contradiction. Suppose $G$ can be divided into two complementary detour convex sets $A$ and $B$ with $|A|, |B| > 1$. Thus, $A$ and $B$ are connected sets, and since their union is biconnected, there exists vertices $a, a' \in A$ that are adjacent to some vertices $b, b' \in B$ respectively, or $ab, a'b' \in E(G)$. From the fact that $A$ is detour convex it follows that $aa'$ detour $P_{aa'}$ lays in $A$, let $|P_{aa'}| = n$. By the same logic, $bb'$ detour $P_{bb'}$ lays in $B$ and, since path $b, P_{aa'}, b'$ has length $n + 2$, we have $|P_{bb'}| > n + 2$. But then, the path $a, P_{b,b'}, a'$ is longer than $P_{aa'}$ which contradicts the assumption that $P_{aa'}$ is $aa'$ detour. This asserts that the existence of such detour convex $A$ and $B$, without loss of generality, is possible only if $B \cap (\bigcup_{a \in A} N(a)) = \{x\}$ for some $x \in B$. That makes $G$ a union of blocks which contradicts the fact that $G$ is biconnected. □

Unlike all-path convexity, in detour convexity bridgeless components, which are also detour convex, can be partitioned into smaller detour convex sets by the next theorem.

**Theorem 3.3.** *A biconnected graph $G$ has detour convex halfspace if and only if it has a detour extreme vertex.*

*Proof.* By contradiction, suppose that the cardinality of either of halfspaces is greater than 1. Then, the proof follows from the last theorem. Otherwise, let some of the set be a non-detour extreme vertex $x$. Thus, vertices from $V(G) \setminus \{x\}$ have some detours that have $x$ so $V(G) \setminus \{x\}$ is not detour convex, which is a contradiction.

Let $x$ be a detour extreme vertex in $G$. Then the set $S = \{x\}$ is detour convex. By the characteristics of extreme vertex, $x$ is only an end or a start point of some detour in $G$. Therefore, $V(G) \setminus \{x\}$ is also a detour convex set. □

**Theorem 3.4.** *In a graph $G$, two disjoint sets $A$ and $B$ can be separated by complementary detour convex halfspaces $H_1 \supset A$ and $H_2 \supset B$ if and only if $H_1 \cap M = \{x\}$, where $M$ is the block shared by the halfspaces and $x$ is detour extreme vertex of $G[M]$.*

*Proof.* **Necessity** follows from the previous theorem and the fact that a block is detour convex.

For **sufficiency**, let disjoint sets $A$ and $B$ are separated by complementary detour convex halfspaces $H_1 \supset A$ and $H_2 \supset B$. Proving by contradiction, suppose $H_1$ and $H_2$ intersect the same block $M$ which is not the bridge. Let $|H_1 \cap M|, |H_2 \cap M| > 1$, then $H_1$ and $H_2$ are not detour convex by Theorem 3.2. Then let $|H_1 \cap M| = x$, where $x$ is not detour extreme in $M$, then there exists a pair of vertices $a, b \in M \cap H_2$ whose detour $P_{ab}$ contains $x$, so $H_2$ is not detour convex. □

The following two results measure the time complexity of finding the halfspaces that separate some subsets of $G$.

**Lemma 3.4.** *Finding detour convex halfspaces that separate two disjoint subsets of $G$ is NP-hard.*

*Proof.* Halfspace separation involves finding detour extreme vertices in the graph induced by the block that is shared by halfspaces. Such vertices are precisely ones that do not lay on the detour of any pair of its vertices. Thus, this problem involves finding the longest paths which is an NP-hard problem. □

**Corollary 3.3.** *Finding detour convex p-partition of the $G$ is NP-hard since 2-partition can be represented as halfspace separation for some subsets of $G$.*

The next lemma ensures that a biconnected graph $G$ with $|V| > 2$ can be partitioned into $e + 1$ detour convex sets where $e$ is the number of detour extreme vertices of $G$.

**Lemma 3.5.** *A biconnected graph $G$ consists only of detour extreme vertices if and only if it is a $K_2$*

*Proof.* **Sufficiency**. In $K_2$ each of the two vertices can be only a start point or an endpoint of some detour which coincides with the characterization of detour extreme points.

**Necessity**. Let us consider a vertex $x \in V(G)$ and its neighborhood $N(x)$. In this neighborhood exists $a \in N(x)$ which lies on some detour that starts or ends on $x$, but since $a$ is an extreme itself the detour ends or starts on $a$ respectively. Therefore, $D_G(a, x) = 1$ which means that there is no cycle in $G$ containing the edge $ax$, and since $G$ is a block, this is possible only if $G = K_2$. □

Let $B_G$ be a family of all blocks whose induced subgraph is not $K_2$ in $G$ and $K_G$ is a family of blocks whose induced subgraph is $K_2$ or a bridge in $G$. We define $Ex_G : 2^{V(G)} \rightarrow$

$\mathbb{R}$, which is a number of detour extreme vertices in a subgraph induced by some subset of $G$.

Further result shows that any graph $G$ is $p$-all-path convex for all $p \leq \sum\limits_{B \in B_G} Ex_G(B) + |K_G| + 1$. As mentioned in the next proof, the process of edge deletion is shown in Figure 3 where dashed edges of $G$ are to be deleted since $a$ is detour extreme in $G$.

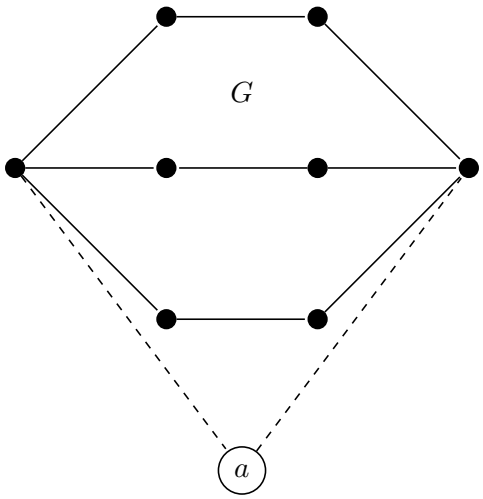


FIGURE 3. A biconnected graph $G$ with detour exteme vertex $a$

**Lemma 3.6.** *A connected graph $G$ can be partitioned into at most $\sum\limits_{B \in B_G} Ex_G(B) + |K_G| + 1$ detour convex sets.*

*Proof.* Since every all-path convex set is also a detour convex set, the least number of detour components in p-partition equals the number of all-path components that are created by removing bridges. By Lemma 3.1, the number of such components is $|K_G| + 1$. Further, every such bridgeless connected component $C$ can be partitioned into more detour convex due to the extreme vertices. Using previously proven Theorem 3.4, we can divide $C$ into complementary detour convex sets by finding a block $B \subseteq C$ and a vertex $x$ that is extreme in $G[B]$ and removing edges that connect $x$ to $N_{G[B]}(x)$. Moreover, the previous theorem ensures that such removal for each extreme vertex always increases the number of connected components by 1. An overall number of such divisions for the component equals $\sum\limits_{B \in B_C} Ex_C(B)$, which means that $C$ can be partitioned into $\sum\limits_{B \in B_C} (Ex_C(B) + 1)$ detour convex sets. Therefore, since the number of such components is $|K_G| + 1$, the maximum possible number of detour convex set in the partition is $\sum\limits_{B \in B_G} Ex_G(B) + |K_G| + 1$. □

Note that trees are simultaneously $p$-all-path convex and $p$-detour convex since they do not have non-bridge blocks.

## 4. Conclusions

In the paper, we showed that the p-partition problem is NP-hard for detour convexity and linear for all-path convexity. For the former convexity, it was proved by algorithms in section 3.2. We also considered the closely related problem of separation for both

convexities. This way we not only introduced new properties of the convexities, but also established a relation between detour convexity and all-path convexity.

## 5. Acknowledgments

I want to thank the Ukrainian military for keeping Mena safe. I also express my gratitude to my scientific supervisor, Sergiy Kozerenko, without whom this paper would not be possible.

Finally, I would like to thank the anonymous referees for their valuable comments and suggestions, which helped improve the quality of this work.

## References

[1] Nielsen, M. H. and Oellermann, O. R., (2012), Separation properties of 3-Steiner and 3-monophonic convexity in graphs, Discrete Mathematics, 312(22), pp. 3293-3305.
[2] Elaroussi, M., Nourine, L. and Vilmin, S., (2024), Half-space separation in monophonic convexity, arXiv preprint arXiv:2404.17564.
[3] Farber, M. and Jamison, R. E., (1986), Convexity in graphs and hypergraphs, SIAM Journal on Algebraic Discrete Methods, 7(3), pp. 433-444.
[4] Chartrand, G., Johns, G. L. and Tian, S., (1993), Detour distance in graphs, In Annals of discrete mathematics, 55, pp. 127-136.
[5] Santhakumaran, A. P. and Chandran, S. U., (2018), The detour hull number of a graph, Algebra and discrete mathematics, 14(2), pp. 307-322.
[6] Arco, R. and Canoy Jr, S., (2017), Detour convexity in graphs, Journal of Analysis and Applications, 15(2), pp. 117-131.
[7] Seiffarth, F., Horváth, T. and Wrobel, S., (2023), Maximal closed set and half-space separations in finite closure systems, Theoretical Computer Science, 973, p. 114105.
[8] González, L. M., Grippo, L. N., Safe, M. D. and dos Santos, V. F., (2020), Covering graphs with convex sets and partitioning graphs into convex sets, Information Processing Letters, 158, p. 105944.
[9] Haponenko, V. and Kozerenko, S., (2024), All-Path Convexity: Two Characterizations, General Position Number, and One Algorithm, Discrete Math. Lett., 13, pp. 58–65.
[10] van De Vel, M. L., (1993), Theory of convex structures, Elsevier, 50.
[11] Tarjan, R., (1972), Depth-first search and linear graph algorithms, SIAM journal on computing, 1(2), pp. 146-160.
[12] Centeno, C. C., Dantas, S., Dourado, M. C., Rautenbach, D. and Szwarcfiter, J. L., (2010), Convex partitions of graphs induced by paths of order three, Discrete Mathematics & Theoretical Computer Science, 12(5), pp. 175-184.

**Vladyslav Haponenko** was born in 2000 in Mena, Ukraine. He earned both his Bachelor's (2021) and Master's (2023) degrees in Applied Mathematics from the National University of Kyiv-Mohyla Academy. In 2023, he began his PhD studies in Applied Mathematics at the same institution, focusing on advanced topics in mathematical modeling and computation. In 2025, he joined the Graph Theory and Network Analysis Laboratory at the Kyiv School of Economics as a researcher. His academic path reflects a strong dedication to mathematical sciences, with a growing interest in the application of graph theory and network analysis to real-world problems.