Active Learning on Convex Subgraphs with Shortest Path Covers

Master's Thesis

by

Maximilian Thiessen
2819718

in fulfillment of requirements for degree
Master of Science (M.Sc.)

submitted to
Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III
Knowledge Discovery and Machine Learning Group

in degree course
Computer Science (M.Sc.)

First Supervisor: Prof. Dr. Thomas Gärtner
TU Wien

Second Supervisor: Prof. Dr. Stefan Wrobel
University of Bonn

Bonn, May 28, 2020

Acknowledgements

First, I want to express my sincere gratitude to my supervisor Prof. Thomas Gärtner. He has not only put a lot of time and effort in countless discussions but also helped me find my research direction, introduced me to the machine learning research community on various trips and always motivated me to surpass myself.

Of course, I am also deeply thankful to my other supervisor Dr. Tamás Horváth for guiding me through my whole studies and always helped me when I had questions. Moreover, I want to give the whole research group MLAI a big thank you, especially Dr. Pascal Welke, Eike Stadtländer, Dr. Michael Kamp, Florian Seiffarth and Till Schulz, for supporting me during the last few years.

I thank the Fraunhofer IAIS for providing computing resources, in particular, I thank Dr. Daniel Trabold and Ahmet Ocakli.

Lastly, I thank my friends Lukas Drexler, Tobias Elvermann, Oliver Kiss, Eike Stadtländer and Armin Schrenk for giving helpful feedback and my significant other Jacqueline Hammer for constant support.

Abstract

In this thesis, we discuss the use of convexity spaces for the problem of learning the labels of a graph in an active setting. This is motivated by the usefulness of the classical convexity notion in Euclidean spaces.

In particular, we assume the labelled classes to form geodesic convex subgraphs. This means that taking a shortest path starting and ending in one class forces it to be completely in this class. Using this assumption, we can develop and analyse new algorithms and label-independent bounds on the needed number of queries to actively learn the labelling of a graph. This is different from most previous approaches, where the bounds typically depend on properties of the labels themselves. We achieve this with so-called shortest path covers, which are sets of shortest paths jointly covering the graph's vertices.

To get provably good bounds, we provide a logarithmic approximation for the problem of finding a shortest path cover of minimum size.

We consider various different problem settings, ranging from binary and multi-class classification assuming each class to be convex in the graph, to more difficult ones, where only one of the considered classes is convex. Additionally, we develop strategies to explicitly allow a certain amount of prediction error to save labels.

We evaluate our new algorithms on various synthetic and real-world datasets and show that they are competitive with state-of-the-art approaches. Moreover, we discuss that we can still use our developed approaches, even though in practical situations the convexity assumptions are typically not globally satisfied, merely admitting some local structure.

Contents

1 Introduction

While the amount of collected data has massively increased over the last decades, the portion of labelled data, typically needed for learning algorithms, remains relatively small. Active learning procedures try to tackle this problem by letting the algorithms themselves be part of the often tedious annotation process. What has started as theoretical research in the 80s [Angluin, 1988] and had a renaissance in the 00s [Dasgupta, 2006, Balcan et al., 2009b, Zhu et al., 2005] slowly found its way to practical projects and is nowadays an essential tool used by many companies worldwide [Settles, 2011, Holzinger, 2016].

The main benefit of active approaches is that they usually need a lot less labelled data while achieving the same performance compared to non-active approaches. Moreover, they often come with accompanying bounds on the required amount of *labelled* data and performance guarantees. These can be used by practitioners to perform, for example, a cost assessment before even starting the procedure. Unfortunately, these guarantees typically depend on properties of the labels themselves and thus are often not very useful in practical situations. In this thesis, we make certain convexity assumptions on our data that allow us to develop new bounds, which solely depend on the structure of the unlabelled data, and therefore can be directly used by practitioners.

Besides, we not only have the data itself to work with but we often can measure the similarity between individual data points. For example, we can use additional background knowledge that describes the relationships of the data points or compute the similarity using the features of the data points. These similarities, together with the unlabelled data itself, can be represented as a graph forming the input to an active learning algorithm. In this graph representing the dataset, we will assume that the classes form *convex* subgraphs.

The motivation for this assumption is that traditional learning problems in Euclidean space often become easily solvable if the labelled classes form convex regions in the input space, while still staying practically useful and theoretically interesting. Classical examples are the Perceptron algorithm of Rosenblatt [1958] with the convergence theorem of Novikoff [1963] and the hard-margin SVM [Cortes and Vapnik, 1995].

For graph-based or even set-system based input spaces, the active learning community has not yet formalized and used any notions of convexity, even though there is a lot of theoretical research in fields like metric graph theory and convexity theory from the last few decades [Pelayo, 2013, Duchet, 1988, Kay and Womble, 1971]. The purpose of this thesis is to bridge the gap between the theoretical tools used in general convexity theory and learning problems on graphs to get meaningful insights, useful theorems and practical algorithms.

We will use the so-called geodesic graph convexity in a transductive learning setting on graphs. In particular, we will assume that having a shortest path in the graph starting and ending in the same class implies that the whole path belongs to this class, which seems to be a natural assumption in various application domains like community detection in social networks, opinion polls, disease spreading and drug discovery [Leskovec et al., 2009, Sabater and Sierra, 2002, Balcan et al., 2009a, Deshpande et al., 2005].

The goal of the active learning procedure is to accurately predict the labels of the whole graph by iteratively querying only a small subset of the vertices, relying on the convexity assumption. Our main new tool to design and analyse such procedures are so-called shortest path covers, being a set of shortest path jointly covering the vertices of the graph, enabling us to approach this problem in a structured way and develop bounds on the number of queries needed to completely infer the graph's labels.

We will discuss various problem settings ranging from binary and multi-class classification, where we assume each class to form a convex subgraph, to a more involved setting, assuming only one of the classes to be convex. In addition to it, we will turn these theoretically grounded approaches into practically efficient algorithms. Finally, we will evaluate our new approaches on various synthetic and real-world datasets, compare it to the state-of-the-art approach S^2 [Dasarathy et al., 2015] and additionally measure how well these datasets fit the convexity assumptions.

Related Work

The study of general convexity spaces can be traced back to almost 50 years ago [Kay and Womble, 1971, van De Vel, 1993]. One of the earliest uses of general convexity spaces in machine learning was the work of [Auer and Cesa-Bianchi, 1998] that deals with the task of online learning convex classes. We will mostly deal with a special kind of convexity space given by shortest paths of graphs called geodesic convexity [Pelayo, 2013]. Shortest path covers, our main tool for developing and analysing active learning algorithms, are used by Pan and Chang [2006] motivated by games on graphs.

Our main assumption on the data is that each class forms a convex subset of vertices in a weighted graph. As this, it can be seen of a generalization of the work of Missura and Gärtner [2011], where they study a similar problem on partially ordered sets, which can be seen as unweighted directed acyclic graphs. Furthermore, the idea to use binary search as a subroutine in our active learning algorithm is inspired by Nowak [2009], Emamjomeh-Zadeh et al. [2016], Dasarathy et al. [2015] and Gärtner and Garriga [2007].

Emamjomeh-Zadeh et al. [2016] studied a similar problem to ours, but instead on relying on the convexity assumption they use a more powerful query oracle to achieve bounds on the needed number of queries. Seiffarth et al. [2020] also studied geodesic and general convexity spaces in the machine learning context but with non-active approaches.

In our empirical evaluation we, among other things, test how close typical real-world graphs are to our convexity assumption. Marc and Šubelj [2018] conducted a similar study on various large-scale graphs but without considering labels.

2 Geodesic Convexity

Classifying data with binary labels forming convex regions in Euclidean space is one of the oldest [Rosenblatt, 1958] and most studied problems in machine learning. There exist efficient algorithms [see e.g. Maass and Turán, 1994, Servedio, 1999] for this problem and various theoretical guarantees on the performance of these methods are being developed up to this day [Diakonikolas et al., 2019].

We want to use the generalized notion of convexity on graphs and identify active learning classification problems that become efficiently solvable due to convexity assumptions. In particular, it enables us, similar to the Euclidean setting, to develop practical algorithms and bounds on the number of needed queries to learn the labels of a graph.

In this chapter, we will introduce the needed foundations of convexity theory on graphs and discuss our main tool, the shortest path cover, which will be the key to develop and analyse efficient active learning algorithms in the later chapters.

2.1 Basic Notions

We start this section with some basic notions [see e.g. Korte and Vygen, 2018] that we will use throughout the thesis.

A graph $G = (V, E)$ consists of a finite set V of *vertices* and a finite set E of *edges* between the vertices. Graphs can be *undirected*, meaning $E \subseteq \{e \subseteq V \mid |e| = 2\}$, or *directed* with $E \subseteq \{(x, y) \mid x, y \in V, x \neq y\}$. For any graph G, $V(G)$ denotes the vertex set of G and $E(G)$ its set of edges. A graph is *weighted* by a function $w : E(G) \to \mathbb{R}$. We can always assume a graph to be weighted by setting $w(e) = 1$ for all edges $e \in E(G)$. A graph $G' = (V', E')$ is a *subgraph* of another graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A subgraph is *induced* by a vertex set V' if it consists of all edges going from and to vertices in V', i.e. $E' = \{e \in E \mid e \subseteq V'\}$ or $E' = \{(v, w) \in E \mid v, w \in V'\}$ respectively for directed graphs. We will denote the subgraph of G induced by V' as $G[V']$. A *path* P is a graph, where for a $k \in \mathbb{N}$, the vertex set is $V(P) = \{v_1, \dots, v_k\}$ and the edge set $E(P)$ consists of exactly the edges going from v_i to v_{i+1} for all $i < k$. The vertices v_1 and v_k are the *endpoints* of P, where we also say P is a v_1-v_k-path. For any two vertices $x, y \in V(G)$, we can define a *distance*

$$d(x, y) = \min_{P\colon P \text{ is an } x\text{-}y\text{-path}} \sum_{e \in E(P)} w(e),$$

if there exists any x-y-path and ∞ otherwise. A path achieving this minimum for any pair of vertices $x, y \in V(G)$ is called a *shortest path*, or more precisely a w-shortest x-y-path. For two graphs

G_1 and G_2, let $G_1 \square G_2$ denote the *Cartesian graph product* [see e.g. Hammack et al., 2011] of G_1 and G_2. The vertex set $V(G_1 \square G_2) = V(G_1) \times V(G_2)$ is the Cartesian product of the vertex sets of G_1 and G_2. There is an edge between two vertices (x, x') and (y, y') in $G_1 \square G_2$ if and only if either

- $x = y$ and there is an edge connecting x' and y' in G_2, or
- $x' = y'$ and there is an edge connecting x and y in G_1.

An optimisation problem with cost function f has the form $\arg\min_x f_I(x)$ or $\arg\max_x f_I(x)$ for any instance I of the problem. We say an algorithm with polynomial runtime is a *g-approximation* for an optimisation problem if for any instance I with optimum solution x_I it finds an x such that

$$\max\left\{\frac{f_I(x_I)}{f_I(x)}, \frac{f_I(x)}{f_I(x_I)}\right\} \leq g.$$

The *approximation-ratio* g can be a constant $g \in \mathbb{R}_{\geq 1}$ or a function $g(I)$ depending on the instance I. For graph-related problems g is typically dependent on the number of vertices $|V|$.

Lastly, the function ln refers to the natural logarithm and log to the binary logarithm.

2.2 Convexity Spaces

Convexity spaces are a generalization of the Euclidean notion of convexity and its concepts to arbitrary set systems. The following definitions and propositions are largely based on the works of Kay and Womble [1971] and van De Vel [1993].

Definition 1: For a finite set X and a family of subsets $\mathcal{C}$ of X the pair $(X, \mathcal{C})$ is a *convexity space* if the two following conditions hold:

1. $\emptyset, X \in \mathcal{C}$
2. $\mathcal{C}$ is closed under taking arbitrary intersections

For the rest of this section, we will assume a finite ground set X. For infinitely-sized ground sets additional conditions are needed [van De Vel, 1993]. We call any $F \in \mathcal{C}$ *closed* or *convex*. Closely related to convexity spaces is their closure operator.

Definition 2: Given a convexity space $(X, \mathcal{C})$, the mapping $\sigma : 2^X \to \mathcal{C}$, defined as

$$\sigma(F) = \bigcap \{C \in \mathcal{C} \mid F \subseteq C\}$$

for all $F \subseteq X$, is called a *closure operator*.

We call $\sigma(F)$ the *closure* or *(convex) hull* of F. By definition, the closure of a set F is the smallest convex set containing F. A closure operator σ fulfils the following properties.

Proposition 3: *Let σ be a closure operator of a convexity space $(X, \mathcal{C})$. Then the following statements are true:*

- $F \subseteq \sigma(F)$ *for each* $F \subseteq X$ *(extensive)*

- $F \subseteq F'$ *implies* $\sigma(F) \subseteq \sigma(F')$ *(monotone)*
- $\sigma(\sigma(F)) = \sigma(F)$ *(idempotent)*
- $\sigma(\emptyset) = \emptyset$ *(normalized)*

Additionally, the fixed points of the closure operator are exactly the convex sets.

Proposition 4: *For a convexity space* $(X, \mathcal{C})$ *and its closure operator* σ, $F \in \mathcal{C}$ *if and only if* $\sigma(F) = F$.

And thus there is a one-to-one relationship between a convexity space $(X, \mathcal{C})$ and its closure.

Proposition 5: *Given a set* X *and a mapping* σ *defined on the power set of* X, *if* σ *fulfils the properties of proposition 3 it will uniquely define a convexity space* $(X, \mathcal{C})$ *where* $\mathcal{C} = \{F \subseteq X \mid \sigma(F) = F\}$.

We introduce a new concept based on convex sets, which we call the *convex shadow*.

Definition 6: For a convexity space $(X, \mathcal{C})$ with closure operator σ, we call the mapping $\tilde{\sigma}$ defined by

$$\tilde{\sigma}(A; B) = \{x \in X \mid \sigma(B \cup \{x\}) \cap A \neq \emptyset\}$$

for all $A, B \subseteq X$ the *convex shadow operator*. $\tilde{\sigma}(A; B)$ is the *convex shadow* of the set A *under* B.

We can easily check that this forms a new closure operator:

Theorem 7: *For a convexity space* $(X, \mathcal{C})$ *with closure operator* σ *and any fixed* $B \subseteq X$, *the convex shadow operator* $\tilde{\sigma}(\cdot; B)$ *fulfils the three properties of proposition 3.*

Proof. Extensivity, monotonicity and normalization follow directly.

For idempotency take an $x \in \tilde{\sigma}(\tilde{\sigma}(A; B); B)$. We have to show that $x \in \tilde{\sigma}(A; B)$. By the definition of x, there must be a $y \in \sigma(B \cup \{x\}) \cap \tilde{\sigma}(A; B) \neq \emptyset$. Since $y \in \tilde{\sigma}(A; B)$ this implies $\sigma(B \cup \{y\}) \cap A \neq \emptyset$. Additionally, as $(X, \mathcal{C})$ is a convexity space, $\sigma(B \cup \{x\}) \cap \sigma(B \cup \{y\})$ is convex. B and y are contained in $\sigma(B \cup \{x\}) \cap \sigma(B \cup \{y\})$ because σ is extensive. Together with the fact that $\sigma(\cdot)$ always returns the smallest containing convex set, we have

$$\sigma(B \cup \{y\}) \subseteq \sigma(B \cup \{x\}) \cap \sigma(B \cup \{y\}) \subseteq \sigma(B \cup \{x\}).$$

Finally, this yields

$$\sigma(B \cup \{x\}) \cap A \supseteq \sigma(B \cup \{y\}) \cap A \neq \emptyset$$

and so $x \in \tilde{\sigma}(A; B)$. This shows the idempotency of $\tilde{\sigma}$, as the other inclusion is clear by the extensivity of $\tilde{\sigma}$. □

For some intuition of this operator, assume we know that the convexity space admits a partition into two convex sets and that A is in one part of the partition and B in the other. Then $\tilde{\sigma}(\sigma(A); B)$ gives us the elements of X that are *behind* A from B's *point of view*, thus they must be in A's half of the partition because otherwise, they would violate the convex partition assumption.

In the rest of this work, we focus on convexity spaces induced by paths on graphs.

2.3 Interval Convexity Spaces on Graphs

There exist various convexity definitions for graphs. We mostly deal with the geodesic convexity [Pelayo, 2013], based on shortest-paths. Additionally, we use the monophonic convexity [Dourado et al., 2010]. Both convexities are based on the notion of a *closed interval*.

Definition 8: A convexity space $(X,\mathcal{C})$ is an *interval* convexity space, if there exists a mapping $I : X \times X \to 2^X$ called *(closed) interval* such that for all $F \subseteq X$,

$$F \text{ is closed if and only if } \bigcup_{x,y \in F} I(x,y) = F.$$

We will write $I(F)$ as a shorthand for $\bigcup_{x,y\in F} I(x,y)$. An example for an interval convexity space is the canonical convexity space in $\mathbb{R}^n$ induced by Euclidean shortest paths i.e. lines, with $I(x,y) = \{\lambda x + (1-\lambda)y \mid \lambda \in [0,1]\}$.

Let $G = (V,E)$ be a graph possibly weighted according to an edge weight function $w : E \to \mathbb{R}_{>0}$, where we have to restrict the weights to be strictly greater than zero due to technical reasons described in lemma 12. Using G, we can define a convexity space on the vertices $(V,\mathcal{C})$ called a *graph convexity space*. Typically, graph convexity spaces will admit an interval mapping $I(\cdot,\cdot)$ based on a set of paths $\mathcal{P}$ [Pelayo, 2013, Duchet, 1988]. That is to say, for all $x,y \in V$,

$$I(x,y) = \bigcup_{P \in \mathcal{P}} \{V(P) | P \text{ is an } x\text{-}y\text{-path}\}$$

and therefore G and $\mathcal{P}$ together induce a convexity space with

$$F \in \mathcal{C} \text{ if and only if } F \text{ contains all vertices from the paths in } \mathcal{P} \text{ starting and ending in } F.$$

Our focus will be on the *geodesic* convexity induced by the set of possibly weighted shortest-paths

$$\mathcal{P}_S = \{V(P) \mid P \text{ is a shortest } x\text{-}y\text{-path for some } x,y \in V\}$$

of G, where a set of vertices is convex if it contains all vertices lying on any shortest path joining two vertices of the set. We also call the subgraph induced by a convex vertex set convex. We denote the geodesic interval mapping as I_S and the geodesic closure operator as σ_S or simply I and σ if it is clear from the context. Additionally, we will also use the *monophonic* convexity given by the set of *induced* paths

$$\mathcal{P}_M = \{V(P) \mid P \text{ is an induced path in } G\}.$$

A path is induced if it is an induced subgraph of G. Put differently, there is no edge between any two vertices of the path but the edges of the path themselves.

We have the following useful property combining geodesic and monophonic convexities.

Proposition 9: *Given an unweighted undirected graph $G = (V,E)$ and its geodesic convexity given by $I_S, \sigma_S, \mathcal{P}_S$ and the monophonic convexity by $I_M, \sigma_M, \mathcal{P}_M$, the following relations hold for all $F \subseteq V$:*

- $\mathcal{P}_S \subseteq \mathcal{P}_M$

- $I_S(F) \subseteq I_M(F)$
- $\sigma_S(F) \subseteq \sigma_M(F)$

Proof. The claims follow by the fact that each shortest path must be an induced path in the unweighted graph, otherwise, there would be an edge which can be used as a *shortcut*. □

Lastly, we will need the following definitions. In the undirected and unweighted case, a vertex x is *simplicial* if x and its neighbours are fully connected, i.e. there is an edge connecting each pair of vertices. If the undirected graph is additionally weighted, we will call a vertex x simplicial if each edge between two neighbours y,z of x has less weight than the path from y to z over x, i.e. $w(\{y,z\}) < w(\{y,x\}) + w(\{x,z\})$. For directed graphs, a vertex x is simplicial if for any two vertices $y,z \in V(G)$ with existing edges (y,x) and (x,z) there is also an edge (y,z), and again in the weighted case additionally $w(y,z) < w(y,x) + w(x,z)$. Notably, all vertices with degree one or zero are simplicial.

Another property of a graph G is the *diameter* $\operatorname{diam}(G)$. It is usually defined for unweighted graphs and represents the size of the longest shortest path, i.e.

$$\operatorname{diam}(G) = \max\left\{|E(P)| \;\middle|\; P \in \mathcal{P}_S\right\}.$$

We will also use the same definition of diameter even if G is weighted, thus the diameter is the maximum number of edges that are used in any weighted shortest path.

Figure 1 summarizes the convexity spaces used in this thesis.

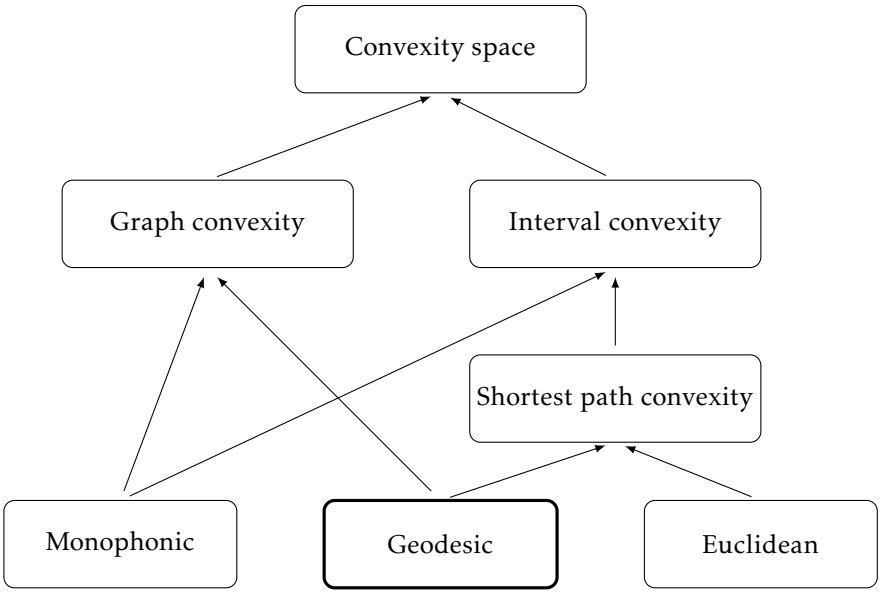


Figure 1: *Summary of the convexity spaces in consideration. The arrows represent "is"-relationships*

2.3.1 Computing Convex Hulls

One benefit of a finite interval convexity space $(V, \mathcal{C})$ with interval mapping I and closure operator σ is that we can iterate the computation of the closed interval to eventually get a convex set. It follows directly that this must be the convex hull.

$$\text{For all } F \subseteq V : \sigma(F) = I^k(F),$$

for a large enough $k \in \mathbb{N}$, where I^k is I composed $(k-1)$ times with itself.

Still, efficiently computing convex hulls for a graph-based convexity space is not always a straightforward task. For example in the geodesic convexity, we can not simply enumerate all possible connecting shortest paths from $\mathcal{P}_S$ between any two vertices, since there could be exponentially many of them. Nevertheless, we have the following result.

Theorem 10: *Computing the closed interval $I(F)$ of any $F \subseteq V$ in the geodesic convexity space induced by a weighted graph $G = (V, E)$ is possible in $\mathcal{O}\big(|F| \cdot (|V| \log |V| + |E|)\big)$ time. Computing the convex hull $\sigma(F)$ is possible in $\mathcal{O}\big(|\sigma(F)| \cdot (|V| \log |V| + |E|)\big)$ time.*

Proof. We want to use Dijsktra's algorithm [Dijkstra et al., 1959] to compute distances, but as it is only applicable to directed graphs, we have to modify the possibly undirected graph G to a directed version by simply replacing each edge $\{x, y\}$ with the two directed edges (x, y) and (y, x) of same weight $w(\{x, y\})$. For each $f \in F$ perform the following procedure:

Add a new vertex $f^* \notin V$ to the graph and connect all vertices $f' \in F \setminus \{f\}$ to f^* with a directed unit-weight edge (f', f^*). Compute all the distances $d(f, \cdot)$ from f to all other vertices with one application of Dijkstra's algorithm. Reverse all the edges and run again Dijkstra's algorithm with f^* as the source to get all distances $d(\cdot, f^*)$ to f^*. Now go through all $v \in V$. If

$$d(f, v) + d(v, f^*) = d(f, f^*),$$

v lies on a shortest f-f^*-path. This check can be performed in constant time for each v. For a v fulfilling this condition, as subpaths of shortest paths are again shortest paths, there must be an $f' \in F$ s.t.

$$d(f, v) + d(v, f') = d(f, f'),$$

and thus v must be in $I(F)$. The set of all such v forms exactly $I(F)$.

The runtime to compute $I(F)$ is dominated by the $2|I(F)|$ calls of Dijkstra's algorithm, each with a runtime of $\mathcal{O}(|V| \log |V| + |E|)$ [Fredman and Tarjan, 1987]. For the computation of $\sigma(F)$ we have to apply the procedure iteratively to all new vertices added in each step, resulting in $2|\sigma(F)|$ application of Dijkstra's algorithm. □

In the unweighted case, the runtime reduces to $\mathcal{O}(|F| \cdot |E|)$, respectively $\mathcal{O}(|\sigma(F)| \cdot |E|)$, by using a breadth-first-search instead of Dijkstra's algorithm [Pelayo, 2013].

Even though there is an efficient way to compute convex hulls in the geodesic convexity, this is not always the case for other convexity spaces. For the monophonic convexity, it is even NP-hard to compute the closed interval and the convex hull [Dourado et al., 2010].

2.4 Covering a Graph with Shortest Paths

Our main algorithmic tool to actively learn on graphs are *shortest path covers*. We will use them to design various algorithms and to get an upper bound on the number of queries needed to infer all labels. In our context, a shortest path cover is a set of shortest paths that jointly covers all the vertices of a graph.

Definition 11: For a possibly weighted graph $G = (V, E)$ and its set of shortest paths $\mathcal{P}_S$, a set $\mathcal{S} \subseteq \mathcal{P}_S$ is called a *shortest path cover* for G if

$$V = \bigcup_{P \in \mathcal{S}} V(P).$$

Computing a minimum shortest path cover, i.e. one using a minimum number of paths, is in itself an interesting research problem, see for example the work of Pan and Chang [2006] motivated by the game *cops and robbers*. To the best of our knowledge, so far there are no hardness or approximation results regarding the minimum shortest path cover problem, see for example the recent survey of Manuel [2018].

In the following, we will discuss a logarithmic approximation to this problem using the classical greedy algorithm of Chvatal [1979] for the general set-cover problem. In this problem, we have a set system $(A, \mathcal{B})$ and the goal is to use the minimum number of sets $B \in \mathcal{B}$ to cover the set A. The greedy algorithm iteratively selects a set $B \in \mathcal{B}$ covering a maximum number of elements from A not already covered, until completion. It achieves a $(1 + \ln r)$ approximation ratio [Chvatal, 1979], where r is the size of the largest set in $\mathcal{B}$. It depends on a subroutine that computes the greedily selected set B covering the maximum number of new elements. In our setting, this translates to a single shortest path covering a maximum number of vertices not already covered. Computing such a path requires strictly positive edge weights.

Lemma 12: *Computing a shortest path P, having the maximum number of vertices $|V(P)|$ among all shortest paths, is NP-hard for edge weights $w : E(G) \to \mathbb{R}_{\geq 0}$.*

Proof. Take an instance of the NP-hard Hamiltonian path problem [Garey and Johnson, 1979], where the goal is to check whether a given unweighted graph G has a path P using all the vertices of the graph i.e. $V(P) = V(G)$. Set $w(e) = 0$ for all edges, which results in all paths being shortest paths with total weight 0. Computing a shortest path P with a maximum number of vertices among all shortest paths, thus among all paths, immediately solves the Hamiltonian path problem. □

We can still solve this problem for strictly positive weights using a generalization of Dijkstra's algorithm [Dijkstra et al., 1959]. For this, we will need the following definition.

Definition 13: [Sobrinho, 2001] Let S be a set with a binary operation $\oplus$ and an order relation $\preceq$. Let two distinguished elements $\overline{0}, \overline{\infty} \in S$ be in S. We call the structure given by $\left(S, \oplus, \preceq, \overline{0}, \overline{\infty}\right)$ a *generalized weight* if it fulfils the following five properties.

1. $(S, \oplus, \overline{0})$ is a monoid:
   - S is closed under $\oplus$: $a \oplus b \in S$, for all $a, b \in S$
   - $\oplus$ is associative: $a \oplus (b \oplus c) = (a \oplus b) \oplus c$, for all $a, b, c \in S$
   - $\overline{0}$ is the identity: $a \oplus \overline{0} = \overline{0} \oplus a = a$, for all $a \in S$
2. $\overline{0}$ is a least element: $\overline{0} \preceq a$ for all $a \in S$
3. $\overline{\infty}$ is an absorptive element: $a \oplus \overline{\infty} = \overline{\infty} \oplus a = \overline{\infty}$, for all $a \in S$.
4. $\preceq$ is a total order on S:
   - $\preceq$ is connex: $a \preceq b$ or $b \preceq a$ for all $a, b \in S$
   - $\preceq$ is anti-symmetric: $a \preceq b$ and $b \preceq a$ implies $a = b$, for all $a, b \in S$
   - $\preceq$ is transitive: $a \preceq b$ and $b \preceq c$ implies $a \preceq c$, for all $a, b, c \in S$
5. $\oplus$ is isotone for $\preceq$: $a \preceq b$ implies both $a \oplus c \preceq b \oplus c$ and $c \oplus a \preceq c \oplus b$ for all $a, b, c \in S$

Indeed, as the following lemma states, we can apply Dijkstra's algorithm to generalized weights.

Lemma 14: *[Sobrinho, 2001] Let $\left(S, \oplus, \preceq, \overline{0}, \overline{\infty}\right)$ be a generalized weight and G a graph with edge weights $w^* : E(G) \to S$. Applying Dijkstra's algorithm to (G, w) with a starting vertex $s \in V(G)$, yields w^*-shortest s-t-paths for all vertices $t \in V(G)$. This means for all $t \in V(G)$ the returned path P_t has minimum total weight regarding $\preceq$ and $\oplus$:*

$$\bigoplus_{e \in E(P_t)} w^*(e) \preceq \bigoplus_{e \in E(P')} w^*(e),$$

for any s-t-path P'.

This enables us to compute a shortest path having maximum vertices.

Lemma 15: *For a graph $G = (V, E)$ and a weight function $w : E(G) \to \mathbb{R}_{>0}$, finding a shortest path P using the maximum number of vertices, i.e. with maximum $|V(P)|$ among all shortest paths, is possible in time $\mathcal{O}(|V|(|E| + |V| \log |V|))$.*

Proof. Modify w to the tuple $w^*(e) = (w(e), -1)$, lexicographically ordered by $\preceq$, i.e. for tuples $(a_1, a_2), (b_1, b_2) \in \mathbb{R}_{>0} \times \mathbb{Z}_{\leq 0}$, we have $(a_1, a_2) \preceq (b_1, b_2)$ if and only if $(a_1 < b_1)$ or $(a_1 = b_1$ and $a_2 \leq b_2)$. Let $\overline{0} = (0, 0)$, where we define the relation $\preceq$ for the zero element to be $\overline{0} \preceq (a_1, a_2)$ if and only if $0 \leq a_1$. Let $\overline{\infty} = (\infty, -\infty)$, where ∞ is a symbol representing an element that is larger than all elements in $\mathbb{R}_{>0}$ and $-\infty$ being smaller than all elements in $\mathbb{Z}_{\leq 0}$. We define comparisons with $\overline{\infty}$ again lexicographically. Let $\oplus$ be the canonical vector addition on the Euclidean plane $\mathbb{R}^2$ extended to $\overline{\infty}$ such that it becomes absorptive. Let $S = (\mathbb{R}_{>0} \times \mathbb{Z}_{\leq 0}) \cup \left\{\overline{0}, \overline{\infty}\right\}$.

We now show that $\left(S, \oplus, \preceq, \overline{0}, \overline{\infty}\right)$ is a generalized weight and thus lemma 14 allows us to apply Dijkstra to the weights w^*.

The first and the third condition regarding only $\oplus$ and not $\preceq$ are immediately true, because they hold in $\mathbb{R}_{>0} \cup \{0, \infty\}$ and $\mathbb{Z}_{\leq 0} \cup \{0, -\infty\}$ for each component. It is also easily checked that $\preceq$ defines a total order on S. For that, we remark that the anti-symmetric property holds for all elements in $\mathbb{R}_{>0} \times \mathbb{Z}_{\leq 0}$ compared with zero, because the first component is never 0. Additionally, $\overline{0}$ is a least element, because for any $(a_1, a_2) \in S \setminus \{\overline{0}\}$ the first component is by definition larger than zero $a_1 > 0$ and thus $\overline{0} \preceq (a_1, a_2)$ follows independent of a_2. Lastly, $\oplus$ is isotone because the regular addition "$+$" is isotone in each component regarding the canonical order "$\leq$". Again it is crucial that we do not have any elements of the form $(0, x)$ for an $x \neq 0$, because this could violate our defined order $\preceq$.

Compute a shortest w^*-weighted path for all pairs of vertices by applying Dijkstra's algorithm to each $v \in V$ on (G, w^*) and take the path P having the maximum $|V(P)|$ of those. This path P will minimize the weight $\bigoplus_{e \in E(P_t)} (w(e), -1)$ and thus is a shortest x-y-path with respect to the original weights w for some vertices $x, y \in V(G)$ by the definition of $\preceq$. Additionally, among all w-shortest x-y-paths, P will have the maximum number of vertices $|V(P)|$ as it also minimizes the amount of -1 in the second component per edge in the path. Overall, by construction P has the maximum number of vertices $|V(P)|$ among all shortest paths.

As the runtime to check $\preceq$ and perform vector addition $\oplus$ is not more than 2 times longer than computing their respective counterparts with regular real-valued weights, the overall runtime amounts to $|V|$ application of Dijkstra's algorithm each taking $\mathcal{O}(|E| + |V| \log |V|)$ steps. □

Putting everything together gives us the logarithmic approximation.

Theorem 16: *There exists a $(1 + \ln(\operatorname{diam}(G)))$-approximation for the minimum shortest path cover problem.*

Proof. We apply the greedy algorithm for the general set cover problem of Chvatal [1979] to get a shortest path cover $\mathcal{S}$. For this, we need to compute a shortest path that uses the maximum number of uncovered vertices. For the first shortest path we can apply the procedure described in lemma 15. For the following pathss we have to adjust the weights w^* to not account for already covered vertices. Let $\mathbb{1}_{\{A\}}$ be the indicator function, i.e. 1 if statement A is true and 0 otherwise. Then we can define for each directed edge (a, b) with weight $w(a, b)$ the modified weight

$$w^*(a, b) = \left(w(a, b), -\mathbb{1}_{\{b \text{ is not covered}\}}\right).$$

With the same argumentation as in lemma 15, the described procedure therein will yield a shortest path using a maximum number of not yet covered vertices. This allows us to apply the greedy algorithm and shows the approximation ratio, as in our case the size of the largest set is $r = \operatorname{diam}(G) + 1$.

The runtime of this approximation is $\mathcal{O}(|V|^2(|E| + |V| \log |V|))$ or $\mathcal{O}(|V|^4)$ in dense graphs, because we have to apply the procedure of lemma 15 with a runtime of $\mathcal{O}(|V|(|E| + |V| \log |V|))$ at most $|\mathcal{S}| \leq |V|$ times. □

Said differently, for any graph G with minimum shortest path cover $\mathcal{S}^*$, we can find a shortest path cover $\mathcal{S}$ in polynomial time such that $|\mathcal{S}| \leq |\mathcal{S}^*|(1 + \ln(\operatorname{diam}(G)))$.

Settling the hardness of this problem remains an open challenge. We assume that there might be a reduction of related covering problems like in the work of Dumitrescu and Jiang [2015] and Kumar et al. [2000], proving APX-hardness or even forbidding an approximability better than $\mathcal{O}(\log n)$ unless $P = NP$.

2.5 Hull Sets

Our second algorithmic tool for the analysis of the number of queries comes from the notion of *hull sets*, yielding lower bounds. For general convexity spaces $(X, \mathcal{C})$ and its induced closure operator σ, a hull set is a set $F \subseteq X$ whose closure is the whole space: $\sigma(F) = X$. For the geodesic convexity of a graph $G = (V, E)$, there exists the related notion of a *geodesic set*, that is a set R such that $I(R) = V$. A geodesic set is a hull set but the reverse direction does not hold. Moreover, the size of the minimum geodesic set can be arbitrarily larger than the one of a minimum hull set [Pelayo, 2013].

We have the following useful lemma.

Lemma 17: *Any hull set of a graph G contains all simplicial vertices of G*

Proof. This is a direct generalization of the proof of Pelayo [2013, Theorem 2.1] for the undirected and unweighted case. It follows by the fact that any simplicial vertex v will never be in a convex hull of a set $F \subseteq V(G) \setminus \{v\}$ not already containing it. Independent of whether the graph is weighted or directed, by the definition of a simplicial vertex, any shortest path not starting and ending in v will skip it, because otherwise this otherwise lengthen the path. □

Regarding the computational complexity of these quantities, we have the following results. Computing a minimum geodesic set is NP-hard, $\log|V|$-approximation is not possible unless P=NP, but at least there exists one with a ratio of $\mathcal{O}(\sqrt[3]{|V|}\log|V|)$ [Chakraborty et al., 2019]. Computing a minimum hull set is APX-hard [Coelho et al., 2015] and to the best of our knowledge, there is no known approximation algorithm with sublinear ratio, in $|V|$, for this problem. Nevertheless, the problem is solvable in polynomial time for graphs of bounded treewidth [Kanté et al., 2019], but the described approach is rather of theoretical interest. Checking whether a vertex is simplicial can easily be done, by simply checking the existence of the corresponding edges and possibly their weights.

To get some intuition for the difficulty of this problem, we show that the greedy strategy performs poorly on this problem:

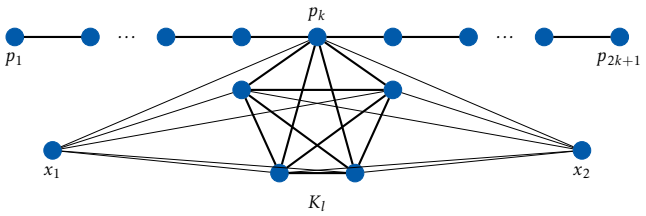


Figure 2: *Counterexample for a greedy hull set algorithm*

Proposition 18: *A greedy algorithm for the minimum hull set problem, selecting vertices one by one maximizing the increase of the closure size, can produce a hull set being $|V|/8$ times as large as the minimum hull set.*

Proof. Let $l, k \in \mathbb{N}$ with $2k+1 > l+2$. Construct the following graph. Start with a path with vertices $\{p_1, \dots, p_{2k+1}\}$ for some k. Fully connect the vertex p_k and $l-1$ new vertices $v_1, \dots, v_{l-1}$, resulting in a complete graph K_l. Lastly, add two new vertices x_1 and x_2 both connected to each vertex of K_l. The situation for $l = 5$ is shown in fig. 2.

Clearly, the minimum hull set is formed by p_1, p_{2k+1}, x_1 and x_2. This can be seen as they are indeed a hull set and all simplicial, thus mandator by lemma 17.

A greedy algorithm starting with a pair of vertices maximizing the convex hull would pick p_1 and p_{2k+1} having a hull of size $2k+1$, as this is larger than all other possibilities. In particular, the hull size of x_1 and x_2 is $2+l$ and the size of any hull of x_1 or x_2 with any path vertex p_i is at most $k+1$, both smaller than $2k+1$ by our assumption on k and l.

The greedy algorithm will now proceed by iteratively selecting one vertex maximizing the size of the convex hull. Having started with the path, each remaining vertex x_1, x_2 or any of the v_i increases the hull on its own only by one. Thus, the algorithm may select each of the vertices v_i in $V(K_l) \setminus \{p_k\}$ one by one and in the end adding x_1 and x_2 yielding a hull set of size $l+3$.

If $2k+1 = l+3$, the overall number of vertices in the graph is $|V| = 2k+1+l-1+2 = 2l+4$ and thus, for larger and larger l the computed hull set approaches a size of $\frac{|V|}{2}$, whereas the optimum was 4, which is independent of $|V|$. □

As we are primarily interested in the size of the minimum hull set and not the set itself, any lower bound is helpful. By lemma 17, we know that the number of simplicial vertices s in the graph is a simple lower bound which additionally can easily be computed by scanning each vertex and checking the corresponding edge conditions.

At least in the unweighted and undirected case, we can do better. As mentioned in proposition 9 the monophonic convex hull is always a superset of the geodesic convex hull and so every geodesic hull set is also a monophonic one. Thus, the size of a minimum monophonic hull set lower bounds the size of the minimum geodesic hull set. Additionally, every simplicial vertex is also part of the monophonic hull by the same argument as in lemma 17, as induced paths are also shortest paths. This is summarized in the following lemma.

Lemma 19: *[Pelayo, 2013, Section 2.3] For an unweighted and undirected graph with s simplicial vertices, we have*

$$s \leq |\textit{minimum monophonic hull set}| \leq |\textit{minimum geodesic hull set}|.$$

This can be seen as relaxing the problem from shortest paths to induced paths. Interestingly, even though it is NP-hard to compute a monophonic convex hull, we can still compute the minimum monophonic hull set efficiently in $\mathcal{O}(|V|^3|E|)$ time [Dourado et al., 2010], at least for undirected and unweighted graphs. Unfortunately, this lower bound is not tight as the size of the minimum

geodesic hull set can be arbitrarily far away from the size of a minimum monophonic hull set [Pelayo, 2013, Theorem 2.11].

To bound the minimum hull set H^* from above, we can take the endpoints of each path in a shortest path cover $\mathcal{S}$, which yields a hull set, and so get the following simple upper bound: $|H^*| \leq 2|\mathcal{S}|$.

2.6 Convex Graph Partitions

Lastly, we want to state some important aspects of convex graph partitions, as this will be our main assumption on the data. A proper convex r-partition of a graph is a partition of the vertex set into r convex non-empty subgraphs. Interestingly, deciding whether a graph admits such a partition is NP-hard for all $r \geq 2$ [Artigas et al., 2011].

Some graph classes have a special structure making them more appropriate for this setting. For example, graphs having the *Kakutani* property. A graph is Kakutani if it is possible to extend any two sets with non-overlapping convex hulls to a convex bipartition. This extension can be efficiently performed with the procedure of Seiffarth et al. [2020]. A particular graph class having this property is the set of graphs not having $K_{2,3}$, which is the complete bipartite graph having $2+3$ vertices, as a minor [Seiffarth et al., 2020]. Even though this class is rather limited by only allowing, for example, up to $|E(G)| \leq 2(|V(G)|-1)$ edges [Chudnovsky et al., 2011], it still contains practically useful graphs like outerplanar graphs [Horváth et al., 2010].

3 Active Learning on Labelled Convex Subgraphs

We use the following model of active learning. We are given a graph $G = (V, E)$ and edge weights w. There exists some binary vertex labelling function $\lambda : V \to \{0, 1\}$ that is unknown. The algorithm can iteratively query the label $\lambda(v)$ of a single vertex v and will receive the true (non-noisy) label. This problem was often studied before [Cesa-Bianchi et al., 2010, Settles, 2009, Dasarathy et al., 2015].

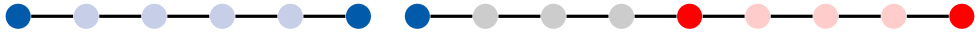
We are interested in upper and lower bounds on the number of queries needed to either

1. identify λ completely, or
2. identify λ up to a small relative error tolerance ϵ.

In particular, we want to emphasize that we are interested in *model-free* bounds, which do not depend on the labels λ but only on the graph structure. This is in contrast to most previous work, where usually bounds are based on some properties of the labels, such as the size of the *concept-cut* [Cesa-Bianchi et al., 2010, Settles, 2009, Blum and Chawla, 2001] and *clusteredness* [Dasarathy et al., 2015]. In section 4.3, we discuss one typical example of such bounds. It is not possible to get such bounds for arbitrarily labelled graphs. Therefore, we will make convexity assumptions regarding the labels.

3.1 Active Learning on a Convex Bipartition

We start with the simple setting where both classes, the positive $V_+ = \{v \in V(G) \mid \lambda(v) = 1\}$ and the negative $V_- = V \setminus V_+$, are convex in the graph G regarding its geodesic convexity, i.e. $\sigma(V_+) = V_+$ and $\sigma(V_-) = V_-$. This also means that the hulls are disjoint, $\sigma(V_+) \cap \sigma(V_-) = \emptyset$, and so they form a convex bipartition. Afterwards, we will look at the case where only the positive class V_+ is convex.



(a) Here we can directly infer the labels of the whole path using just the endpoints

(b) Here we have to resort to binary search

Figure 3: *Two cases when querying shortest paths*

3.1.1 Complete Identification of the Labels

The main result in this section is a querying algorithm using shortest path covers. The main idea is based on binary search. With an already computed shortest path cover, we start by iteratively querying the endpoints of each path. If the labels of the endpoints are the same, we can directly assume the whole path to have this label, due to the convexity assumption, as illustrated in fig. 3(a). Otherwise, they are not the same, but then we know, again by convexity, that the path has exactly one edge in the middle where the label switches from positive to negative. We can easily find this point by binary search, as shown in fig. 3(b).

Algorithm 1: Shortest-path-cover-based Querying (SPC)

Input: Graph G, shortest path cover $\mathcal{S}$
Output: Labelling l s.t. $l(v) = \lambda(v)$ for all $v \in V(G)$
1 $l[v] = \emptyset$ for all $v \in V(G)$
2 **foreach** $P \in \mathcal{S}$ **do**
3 query labels $l[x]$,$l[y]$ of the endpoints x, y of P
4 **if** $l[x] = l[y]$ **then**
5 $l[v] := l[x]$ for all $v \in V(P)$
6 **else**
7 `BinarySearch`(P, l)
8 **return** l

This immediately results in the baseline algorithm 1. It computes a labelling l that in the end will equal the true labelling λ. `BinarySearch` is a subroutine that labels the whole path P, by a procedure similar to binary search, finding the two vertices where the label flips. It iteratively bisects the unqueried region, by querying the vertex closest to the midpoint of the unqueried region of the path. One of the endpoints will have the same label as the new one, and thus, we can assign the region between the two exactly this label. Figure 3(b) shows the first step of this procedure.

Theorem 20: *Algorithm 1 correctly identifies λ, for all weighted graphs given any shortest path cover $\mathcal{S}$, labelled according to a convex bipartition, in polynomial runtime, using $\mathcal{O}(|\mathcal{S}|\log(\mathrm{diam}(G)))$ queries.*

Proof. The correctness of the algorithm is straightforward. If the condition in line 4 holds, $l(v) = \lambda(v)$ for all vertices $v \in V(P)$, as the algorithm queried the endpoints and the other vertices must have the same label by the convexity assumption and the fact that P is a shortest path.

If the condition is not true, we resort to the `BinarySearch` subroutine, which finds the two vertices where the label changes. Then by the convexity assumption, the algorithm correctly labels the rest.

For each path $P \in \mathcal{S}$, algorithm 1 starts by querying the two endpoints of the path and is either directly finished or continues with binary search. In the latter case, let l be the remaining number of unknown vertices on the path, so $|V(P)| - 2$ in the beginning. `BinarySearch` will at least halve

l with every query in the midpoint, or a vertex closest to it, of the path. In particular, with every query, independent of whether l is odd or even, l will go down to at least

$$\left\lfloor \frac{l}{2} \right\rfloor \geq \frac{l}{2},$$

as we always infer the possibly larger half of the remaining vertex labels. And so, the number k of bisections must fulfil

$$\left(\frac{1}{2}\right)^k l < 1,$$

implying that $k = \lfloor 1 + \log l \rfloor$ is always enough.

For all P with $|V(P)| \leq 2$, algorithm 1 will query $|V(P)|$ times and for larger ones it will use

$$2 + \lfloor 1 + \log(|V(P)| - 2) \rfloor$$

queries. Summed over all paths we get

$$\sum_{P \in \mathcal{S}} 2 + \lfloor 1 + \log |V(P)| \rfloor \leq |\mathcal{S}| \left(2 + \lfloor 1 + \log(\operatorname{diam}(G)) \rfloor\right)$$

as an upper bound on the number of queries made by algorithm 1.

This also bounds the runtime by $\mathcal{O}(|V(G)| + |\mathcal{S}| \log(\operatorname{diam}(G)))$. □

If the paths in the shortest path cover overlap a lot, we can use the following proposition to get a slightly better bound. It is stated in a more general way than needed here, because we will use it later, as well.

Proposition 21: *Let G be a graph, labelled in a way such that the positive class is convex and P be a shortest path in the graph with vertices $V(P) = \{v_1, \ldots, v_k\}$ and edges going from v_i to v_{i+1} for all $i < k$. Take a subgraph of P, which will be a set of subpaths, and close the emerged gaps by connecting each vertex with the highest index in one subpath to the vertex having the next index available. This results in a new path P'. P' will be labelled in such a way that the positive class will be convex in P' itself. Additionally, if G was labelled according to a convex bipartition, P' will also have this property.*

Proof. The proposition directly follows from the fact that if P' would violate the convex labelling, the original path P would violate it as well. □

This gives us the following improved bound:

Theorem 22: *For a graph G, labelled according to a convex bipartition with shortest path cover $\mathcal{S} = \{P_1, \ldots, P_k\}$, we can use*

$$\sum_{i=1}^{k} 2 + \left\lfloor 1 + \log \left| V(P_i) \setminus (V(P_1) \cup \cdots \cup V(P_{i-1})) \right| \right\rfloor$$

queries to completely identify the correct labelling

Proof. As the path P_i' resulting from extending $V(P_i) \setminus \bigcup_{j<i} V(P_j)$, as described in proposition 21, is still labelled according to a convex bipartition, algorithm 1 can query it with binary search, instead of each full path P_i.

Applying the bound of theorem 20 to the $|V(P_i')|$ instead of the $|V(P_i)|$ yields the result. □

We want to emphasize that these bounds hold for all possible labellings λ and are hence worst-case bounds. Nevertheless, we can show that they are best possible for general graphs by providing examples where exactly this many queries are needed.

Proposition 23: *There exists a graph G with a minimum shortest path cover $\mathcal{S}^*$ such that any deterministic algorithm needs $\Omega(|\mathcal{S}^*|\log(\operatorname{diam}(G)))$ queries to completely identify all labels.*

Proof. Take the disjoint union of k paths each having l vertices. As the paths are not connected, we have to query each of them on their own, since using labels of one path does not allow to infer any label of the others.

Claim: Each path has to be queried $\Omega(\log l)$ times to completely identify the labels.

This can be seen by the following folklore information-theoretic argument. The path can be labelled in $l+1$ different ways. As the algorithm is deterministic and always identifies the correct labelling it must have $l+1$ different *ending states*. Additionally, its decisions are solely based on the queried labels. Querying only one vertex allows the algorithm to decide between two different states, querying two vertices 4 states, and so on. In particular, to have at least $l+1$ different ending states, the algorithm must query at least $\lceil \log(l+1) \rceil$ different vertices of the path and thus the claim is true.

As l is the diameter of this graph and the size of a minimum shortest path cover is k, the proposition follows. □

Additionally, we emphasize that by theorem 16 we can compute a shortest path cover $\mathcal{S}$ at most $\mathcal{O}(\log(\operatorname{diam}(G)))$ times as large as the minimum shortest path cover $\mathcal{S}^*$, resulting in an upper bound on the needed number of queries not only by $\mathcal{O}(|\mathcal{S}|\log(\operatorname{diam}(G)))$ but also $\mathcal{O}(|\mathcal{S}^*|\log^2(\operatorname{diam}(G)))$. We thus have a $\mathcal{O}(\log \operatorname{diam}(G))$-approximation on the needed number of queries in the worst-case.

Even though this bound is tight, there are graphs where it is rather loose:

Theorem 24: *For any $k, l \in \mathbb{N}$ there exists a graph G with $|V| = k^l$ vertices, the hypercube with k vertices along each edge, and a minimum shortest path cover $\mathcal{S}^*$ of size $|\mathcal{S}^*| \geq \frac{|V|^{1-\frac{1}{l}}}{l}$, but with only $\mathcal{O}(\log(kl))$ needed queries to identify all labels.*

Proof. For $l = 1$, G is the path P_k of length k, for $l = 2$, it is the $k \times k$-grid. For general l, G is $P_k \square \cdots \square P_k$, the $(l-1)$-fold Cartesian graph product of P_k with itself.

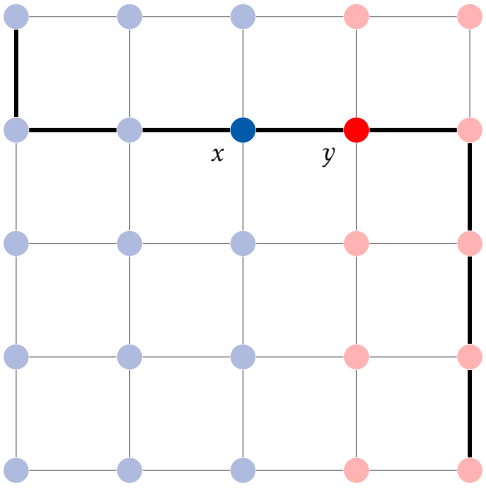


Figure 4: *Hypercube graph with side length $k = 5$ and dimension $l = 2$. One longest shortest path with its* splitting *edge $\{x, y\}$ is marked. The edge's labels uniquely determine the rest of the labels*

As the diameter of G is kl, each shortest path has length at most kl. Thus, for any shortest path cover $|\mathcal{S}|$, we must have $|\mathcal{S}|kl \geq |V|$, as it covers all vertices by definition. So,

$$|\mathcal{S}| \geq \frac{|V|}{kl} = \frac{k^l}{kl} = \frac{|V|^{1-\frac{1}{l}}}{l}.$$

On the other hand, using one longest possible shortest path is enough to infer the labels of the whole hypercube. First, query the endpoints of the path. If they have the same label, the whole hypercube must have this label, because any other point lies on some shortest path of these two vertices. If not, infer the labels on the path by using $\mathcal{O}(\log(kl))$ queries with binary search. There will be one *splitting* edge $\{x, y\}$ where the labels of its endpoints switch. The two convex shadows $\tilde{\sigma}(\{x\};\{y\})$ and $\tilde{\sigma}(\{y\};\{x\})$ will partition the graph and completely determine its labels. This can be seen by the following argument.

Because the graph is a Cartesian product of paths, we can identify any vertex v with l-dimensional coordinates in $\{1, \ldots, k\}$. The vertices x and y have the same coordinate values, but one, where they differ by one. Additionally, realize that any shortest path between two vertices in this graph will decrease the difference between the values in each coordinate one by one with each edge. Thus, we can reach any vertex v in G, where v is closer to y in the coordinate where x and y differ, with a shortest path by starting at x going to y. With the same argument, the other half of the vertices can be reached with a shortest path first going from y to x. However, as the labels of x and y differ, this shortest path determines the label of v, because only one of the two possible labels will not contradict the convexity assumption. Figure 4 illustrates the situation for $l = 2$. □

This means that by choosing k large enough for a fixed l, we can get arbitrarily close to a linearly-sized shortest path cover, even though binary searching one single path is enough.

This is where we can use a lower bound to get a better understanding of the number of queries we will need. While with the shortest path cover, we got a worst-case upper bound on the number of queries, which can be quite loose, we will get a lower bound using the number of simplicial vertices.

Theorem 25: *For any graph with convex binary classes and s simplicial vertices, at least s many queries are needed in the worst-case to correctly identify a labelling λ.*

Proof. Assume we could use less, say we query the set F with $|F| < s$. Then there exists a simplicial vertex $v \in V \setminus \sigma(F)$. By the definition of a simplicial vertex, v will not be in any convex hull, not already containing itself. This gives us two valid convex bipartitions. Either the whole graph has the same label or all but v have the same label and v the opposite one. There is no way to differentiate these two cases by only knowing the labels of the set F. □

For a special graph class we get a stronger bound:

Theorem 26: *For any $K_{2,3}$ minor-free graph G with minimum hull set H^*, at least $|H^*|$ many queries are needed in the worst-case to correctly identify a labelling λ.*

Proof. Assume we could use less, say we query the set F, which is not a hull set. Then there is a vertex $v \in V \setminus \sigma(F)$ outside of the convex hull. By the Kakutani property of the graph, we know that we can extend F and $\{v\}$ to a convex bipartition. Additionally, the whole graph having the same label is also a valid convex bipartition. Differentiating these two cases is not possible by only querying the set F. □

Unfortunately, this result only holds for a minimum hull set H^* and not for any hull set. Since there are no reasonable approximations for H^*, the only thing we can do to get an actual lower bound is to use the number of simplicial vertices s in the graph or the size of a minimum monophonic hull set, which is also a lower bound on the needed queries by lemma 19.

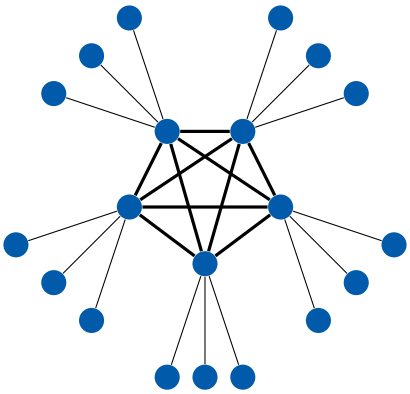


Figure 5: *Example graph with $l = 5$ and $k = 3$ for the loose behaviour of the hull set bound*

We emphasize that this bound is a rather pessimistic bound being true for some worst-case labellings. There are graphs and labellings where fewer queries can be used. For example, take the complete graph with l vertices and append to each vertex k new vertices resulting in a graph with $(k+1)l$ vertices. Figure 5 shows the graph with $k = 3$. All the vertices not on the complete graph are simplicial and thus by lemma 17 the minimum hull set size is at least kl, but we only have to query this much if the whole complete graph has the same label. In this situation, there could be at most one single simplicial vertex with a different label as the rest and so we have to query all of them in the worst-case to find it. In all the other cases where the complete graph is not

labelled with one colour, we can directly infer the labels of the other vertices. So in total, there are $2kl + 2$ labellings where we have to query the full hull set at least of size kl and $2^l - 2$ labellings where querying the complete graph is enough to infer the rest, resulting in k times fewer queries. Hence, for most of the labellings, a lot fewer queries are needed than the bound might suggest. The main reason for this pessimistic behaviour is the model-free nature of the hull set bound.

3.1.2 Partial Identification of the Labels

Often in practice, we are not interested in perfectly inferring the labelling λ of a graph G, but we can allow a certain amount ϵ of relative error. The question is then if we can do better and save some queries while guaranteeing an accuracy of $1 - \epsilon$, hence an absolute error of at most $\epsilon|V(G)|$.

The answer to this question is yes and the idea is to simply stop the binary search procedure in algorithm 1 *early*, instead of always querying until the end.

Theorem 27: *Given a graph G, labelled according to a convex bipartition with a shortest path cover $\mathcal{S} = \{P_1, \dots, P_k\}$ and an error threshold $\epsilon \in (0,1)$, we can use*

$$\mathcal{O}\left(\mathcal{S} \log \frac{1}{\epsilon}\right)$$

many queries to guarantee an error of at most $\epsilon|V(G)|$.

Proof. We apply algorithm 1 to the modified paths P_i', constructed from $V(P_i) \setminus \bigcup_{j<i} V(P_j)$ as described in proposition 21, but stop the binary search procedure as soon as we can guarantee an error of at most $\epsilon|V(P_i')|$ on the path P_i'. In particular, compared to the proof of theorem 20, instead of needing k queries bisecting the path such that

$$\left(\frac{1}{2}\right)^k |V(P_i')| < 1,$$

we will only need

$$\left(\frac{1}{2}\right)^k |V(P_i')| < \epsilon |V(P_i')|,$$

resulting in $k = \left\lfloor 1 + \log \frac{1}{\epsilon} \right\rfloor$ being enough.

The region of size at most $\epsilon|V(P_i')|$ can be labelled arbitrarily, while the rest is guaranteed to be labelled correctly by the binary search.

Now, as all the P_i' are disjoint, we can simply sum up the errors made on each path to get a total error of at most

$$\sum_{i=1}^{k} \epsilon |V(P_i')| = \epsilon \sum_{i=1}^{k} \left| V(P_i) \setminus \bigcup_{j<i} V(P_j) \right| = \epsilon |V(G)|,$$

using at most

$$\mathcal{S}\left(2 + \left\lfloor 1 + \log \frac{1}{\epsilon} \right\rfloor\right)$$

many queries. □

We want to remark that applying the early-stopping binary search technique to the original paths P_i, instead of the modified P_i', is not enough and might lead to more queries, as we would possibly search the same region multiple times.

Additionally, realize that this bound does not depend on the number of vertices $|V(G)|$ in the graph, but solely on the size of the shortest path cover $\mathcal{S}$ and the error threshold ϵ. For example, to guarantee a relative error of at most 10%, we will need at most $6 \cdot |\mathcal{S}|$ queries, as $4 = \lfloor 1 + \log 10 \rfloor$.

If we choose an $\epsilon < 1/|V(G)|$ smaller than one vertex, we recover the improved bound of theorem 22 for the error-free case.

3.2 Positive Class Convex

To make our framework more applicable to real-world datasets we relax the assumptions and consider the case where we only assume the positive class to be geodesic convex, i.e. $\sigma(V_+) = V_+$ and the negative class V_- arbitrary. This section is related to the problem of actively learning an interval on the real line [Dasgupta, 2006].

3.2.1 Complete Identification of the Labels

Unfortunately, the promising results from the previous section do not generalize to the less restrictive assumptions here.

Figure 6: *Difficult example path if only the red positive class is convex*

Figure 6 gives an intuition why the case with only one convex class is difficult compared to the previous one: Binary search becomes ineffective. Essentially, we have to query all the vertices to find the red one.

This is only the worst-case, though. As long as at least one of the endpoints of the path has a positive label, we can apply binary search as discussed in the previous section. Nevertheless, complete identification of the labelling in any situation is only possible by querying *all* labels, as otherwise, we can not infer if there could be some positive vertex somewhere on the path. as depicted in fig. 6. Thus, we get:

Theorem 28: *To completely identify the labels of a graph, where only the positive class is convex, we need to query all the labels in the worst-case.*

3.2.2 Partial Identification of the Labels

If we allow making some small relative error ϵ as before in the convex bipartition case, we can still use a shortest path cover. Algorithm 2 goes iteratively through each path. In line 3, we make sure that we only consider the part of the path not being considered before, as described in proposition 21. Then, in the loop of line 5, we query the path evenly spaced with a distance

Algorithm 2: Shortest-path-cover-based Querying (ϵ-SPC)

Input: Graph G, shortest path cover $\mathcal{S} = \{P_1, \dots, P_k\}$, rel. error $\epsilon \in (0,1)$, labels p and n, where p represents the positive convex class

1 $l[v] = \emptyset$ for all $v \in V(G)$
2 **foreach** $i \in \{1, \dots, |\mathcal{S}|\}$ **do**
3 Construct P_i' from $V(P_i) \setminus \bigcup_{j<i} V(P_j)$ as described in proposition 21
4 Identify $V(P_i') = \{v_1, v_2, \dots\}$ with edges going from v_j to v_{j+1} for all $j < |V(P_i')|$
5 **foreach** $j \in \left\{1, \dots, \left\lfloor \frac{|V(P_i')|}{\lceil \epsilon |V(P_i')| \rceil} \right\rfloor\right\}$ **do**
6 query label $l\left[v_{j \cdot \lceil \epsilon |V(P_i')| \rceil}\right]$
7 **if** *all queried labels are negative* **then**
8 $l[v] := n$ for all $v \in V(P_i')$
9 **else**
  /* Bisecting the gap on the left side of the positive region */
10 $\text{left} = \max\left\{1, \min\{j \mid l[v_j] = p\} - \left\lceil \frac{\epsilon |V(P_i')|}{2} \right\rceil\right\}$
11 query label $l[v_{left}]$
  /* Bisecting the gap on the right side of the positive region */
12 $\text{right} = \min\left\{|V(P_i')|, \max\{j \mid l[v_j] = p\} + \left\lfloor \frac{\epsilon |V(P_i')|}{2} \right\rfloor\right\}$
13 query label $l[v_{right}]$
14 $\text{pos} := \sigma(\{v \in V(P_i') \mid l[v] = p\})$
15 $l[v] := p$ for all $v \in \text{pos}$
16 $l[v] := n$ for all $v \in V(P_i') \setminus \text{pos}$
17 **return** l

of $\left\lceil \epsilon |V(P_i')| \right\rceil$ apart. If all these queries yield negative labels, we simply predict the whole path as negative in line 8. Otherwise, we have found a positive vertex and proceed in line 11 and 13 by bisecting once the two possible gaps where a label flip might occur. Then, the algorithm labels all vertices between the positive vertices with smallest and largest index as positive in line 14 and 15. The rest is labelled as negative in line 16. The following theorem proves the correctness of this algorithm.

Theorem 29: *Let G be a graph with a shortest path cover $\mathcal{S}$. Algorithm 2 runs in polynomial time and uses $\mathcal{O}\left(\frac{|\mathcal{S}|}{\epsilon}\right)$ queries to identify the labels up to $\epsilon |V(G)|$ many.*

Proof. The polynomial runtime is clear.

The number of queries for each path P_i' in the loop of line 5 and 6 is

$$\left\lfloor \frac{|V(P_i')|}{\lceil \epsilon |V(P_i')| \rceil} \right\rfloor \leq \frac{|V(P_i')|}{\lceil \epsilon |V(P_i')| \rceil} \leq \frac{|V(P_i')|}{\epsilon |V(P_i')|} = \frac{1}{\epsilon}.$$

In total this gives, together with the optional two additional queries per path in line 11 and 13, at most

$$|\mathcal{S}|\left(2+\frac{1}{\epsilon}\right)=\mathcal{O}\left(\frac{|\mathcal{S}|}{\epsilon}\right)$$

queries.

Now we will show the correctness of the algorithm. For this purpose, let us inspect the gaps that may arise during the querying with the loop in line 5 and 6 of algorithm 2:

- The region before the first query has at most

  $$\left\lceil \epsilon|V(P_i')|\right\rceil - 1 \leq \epsilon|V(P_i')|$$

  vertices.
- The region between any two queries also has at most

  $$i\left\lceil \epsilon|V(P_i')|\right\rceil - (i-1)\left\lceil \epsilon|V(P_i')|\right\rceil - 1 = \left\lceil \epsilon|V(P_i')|\right\rceil - 1 \leq \epsilon|V(P_i')|$$

  vertices.
- Finally, we remark that we can write $|V(P_i')| = t\left\lceil \epsilon|V(P_i')|\right\rceil + z$ for

  $$t = \left\lfloor \frac{|V(P_i')|}{\left\lceil \epsilon|V(P_i')|\right\rceil} \right\rfloor$$

  and $z < \left\lceil \epsilon|V(P_i')|\right\rceil$ and so the region after the last query has again at most

  $$\begin{aligned} &|V(P_i')| - \left\lfloor \frac{|V(P_i')|}{\left\lceil \epsilon|V(P_i')|\right\rceil} \right\rfloor \cdot \left\lceil \epsilon|V(P_i')|\right\rceil \\ &= z \leq \left\lceil \epsilon|V(P_i')|\right\rceil - 1 \leq \epsilon|V(P_i')| \end{aligned}$$

  vertices.

Thus, the largest unqueried region on P_i' has a size bounded by $\epsilon|V(P_i')|$.

Additionally, we know that the positive region is convex in the original path P_i and thus by proposition 21 also in P_i', which divides the path in at most three regions, where the outer two have negative labels and the middle one is positive.

This means that if the condition in line 7 of algorithm 2 is true, the positive region can only be in one of the unqueried gaps of size smaller than $\epsilon|V(P_i')|$, implying at most this many wrongly predicted labels.

If it is false, the algorithm found at least one positive vertex. By the convexity assumption, we know that the region between the known positively labelled vertices must also be positive. Additionally, there are at most two gaps where the labels might switch. These are the two regions where we might make an error when labelling them as negative.

The algorithm bisects these two regions with a query as close to their midpoints as possible, which at least halves the possible error region when labelling everything before and after these midpoints as negative and in between as positive. This is exactly what the algorithm does in line 14-16. And so the prediction error is again bounded by

$$\frac{\epsilon|V(P_i')|}{2} + \frac{\epsilon|V(P_i')|}{2} = \epsilon|V(P_i')|.$$

Overall, the total prediction error is bounded by:

$$\sum_{i=1}^{k} \epsilon|V(P_i')| = \epsilon \sum_{i=1}^{k} \left| V(P_i) \setminus \bigcup_{j<i} V(P_j) \right| = \epsilon|V(G)|,$$

which shows the correctness of algorithm 2. □

A practical algorithm might, instead of using the two additional queries in the case of line 10-16, perform two binary searches in the at most two possible regions of error to achieve an error-free classification on this path with $\mathcal{O}(\log(\epsilon|V(P_i')|))$ more queries.

4 Practical Algorithms and Further Results

In this short chapter, we will talk about practical aspects of our developed algorithms, a multi-class generalization of our problem and discuss the relations of our shortest-path-cover-based querying bound to a traditional bound known from the literature.

4.1 Practical Algorithms

Besides our discussed shortest-path-cover-based algorithms, the following greedy approach, related to the closure algorithm known from the literature [see e.g. Auer and Cesa-Bianchi, 1998, and the references there], seems to be reasonable in practice:

Iteratively query the one vertex whose label would increase the number of known labels the most.

From a theoretical perspective, this corresponds to the discussed greedy algorithm in proposition 18 for the hull set computation and has the same drawbacks of being almost arbitrarily bad. Nevertheless, we can use this idea to enhance our shortest path cover-based querying algorithm. Instead of selecting any path and querying it until it is completely labelled, we can allow the algorithm to switch to more promising paths. Algorithm 3 shows a possible realisation of this idea. The *hull-like* function `f` can be any function that given a set of the same label returns a potentially larger set having the same true label, i.e. for any subset V'_+ of the positive convex class V_+ and any subset V'_- of the negative convex class V_-, the hull $f(V'_+)$ stays a subset of V_+ and $f(V'_-)$ stays a subset of V_-. In particular, the convex interval I, the $(k-1)$-fold convex interval I^k, the convex hull σ and also the convex shadow $\tilde{\sigma}$ is allowed as `f`. For the convex shadow operator as `f` we have to modify the update slightly to $pos := \texttt{f}(pos \cup \{c[P^*]\}; neg)$ and $neg := \texttt{f}(neg \cup \{c[P^*]\}; pos)$. We note that it is not necessary to apply the convex shadow operator multiple times to the same set, as it is idempotent by theorem 7.

Every path admits a new candidate vertex with the `NextCandidate` subroutine that emulates the behaviour of the original algorithm 1 with a queue by first returning the endpoints of the path and then if necessary iteratively the midpoints returned by the `BinarySearch` procedure. Therefore, we immediately get the following result using the proof of theorem 20:

Theorem 30: *Given a graph G with shortest path cover $\mathcal{S}$, algorithm 3 is correct and uses at most $\mathcal{O}(|\mathcal{S}|\log(\mathrm{diam}(G)))$ queries.*

Algorithm 3: Candidate-Maximization-based Querying (CM)

Input: Graph G, shortest path cover $\mathcal{S}$, $\mathtt{f} \in \{\sigma, I, I^2, \dots\}$

1 $l[v] := \emptyset$ for all $v \in V(G)$, $pos := \emptyset, neg := \emptyset$
2 **while** *exists* $v \in V$: $l[v] = \emptyset$ **do**
3 **foreach** $P \in \mathcal{S}$ **do**
4 $c[P] = \mathtt{NextCandidate}(P, l)$
5 $P^* := \arg\max_{P \in \mathcal{S}} \left|\mathtt{f}(pos \cup \{c[P]\}) \setminus pos\right| + \left|\mathtt{f}(neg \cup \{c[P]\}) \setminus neg\right|$
6 query label $l[c[P^*]]$
7 **if** $l[c[P^*]]$ *is positive* **then**
8 $pos := \mathtt{f}(pos \cup \{c[P^*]\})$
9 $l[p] := l[c[P^*]]$ for all $p \in pos$
10 **else**
11 $neg := \mathtt{f}(neg \cup \{c[P^*]\})$
12 $l[n] := l[c[P^*]]$ for all $n \in neg$
13 **return** l

The important difference to algorithm 1 is that we now select one vertex $c[P^*]$ from all these candidates as the most *promising* one to increase the number of known labels, as performed by line 5, instead of going through the paths in a fixed order. The maximization returns a path such that querying its candidate vertex would maximally increase the number of new labels, given by the hull-like function $\mathtt{f}$, summed over the two possible cases of positive and negative labels. This corresponds to the maximal *expected* increase of known labels if we assume the label of $c[P]$ to be equally likely positive and negative. The algorithm then proceeds by querying the new vertex and inferring more labels in the corresponding class with the function $\mathtt{f}$. This is repeated until the full graph is queried.

In practical situations, the two *tricks* of candidate maximization and hull computation in between each query, reduce the number of queries significantly, but they do not affect the theoretical bounds we derived in previous sections, as we will see in the empirical evaluation.

The maximization in line 5 is merely a heuristic, indeed we could use any type of candidate selection there. For example, a more conservative heuristic might maximize the minimum increase of the two new hulls with the new vertex instead, corresponding to the least increase of known labels we will gain querying this candidate. More involved strategies could use some background knowledge. For example, a label distribution over the graph, to maximize the expected increase of known labels. Typical active strategies are additionally often information-theoretically motivated, using entropy as a measure for uncertainty and querying the one vertex where the algorithm is most uncertain of its label [Settles, 2009].

4.2 Multi-Classification Problems

Up until now, we solely discussed binary classification problems, but in various practical situations, there are rather multiple classes $\{1, \dots, r\}$. It turns out as long as we assume all of the classes to

be convex in the graph, i.e. form a convex r-partition, the problem still can be solved with binary search using shortest path covers.

Theorem 31: *There exists a deterministic algorithm with polynomial runtime identifying all labels of a graph labelled according to a convex r-partition given a shortest path cover $\mathcal{S}$ with $\mathcal{O}(|\mathcal{S}|r\log(\mathrm{diam}(G)))$ queries. Additionally, the algorithm does not need to know r in advance.*

Proof. We will adapt algorithm 1 and the `BinarySearch` subroutine to the multi-class case. In fact, we only have to change the binary search slightly, as the rest of the proof of theorem 20 still holds, especially the fact that if the endpoints have the same label the whole path must have this label due to the convexity assumption.

If the endpoints have different labels, we will start by running the regular `BinarySearch` subroutine until it finds a vertex v with a label different to both labels of the endpoints of the current path. Then, we will recurse by running one new binary search on the subpath going from one endpoint to v and another binary search going from v to the other endpoint. This procedure continues until the whole path is labelled correctly. Either it will find the splitting point by a standard binary search or it will at some point find a new label class and again recurse.

Due to the convexity assumption, each not seen class label can only appear in one of the split subpaths, implying that overall the search procedure will run at most $r-2$ times in this new recursion case, as we start with two known label classes, while during the rest perform a standard binary search.

Moreover, before and after each of the at most $r-2$ recursion steps, there is a *period* of standard binary search with at most $\mathcal{O}(\log(\mathrm{diam}(G)))$ queries until a new recursion or completion. We thus have at most $2(r-2)$ periods of standard binary search, yielding at most $\mathcal{O}(r\log(\mathrm{diam}(G)))$ queries for each shortest path and $\mathcal{O}(|\mathcal{S}|r\log(\mathrm{diam}(G)))$ in total.

Note that this procedure does not need to know r in advance. □

We note that the approach behaves exactly as algorithm 1 when $r=2$. Typical multi-class approaches need to know the number of classes r in advance, making our approach applicable to recent multi-class novelty detection [Bodesheim et al., 2015] and open-world [Bendale and Boult, 2015] problem settings.

4.3 Relations to other Bounds

Traditional graph labelling bounds typically not only depend on the graph structure itself, as our approaches but also on some properties of the labelling λ [Cesa-Bianchi et al., 2010, Guillory and Bilmes, 2009, Blum and Chawla, 2001]. One of the most-used property is the size c of the *concept-cut* that denotes the number, or weight, of edges going from one class to the other:

$$c = \sum_{\{x,y\}\in E(G)} w(\{x,y\})|\lambda(x)-\lambda(y)|.$$

It is used for example by Guillory and Bilmes [2009] to bound the number of mistakes of the classical min-cut based algorithm of Blum and Chawla [2001] with predictions l and a queried set L to

$$\frac{1}{2\Psi(L)}\left(c + \sum_{\{x,y\}\in E(G)} w(\{x,y\})|l(x) - l(y)|\right),$$

where $\Psi(L)$ is an efficiently computable function solely depending on the graph structure and not λ.

If the size of the concept-cut is known, we can use it to strengthen our shortest-path-cover-based bound. In particular, if the concept-cut has unweighted size c and is smaller than the size of a computed shortest path cover $\mathcal{S}$, we can tighten the upper bound from $\mathcal{O}(S\log(\mathrm{diam}(G)))$ to $\mathcal{O}(c\log(\mathrm{diam}(G))+(|S|-c))$, as only at most c many shortest paths must be queried by binary search to find the c edges where the labels flip and the rest can be inferred by querying the endpoints and computing convex hulls, resulting in at most $2(|S|-c)$ additional queries.

5 Empirical Evaluation

In this section, we will evaluate our new methods on various synthetic and real-world datasets and compare them to the state-of-the-art algorithm S^2 [Dasarathy et al., 2015]. Additionally, we will measure how close the real-world datasets are to our convex bipartition assumption and evaluate if we still can apply tools like the convex hull for prediction, in situations where the assumption is not completely true.

S^2 algorithm The S^2 algorithm is one of the state-of-the-art graph-based active learning algorithms. It switches between two phases. In the first phase, it will randomly query the graph until it finds two different labels. Then it transitions into the second stage, where it will query the midpoint of a shortest path between two vertices with different labels of shortest length. Hence the name S^2: Shortest shortest path. The high-level goal is to find the edges in the concept-cut quickly. If there are no such paths it will go back to phase one. This repeats until the budget is used. Finally, it uses some label propagation technique [Chapelle et al., 2006, Chapter 11] to predict the remaining labels of the graphs, where we decided to use the established *local-and-global-consistency* approach of Zhou et al. [2004].

We implemented our algorithms, as well as S^2, in `python3.7` relying on the graph algorithms library `graph-tool2.29` [Peixoto, 2020]. The code is publicly available in a `GitHub` repository [Thiessen, 2020].

5.1 Experiments on synthetic graphs

Since we had difficulties finding real-world graphs completely fulfiling our convex partition assumptions. we resorted to generating various synthetic graphs. The results on these graphs should be seen as a promising first direction and can not be directly transferred to results on real-world graphs.

Graph generation We restricted the graphs to outerplanar ones since we can easily partition them into two convex sets, as discussed in section 2.6. In particular, we start with a cycle of size n and keep adding edges uniformly at random as long as the outerplanarity is preserved. If the graph is not outerplanar anymore we backtrack, i.e. remove the last edge, and continue. If this step fails more than $n/3$ times we stop and return the graph, without the last edge. We generated 5 graphs for each n in $\{50, 100, 200, 500\}$. This procedure was implemented in sagemath [The Sage Developers, 2020], as it offers a convenient way of checking the outerplanarity of a graph.

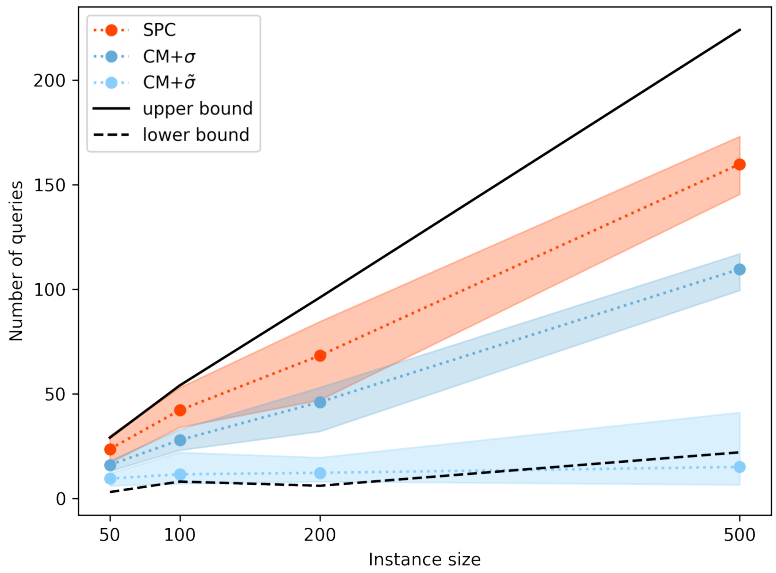


Figure 7: *Number of queries made by our approaches on the synthetic graphs*

Graph labelling We labelled each of the generated graphs in 10 different ways. This was achieved by starting with two vertices drawn uniformly at random and then performing the procedure described in the work of Seiffarth et al. [2020].

Figure 7 summarizes the results of our own approaches on the synthetic graphs. For each graph, we first precomputed one shortest path cover, as described in theorem 16, which is then used by all approaches. The coloured lines indicate the average number of queries needed to completely identify the graphs of one size. Additionally, the shaded area represents the 5% and 95% quantile of the measured number of queries needed. Lastly, the solid black line represents the maximum upper bound computed by theorem 22 and the dashed line the minimum lower bound over all graphs of the same size, given by the number of simplicial vertices and lemma 17.

Our baseline in red, denoted by SPC, is a direct implementation of the shortest-path-cover-based querying algorithm 1 without further adaptations. In blue and light blue we denote two different versions of the candidate-maximization (CM) algorithm 3, where the first uses the convex hull $\sigma(\cdot)$ as the maximization operator f and the latter the convex shadow operator $\tilde{\sigma}(\cdot;\cdot)$.

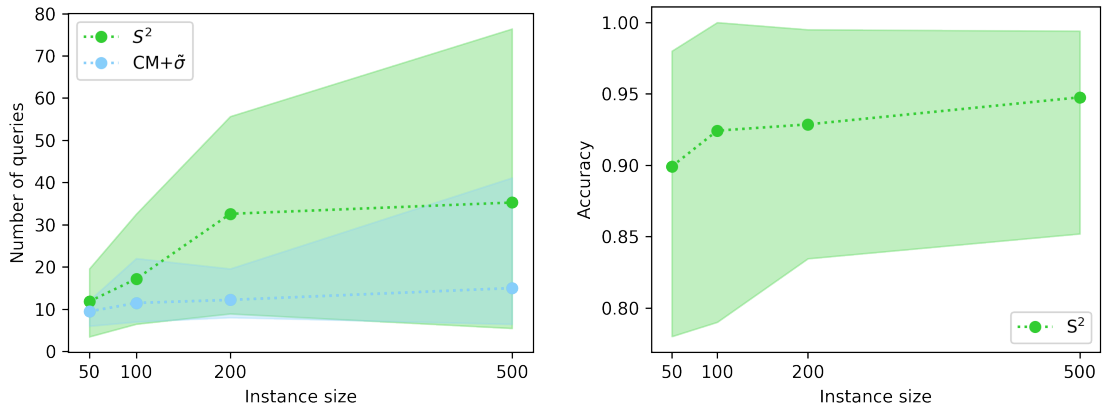
Already the baseline algorithm performs quite well, by querying less than half of the graph most of the time for smaller instances of size 50 and 100, and only a third of the graph for the larger ones of size 200 and 500.

As we can see by the blue line, the candidate maximization approach indeed helps a lot in practice. Typically, we almost need only half as many queries than the baseline approach to completely identify the labels. There are two main reasons for this improvement. First, by computing the closures of the negative and positive classes at the end of each iteration of the candidate-maximization approach we infer a lot of additional new labels compared to the baseline approach. Second, the candidate-maximization approach typically does not query a path until the end with binary search but jumps to more promising paths all the time, in contrast to the baseline

approach which always queries each path in isolation until all the labels of the current path are known.

The usage of the convex shadow operator instead of the simple hull again drastically decreases the number of needed queries to roughly 10 most of the time for all sizes. The reason is the following: As soon as all vertices on the decision boundary, i.e. the endpoints of edges with different labels, are known, the candidate-maximization approach with the convex shadow operator can stop, as opposed to the normal hull, because it can directly infer the rest.

Lastly, let us inspect the behaviour of the bounds. The upper bound is roughly at half the graph size and thus a useful bound in practice. Moreover, for the sizes 50 and 100, it is very close to the actual performance of the baseline algorithm 1. For the larger graphs it is further away, but still gives a first impression on how large the budget should be. The lower bound is interestingly almost on par with the number of queries used by the candidate-maximization algorithm 3 with the convex shadow operator. Remember that the lower bound is not necessarily true for all possible labellings, but only for some worst-case ones, explaining why sometimes on the graphs of size 500 the approaches use fewer queries. Nevertheless, this bound is also useful in practice, complementing the information we get from the upper bound and resulting in a relatively clear picture of the situation, at least on these synthetic graphs.



(a) Needed number of queries

(b) Accuracy

Figure 8: *Results of S^2 on the synthetic graphs*

To get a broader picture we compare these results with the performance of the state-of-the-art algorithm S^2, summarized in fig. 8. Figure 8(a) compares the needed number of queries to completely identify the graph of our best approach algorithm 3 with the convex-shadow operator to S^2. As S^2 is a randomized algorithm that tries to get the best possible accuracy using a fixed budget, we decided to iteratively increase the budget of S^2 until it hits a 100% accuracy for the first time. Here, we clearly see that on most of the graphs, especially for the larger ones, our approach needs significantly fewer queries to completely identify the graph. This is not surprising, as our algorithm has the additional knowledge of the convex bipartition, which S^2 can not use. Still, this gives a first hint on why the convexity assumption might be useful. Additionally, the number of queries needed varies a lot, especially compared to our approach, which is a drawback in practice.

To get a fairer comparison we also evaluated S^2 with exactly the number of queries our best approach used, so roughly 10, as the budget and plotted the resulting accuracies in the accompanying fig. 8(b). As we can see, most of the time S^2 achieves an accuracy higher than 90% or even 95%. This indicates that our generated synthetic instances seem to be rather easy for active learning algorithms, as S^2 performs really well, even without using the convexity assumption. The increased number of queries in the first experiment, to reach 100% accuracy, is mainly due to the fact that S^2 has no way of telling when it has finished.

5.1.1 Positive Class Convex

In this section, we will evaluate our approach on the more difficult setting of having only one convex class and the other arbitrary. We will again compare algorithm 2 with S^2. We used the same graphs as in the previous section but labelled them in a different way by randomly sampling a subset of vertices and setting its hull as the positive class and the rest as the negative.

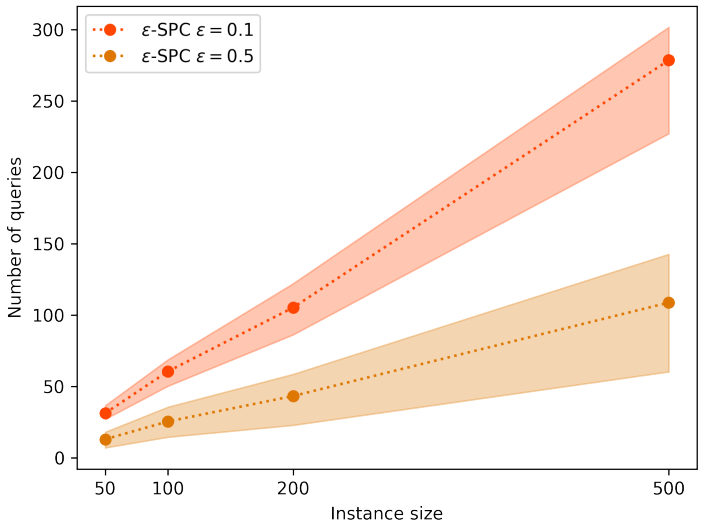
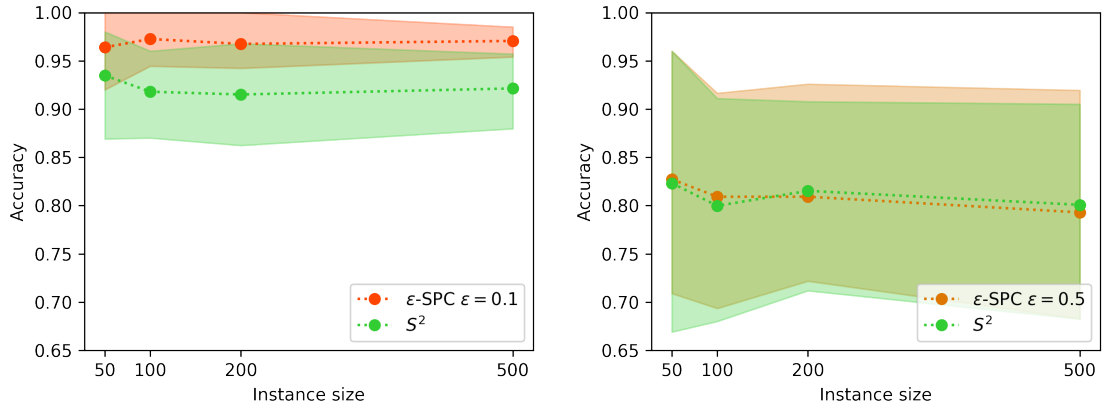


Figure 9: *Needed number of queries for $\epsilon = 0.1$ and 0.5 on the synthetic graphs*

Figure 9 shows the average number of queries a naive implementation of algorithm 2 needed on the synthetic graphs of different sizes for $\epsilon = 0.1$ and 0.5. For $\epsilon = 0.1$ roughly half of the graph was queried before the algorithm could guarantee an error of at most 0.1. In the case of $\epsilon = 0.5$ about a fourth was enough. Compared to the candidate-maximization algorithm 3 with the convex shadow operator from the previous section, these results seem rather bad. Still, the performance is comparable to the baseline algorithm 1, even though we relaxed the convex bipartition assumption in this evaluation. We emphasize that there is a lot of space to improve. For example, we are convinced that also in the one convex case a candidate-maximization approach would help a lot.

Inspecting the achieved accuracy of our approach, depicted by the red and orange curves in fig. 10, we clearly see that our implementation typically achieves much higher accuracy than needed. For the $\epsilon = 0.1$ case, the accuracy is rather at roughly 97% and for the $\epsilon = 0.5$, there is an

even larger gap to the typically over 80% achieved accuracy. On the one hand, this is a positive result as it shows that the algorithm is able to achieve much better accuracy than it is promising. But on the other hand, this suggests that one can probably design a similar algorithm also having a provable accuracy of at least $1 - \epsilon$, using significantly fewer queries than we currently do with our implementation of algorithm 2.



(a) Using the budget of our approach with $\epsilon = 0.1$

(b) Using the budget of our approach with $\epsilon = 0.5$

Figure 10: *Accuracy of S^2 on the synthetic graphs*

Additionally, fig. 10 also shows the performance of S^2 on these instances. It was run twice on each them with the budgets used by our approach with $\epsilon = 0.1$ and 0.5 respectively. Even though the budget is much higher than in the previous section, S^2 performs worse than before, indicating that these new labellings are a lot harder to learn.

Using the budget of our approach with $\epsilon = 0.1$, we see the accuracies of our approach and S^2 depicted in fig. 10(a). On average our approach performs around 2-5% better than S^2. For the case of $\epsilon = 0.5$, depicted in fig. 10(b), the results of our method and S^2 are almost indistinguishable.

This shows that also here the convexity assumption helps a lot to lower the needed number of queries, even when used with a naive implementation of algorithm 2.

The achieved 80-94% accuracy of S^2 with the budgets of $\epsilon = 0.1$ and $\epsilon = 0.5$ compared with the significantly lower budget to reach 100% accuracy in the previous convex bipartition case, clearly shows that the instances in this section are a lot harder even for state-of-the-art approaches.

5.2 Experiments on Benchmark Datasets

In this section, we will evaluate the performance of our approaches on typical benchmark datasets of the semi-supervised learning community. Even though our approaches were designed for the case where the convexity partition assumption holds, we still can run them on graphs not fulfilling this assumption. In practice, we do not expect the graphs to completely have convex classes, hence it is important to assess the performance of our approaches on more realistic cases than the synthetic graphs of the previous section.

We will use all the benchmark binary classification datasets from the semi-supervised learning book [Chapelle et al., 2006] but dataset 8, as it is too large to process with our current hardware. The dataset ids are 1,2,3,4,5,7 and 9. From these datasets, we built similarity graphs and dropped most of the edges by the following method.

We started with the complete graph weighted by the Euclidean distances of the data points. Then we computed for increasing quantile-sizes $q \in \{a \cdot b \mid a \in \{1, \dots, 9\}, b \in \{0.001, 0.01, 0.1\}\}$ the number of simplicial vertices s in the graph where we keep only the edges having a weight smaller than the q-quantile of all edge weights. Of all these graphs, we keep the one, where the number of simplicial vertices drops to 5% of the number of vertices for the first time, as it seemed to yield reasonable shortest path covers sizes in our experiments. For a small q near zero, the graph will consist almost completely of singleton vertices and for a q near one the graph will be close to a complete graph. Both situations induce a trivial convexity space, where all subsets are convex and the shortest path covers have a size of $|V(G)|/2$ or even $|V(G)|$ and thus we have to find a value in between resulting in a rich convexity structure. Additionally, one reason for many simplicial vertices is simply the number of connected components in the graph. We prefer to have a small number of components, typically allowing a large diameter and shortest path covers of small size. The sweet spot to achieve this seems to be at 5%. We emphasize that even though the q values are different for each dataset, we still applied the same procedure to detect it with the 5% rule without any further dataset dependent fine-tuning.

We tested each graph in two different versions, one unweighted and one weighted by the computed distances.

Table 1: *Main characteristics of the benchmark graphs*

	1		2		3		4		5		7		9	
$\|V(G)\|$	1500		1500		1500		400		1500		1500		1500	
$\|E(G)\|$	3750		55500		33000		3800		33000		33000		21744	
weighted	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
$\|\mathcal{S}\|$	318	321	345	394	470	446	97	107	536	553	499	546	303	327
convex paths	97%	98%	90%	93%	91%	92%	48%	56%	73%	75%	78%	77%	53%	60%

Table 1 summarizes the characteristics of the benchmark graphs. In particular, it contains the number of vertices and edges, whether the graph is weighted and the size of the computed shortest path cover and the percentage of paths in the cover fulfiling the convexity partition assumption on their own.

Before discussing the query evaluations, let us first inspect the convexity aspects of the datasets. As clearly can be seen by the last row of table 1 never reaching 100% not all of the shortest paths in each of the shortest path covers allow a convex partition regarding the true labelling. Thus, all the graphs are not labelled according to a convex bipartition. Still, remarkably, for most instances but 4 and 9 almost all of the shortest paths in the shortest path cover have on their own a convex partitioning.

This leads us to our first main takeaway for real-world use cases of our approaches. In practice, or at least on these benchmark graphs, it seems like a convex partition for the whole graph is seldom the case, but *locally* on each shortest path, the labellings *tend* to follow a convex partition.

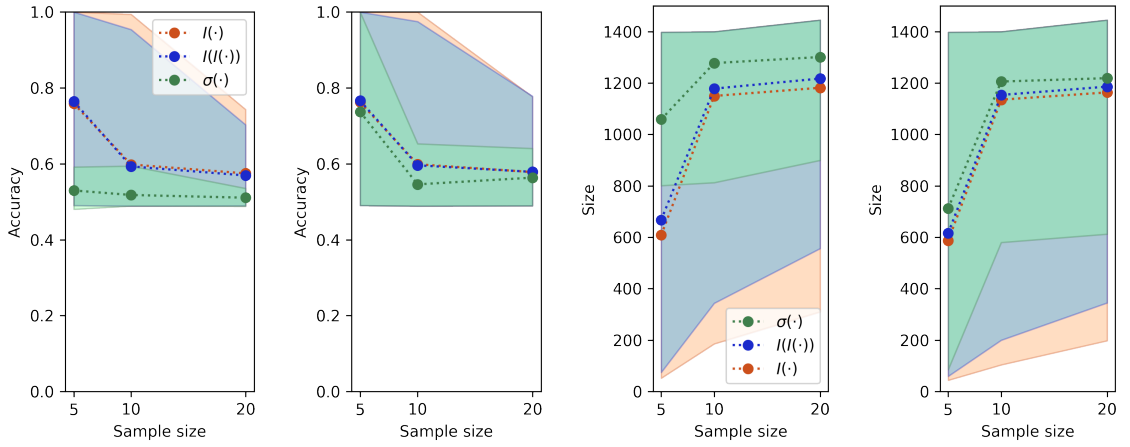
This allows to still exploit the local convexity, since in the end our baseline algorithm 1 does not care about the *global* convex partition assumption, but only makes a prediction error if one of the shortest paths is not labelled according to a convex partition. So, even without running the evaluations, we can say that the baseline algorithm 1 will perform quite well, as it will identify the correct labels for most of the paths.

Additionally, we can see in table 1 that for all but dataset 7 the number of convex paths increases significantly in the weighted case. It seems like more information, here given by the weights, forces the paths to behave more like a convex partition. The reason is that by adding weights, paths that were shortest paths in the unweighted case often lose this property in the weighted case. Moreover, the additional weighting seems to strengthen the behaviour of the paths to have at most one edge where the label flips. This also fits the typical assumptions in semi-supervised learning, like high clusteredness and small concept-cut [Chapelle et al., 2006, Dasarathy et al., 2015].

Until now, we only evaluated the local convexity of a graph by the number of shortest paths fulfilling the convexity partition assumption in a shortest path cover. It is also interesting to see how accurate convex hulls are on these datasets. Indeed, the prediction accuracy of the candidate-maximization algorithm 3 heavily depends on the quality of the computed hulls and not only on the shortest paths like it is with algorithm 1.

To assess the quality of the hulls, we conducted the following experiments. For each of the benchmark graphs and each class, namely positive and negative, we sampled 5 times from the class 5, 10 and 20 vertices. Then we computed the closed interval $I(\cdot)$, the repeated closed interval $I(I(\cdot))$ and the convex hull $\sigma(\cdot)$ of each sample and checked the resulting size and accuracy in terms of the number of correct labels. The results are depicted below for all 7 datasets for both the weighted and unweighted version. Again the bold lines represent the average accuracy, respectively size, and the shaded area represents the 5% and 95% quantile of the measured values.

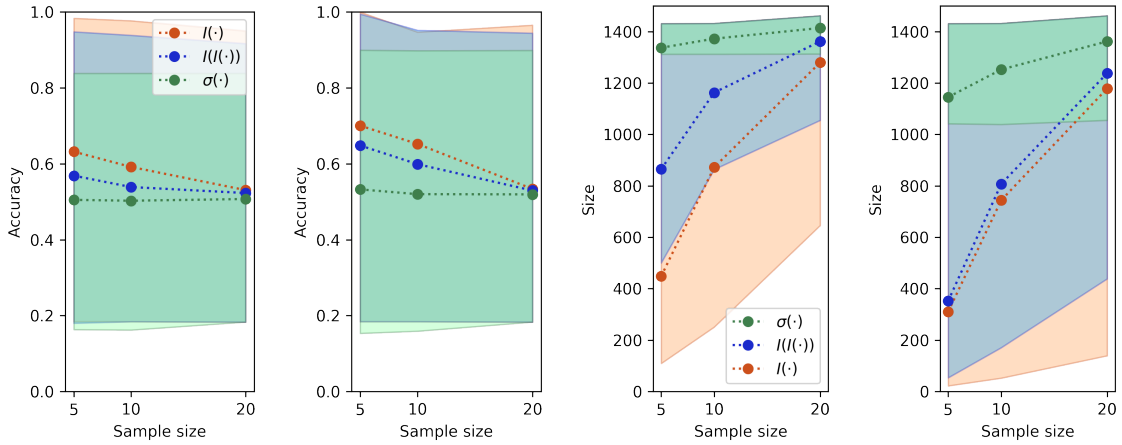
Let us first inspect each dataset individually.



(a) Unweighted accuracy (b) Weighted accuracy (c) Unweighted sizes (d) Weighted sizes

Figure 11: *Hull tests for dataset 1*

The results are promising for the first dataset, as can be seen in fig. 11. For the smallest sample size 5, the average accuracy of I and even I^2 are close to 80%, which is remarkable as the number of known labels goes from 5 to over 600 on average. For the larger sample sizes 10 and 20 the accuracy goes down to roughly 60%, due to the fact that already the closed interval I has a size of 1200 and I^2 and σ being even larger, while there are only 750 instances of each class. Interestingly, in the weighted version also the convex hull performs really well for the small samples of size 5, increasing the number of labels from 5 to 700 with an accuracy of 75% on average.



(a) Unweighted accuracy (b) Weighted accuracy (c) Unweighted sizes (d) Weighted sizes

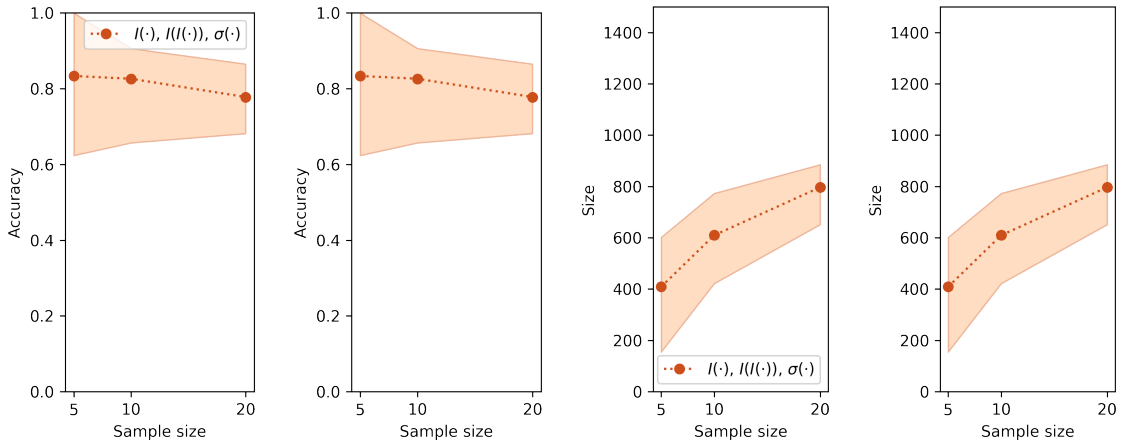
Figure 12: *Hull tests for dataset 2*

For the second dataset, depicted in fig. 12, the accuracy is at a rather bad level of 50 – 60% for all sample sizes and hull types but for the closed interval I and the smallest sample in the weighted case, where it averages to 70%. Again, the accuracy is better in the weighted case and for smaller sample sizes, mainly due to smaller hull sizes. Compared to the results on the previous dataset, the drop of accuracy can be explained by the large increase of the resulting hull sizes. It seems that this dataset is inherently further away from a convex bipartition than the previous one.

The third dataset is rather special, as can be seen in fig. 13. Here the closed interval on all samples was already the convex hull, thus all three different hull types yield the same results. Still, the accuracy is very high, especially for the convex hull, at almost 85% for the smallest samples of size 5. Even for the larger ones, the accuracy drops only to roughly 80%, which is really surprising, especially as the number of labels goes up from 20 to around 800. Adding the weights does not change the situation at all here.

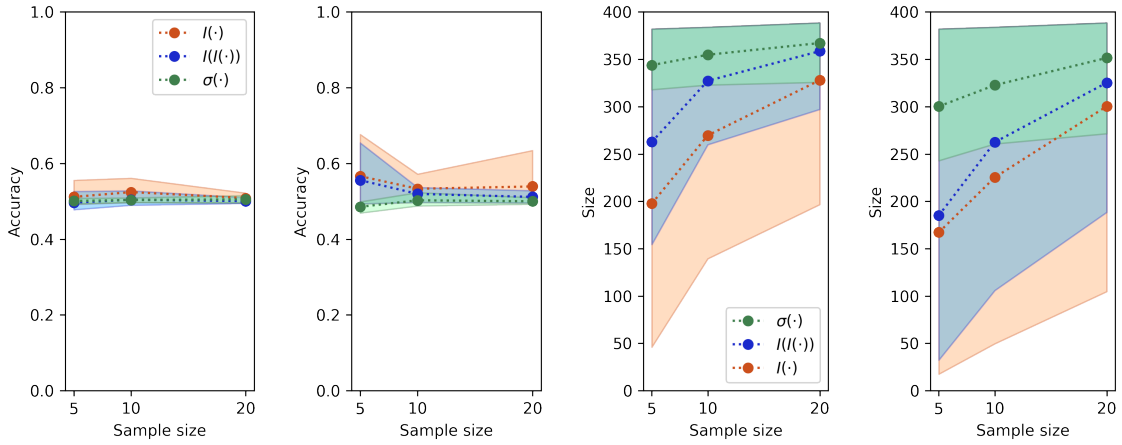
The results for the fourth dataset are presented in fig. 14. It is the only one with 400 vertices, all others have 1500. It seems to be furthest away from a convex partition, as the accuracy is on par with random guessing peaking at 56%. As usual, adding the weight helps a bit, but only to increase the accuracy to 50 – 65%. The hull sizes are typically at least half the graph or even almost the whole graph, which clearly explains the bad performance.

Figure 15 shows that the dataset with id 5 is also a good example for the usefulness of the additional edge weight, as it increases the accuracy roughly by 10% for I and I^2. It also supports the hypothesis that an edge weight reduces the number of shortests paths, as clearly can be seen by



(a) Unweighted accuracy (b) Weighted accuracy (c) Unweighted sizes (d) Weighted sizes

Figure 13: *Hull tests for dataset 3*



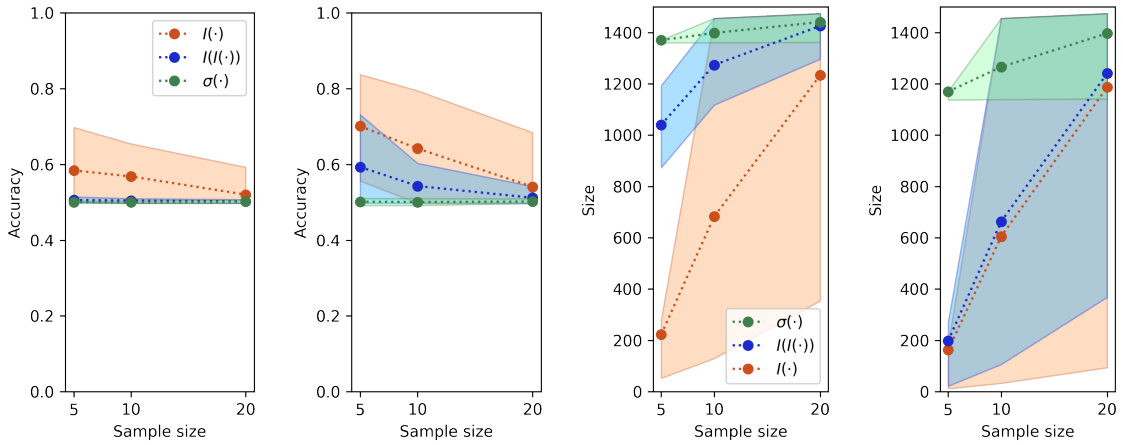
(a) Unweighted accuracy (b) Weighted accuracy (c) Unweighted sizes (d) Weighted sizes

Figure 14: *Hull tests for dataset 4*

comparing the sizes of I^2 in the unweighted and weighted case. The convex hull performs really bad, as on most previous datasets, which can be explained by its size being larger than 2/3 of the graph in each sample.

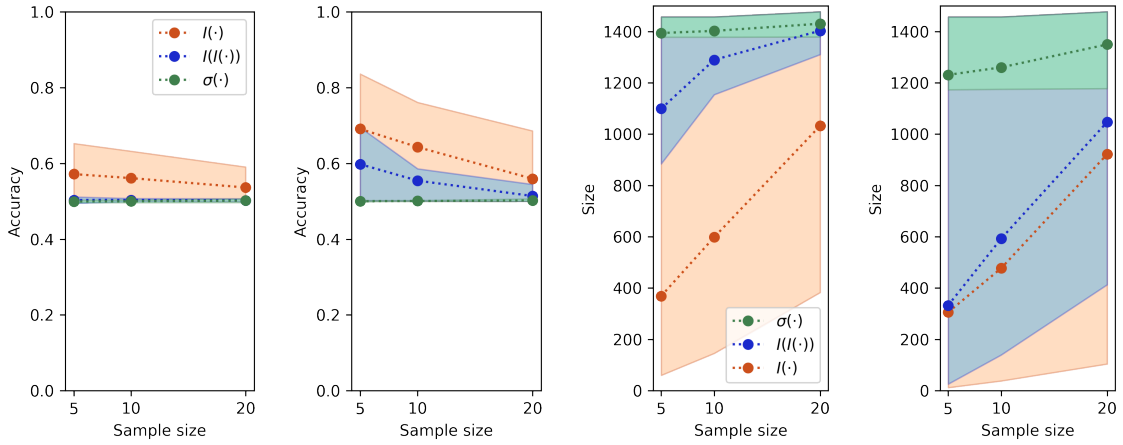
The results of the dataset with id 7 can be seen in fig. 16. Interestingly, this dataset behaves almost exactly as the dataset with id 5. It can be explained by the fact that they are generated in a very similar way by sampling from Gaussian distributions [Chapelle et al., 2006].

On the last dataset, with id 9, the accuracy is better than on the previous two, but it shows exactly the same trends regarding the accuracy and hull sizes. The convex hull sizes are here especially large, being almost the whole graph all the time, while the iterated closed interval is relatively low at least in the weighted case.



(a) Unweighted accuracy (b) Weighted accuracy (c) Unweighted sizes (d) Weighted sizes

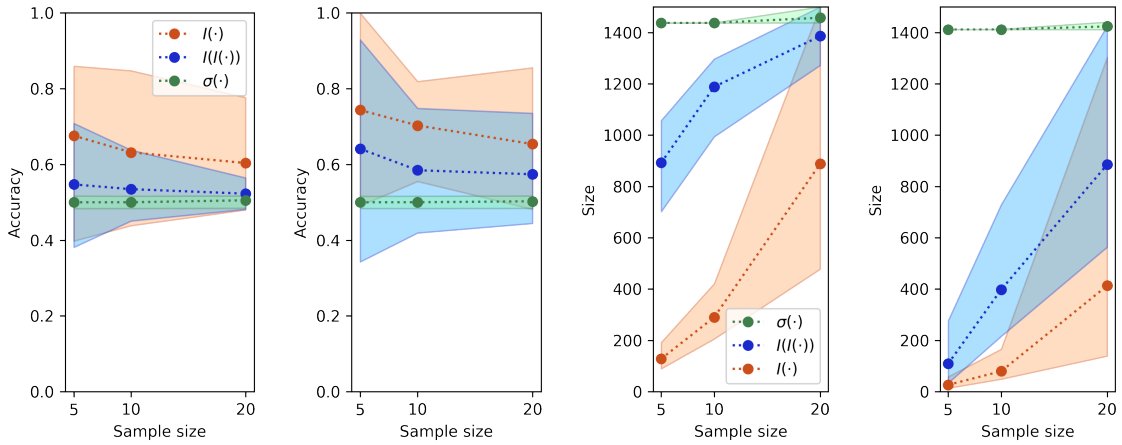
Figure 15: *Hull tests for dataset 5*



(a) Unweighted accuracy (b) Weighted accuracy (c) Unweighted sizes (d) Weighted sizes

Figure 16: *Hull tests for dataset 7*

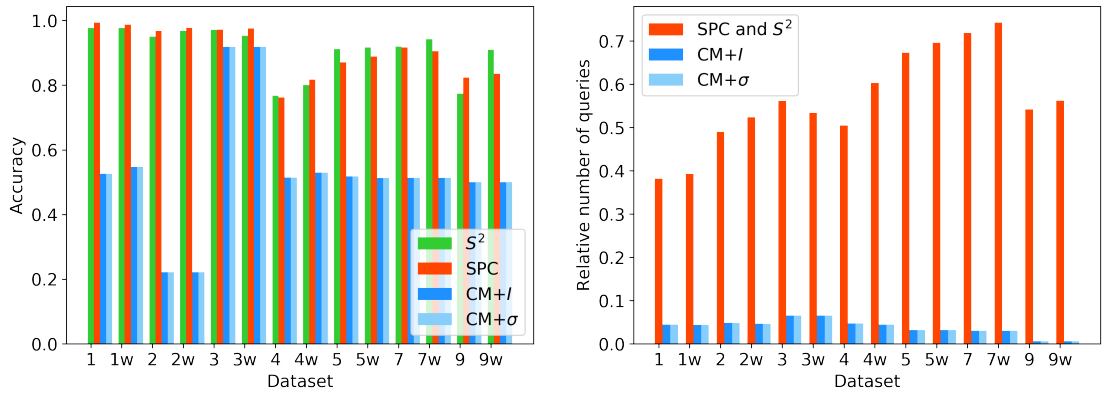
Overall, the trend for all the datasets is that the closed interval performs acceptably, but has quite a small resulting size. The repeated closed interval and the convex hull are a lot larger than the closed interval, but the accuracy quickly drops to typically almost 0.5, which is on average not better than random guessing. This confirms our local convexity assumption. As we go to larger and larger subsets, like produced by the convex hull operator, we leave the local convex features behind and are left with a subset which is not convex anymore. Moreover, not only larger hulls bring down the accuracy but also larger starting samples. Typically for the smaller samples of size 5, the accuracy is significantly better than random guessing, but for the larger samples of size 10 and 20, the hulls are often as large as half or even the whole graph, explaining the really bad accuracy. Nevertheless, the weight helps a lot by increasing the accuracy significantly, especially for the larger hulls, by reducing their size.



(a) Unweighted accuracy (b) Weighted accuracy (c) Unweighted sizes (d) Weighted sizes

Figure 17: *Hull tests for dataset 9*

To sum up, the local convexity hypothesis seems to be true for single shortest paths and also for closed intervals, but as soon as we go to larger structures like convex hulls, the convexity breaks.



(a) Accuracy comparison (b) Relative number of queries

Figure 18: *Querying results on the weighted (denoted with a "w" suffix) and unweighted benchmark datasets*

Query evaluation Finally, we discuss the actual performance of our approaches and S^2 on these datasets. As mentioned we run our baseline algorithm 1 and the candidate-maximization algorithm 3 without modification, which will introduce some prediction error as the convexity assumption does not hold anymore. Figure 18 shows the results for the weighted and unweighted versions of the 9 datasets. The accuracy of the baseline approach is surprisingly high, especially for the first three datasets using 600 to roughly 800 queries of the total 1500. On dataset 5 and 7 the accuracy is also not too bad, but the number of queries is around 1000, which is definitely too high.

The two candidate-maximization approaches used significantly fewer queries, but produce predictions that do not exceed random guessing on most datasets. We have already seen this

behaviour in the previous discussion about accuracies of I^2 and σ. The even worse accuracy of roughly 20% on the second dataset is due to its imbalanced classes, where the majority appears 80% of the time [Chapelle et al., 2006]. A really promising result is achieved on dataset 3, where we get an over 90% accuracy with under 100 queries. We already saw this behaviour on dataset 3, as the closed interval and the convex hull coincided on the random samples.

Interestingly, the two candidate-maximization approaches yield almost exactly the same results, differing only on the second dataset, where the approach with the convex hull used one less query to gain the same accuracy. The reason might be that in practice the choice of the hull-like function `f` does not matter much, resulting in the same candidates for I and σ. Additionally, both approaches will eventually have to query all the simplicial vertices, as they are not capable of inferring them.

As the prediction results were already so bad for the candidate-maximization with I and σ we did not run it with the convex shadow operator, as this would even further decrease the accuracy.

Finally, we also run the S^2 algorithm on these datasets using the number of queries our baseline algorithm 1 needed as the budget. The results are also depicted in fig. 18(a). As we can see most of the time it is on par with our baseline algorithm, being slightly worse on the datasets 1, 2 and 4 and better on 5,7 and 9, but this difference is not significant with a p-value of $p = 0.94$ given by the Wilcoxon signed-rank test [Wilcoxon, 1945]. An interesting fact is that typically all approaches, including S^2, gain in accuracy in the weighted case. This is remarkable because the original S^2 algorithm was designed solely for unweighted graphs and we applied it without further consideration to weighted shortest paths.

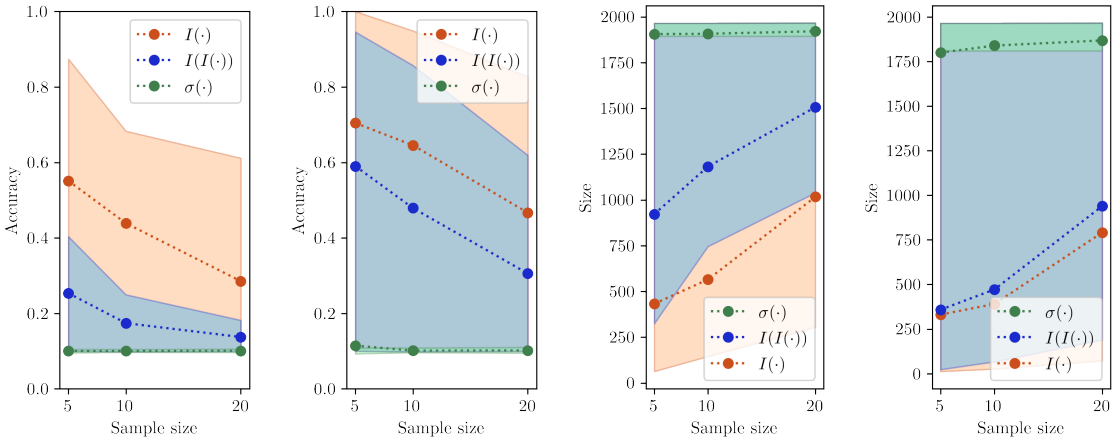
5.3 Real-World Multi-Class Datasets

In this last section of the empirical evaluation, we will apply our modified baseline-approach, as discussed in theorem 31, to two real-world multi-class problems. We do not compare these results to the performance of S^2, as it is only capable of binary prediction. The first one is a small sample of the popular Mnist dataset [LeCun, 1998] of size 2000, each digit appearing 200 times, generated using the Rmnist code [Nielsen, 2017], where R stands for *reduced*. The second one is the Cora citation dataset, where the vertices are articles grouped into 7 different categories. For the Rmnist dataset, we proceeded as with the other benchmark datasets, by forming a similarity graph and producing one weighted and one unweighted version. For the Cora dataset we did not have any similarity measure at hand, but the graph itself. Additionally, we decided to also perform experiments on the Cora graph where we replaced the directed edges with undirected edges, corresponding to dropping the direction of citation. This can be seen as a loss of additional information about the problem, similar to going from weighted to unweighted.

Table 2 sums up the main aspects of the two datasets. Again, as already on the benchmark datasets, we computed a shortest path cover for each of them and checked how many of the shortest paths form a convex partition. Here, the paths have more freedom as we do not need a bipartition, but any partitioning into at most 7, respectively 10, regions, is allowed. Also, we can see the local convexity hypothesis supported here, as the vast majority of shortest paths allows a convex partitioning.

Table 2: *Main characteristics of Rmnist and Cora*

	Rmnist		Cora	
$\lvert V(G)\rvert$	2000		2708	
$\lvert E(G)\rvert$	99000		5429	
No. classes	10		7	
	Unweighted	Weighted	Undirected	Directed
$\lvert\mathcal{S}\rvert$	415	468	747	1391
convex paths	93%	92%	85%	98%



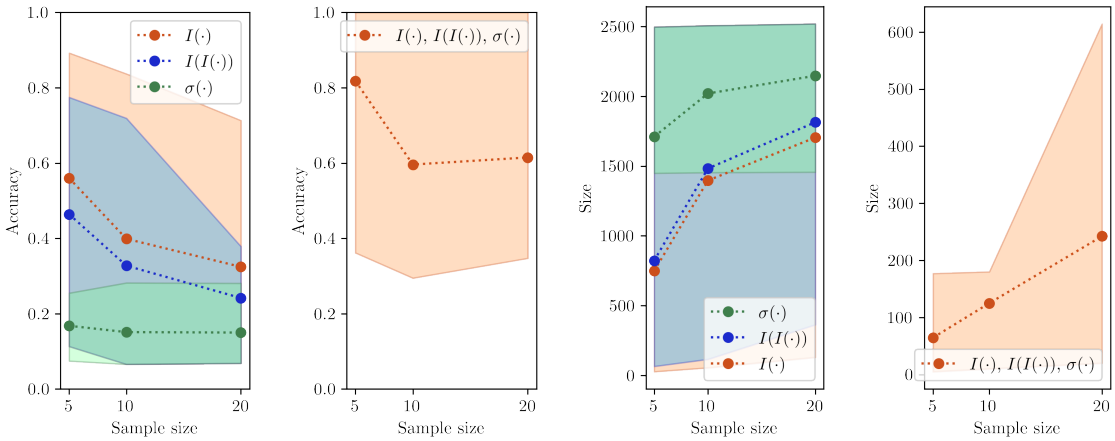
(a) Unweighted accuracy (b) Weighted accuracy (c) Unweighted sizes (d) Weighted sizes

Figure 19: *Rmnist evaluation*

We can additionally see that the weight information does not change the structure of the Rmnist graph too much, whereas the additional direction information in the Cora graph allows on the one hand to compute a shortest path cover that almost completely consists of paths following a convex partition, but on the other hand forces the size of the shortest path cover to almost double compared to the undirected case. This is due to the fact that in the undirected case there are a lot more possibilities to form shortest paths.

To get a better understanding of the possible existing convex structures in the two graphs, we performed some experiments regarding the accuracy of the closed interval and the convex hull. The details are described in the previous section about the benchmark dataset with the only difference of having multiple classes now and not only two.

As before the convex hulls perform as bad as random guessing. The reason is that the convex hull predicts one single label for a region that is almost the whole graph. The results for I and I^2 are promising, especially in the weighted case, achieving in the best case an average accuracy of over 70% and going all the way up to 100% on some samples, while increasing the number of labelled vertices by one or two orders of magnitude. The additional structure given by the weights definitely helps to keep the sizes of I^2 on average small, thus drastically increasing the accuracy by 30-40%.



(a) Undirected accuracy (b) Directed accuracy (c) Undirected sizes (d) Directed sizes

Figure 20: *Cora evaluation*

Let us now inspect the evaluation on the Cora dataset. For the undirected case, the convex hull performs as badly as expected, and I and I^2 yield an accuracy on average of almost 60% down to roughly 30% depending on the sample size. Interestingly, the sizes of I^2 are quite close to those of I, in contrast to most of the previous datasets.

The directed case is the interesting part here. As we had only once before on benchmark dataset 3, all the hull types coincide for all samples, due to the fact that the directed graph does not allow too many different shortest paths, usually only one unique connection between a pair of vertices. The sizes are roughly one order of magnitude smaller than in the undirected case, which increases the accuracy by 20 – 25% compared to the undirected case, having even 100% for some samples.

Query evaluation We evaluate the performance of the baseline algorithm 1, modified as discussed in section 4.2 to the multi-class setting, without any additional fine-tuning.

Table 3: *Query results for Rmnist and Cora*

	Rmnist		Cora	
	unweighted	weighted	undirected	directed
Queries	1721	1716	1762	2230
Accuracy	0.99	0.98	0.95	0.98

Table 3 sums up the main results of the querying. Even though the algorithm queried a large portion of both graphs, which is not the intended use-case of active learning, the accuracy is at a very promising 95 – 99%. We think a more involved algorithm, like the candidate-maximization algorithm 3, and further fine-tuning could lower the number of queries while keeping the accuracy this high.

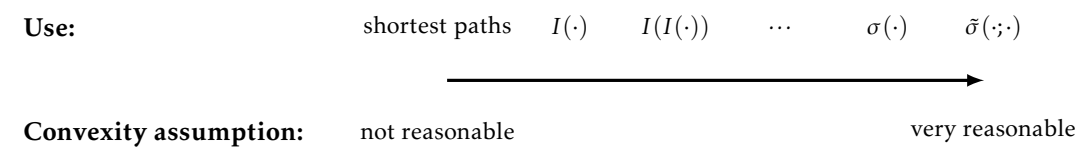


Figure 21: *Given how reasonable the convexity assumption is on your dataset, you can use more and more sophisticated tools. This gives a tradeoff between accuracy and the number of predicted labels*

5.4 Takeaways

We sum up the insights from the empirical evaluation in this small section. As we have seen in every practical situation the graphs are not labelled according to a convex partition. Nevertheless, in almost all cases a certain amount of a local convexity structure was there and could be used by our algorithms. This structure was strengthened by using additional information, like edge weights and information about the direction of the edges, to lower the number of shortest paths not fulfilling the convexity assumption. In almost all cases the graphs were closer to a convex partition and the accuracies were higher on the weighted and directed versions. As a practical rule of thumb, we can use fig. 21 to decide which tool to use to predict labels. The more reasonable the convexity assumption is and the more we trust our data to follow this assumption, the better tools we can use to actively exploit it. If we have only very little trust in the convexity assumptions we could still use single shortest paths or apply the baseline algorithm 1, since as we have seen on many graphs, even if it is globally not convex, single paths tend to follow the assumptions quite often. If we trust the assumption more, we can start computing closed intervals and repeated closed intervals up to the convex hull, to safe a lot of queries, but probably sacrifice some accuracy. In the last end of the spectrum if we are completely convinced of the convexity assumption we can use the convex shadow operator to save even more queries.

6 Conclusion

This thesis introduced the notion of convexity spaces, more precisely geodesic graph convexity spaces, to the world of active learning. Using this notion we were able to develop the two new algorithms, SPC and CM, with theoretical guarantees and could derive bounds on the needed number of queries. We theoretically motivated our approaches, thoroughly evaluated them and showed their competitiveness to the state-of-the-art method S^2 on various synthetic, benchmark and real-world graphs.

On the graph-theoretical level, we developed a new tool called shortest path covers, which give us label-independent tight upper bounds for graphs labelled according to a convex partition. We have developed a logarithmic approximation to find shortest path covers of minimum size. We also discussed the use of hull sets and simplicial vertices to get a lower bound on the query complexity. Using these two tools we developed algorithms for various active-learning settings, including a multi-class one, with different assumptions on the labels and also the possibility of explicitly allowing a certain amount of prediction error.

We turned the rather theoretical baseline approaches to practical ones, still satisfying the guarantees, by using heuristics to perform significantly better in practical situations, as we also showed in various experiments.

On our self-generated synthetic graphs, we have shown that our approaches outperform the state-of-the-art algorithm S^2. We were able to show that on various benchmark graphs we perform on par with S^2, even though on these graphs the convexity assumption, which we used to design our algorithm, does not hold anymore. After in-depth analysis we came to the conclusion that even though globally the convexity assumption is not true, locally, e.g. for a single shortest path, the assumption still holds most of the time across various datasets and argued that this is the key to the good performance of our algorithms.

We end this thesis with some promising future work directions.

6.1 Future Work

Noise robustness It would be interesting to examine our problem with a noisy oracle. This means we can not completely trust the labels we get from the queries anymore, but there will be some small probability of wrong labels. Dasarathy et al. [2015] obtain a noise-robust version by querying each vertex a certain amount of time and then predict the majority of these labels, which could also work in our setting. More involved strategies are work for the future.

Allow small deviations from convexity Similar to the noise case is to instead allow a small number of vertices to break the convexity assumption. Meaning that we start with convex subgraphs, but the true labelling λ might deviate from them on a few vertices. This seems to be a more practical assumption than assuming a complete convex bipartition and it would be interesting to adapt our algorithms to such settings. Auer and Cesa-Bianchi [1998] studied a related problem in a non-active but online model.

Multiple small convex regions Another idea to relax the assumption and bring it closer to real-world datasets is to assume each class to form multiple convex subgraphs, and not only one. For example. local groups of people being interested in one particular topic might form convex regions in a social network, but worldwide the people being interested in this topic presumably not. The graph in fig. 6 also falls in this category, the blue vertices can be seen as two convex subgraphs, implying that such a setting is at least as hard as the case where we assume only the positive class to be convex.

Covering the graph with different structures The example in theorem 24 with the very large shortest path cover needed for the hypercube graph, even though one single path is enough to infer all the labels, is rather disappointing. But the idea of querying a graph by using shortest paths does not have to stop there. Probably different, more complicated, structures can be used to cover the graph and then queried each on its own to yield improved bounds.

One promising step towards such a generalization is to not only take the vertices of the path but also add all vertices whose label can be inferred when knowing the labels of the path. This is not to be confused with the convex hull of the path, as it is not always possible to decide the labels in the hull only by knowing the labels of the path. In particular, we want to only add vertices v whose label can be determined by *any* possible labelling of the path. This means for a path P with vertices $v_1, \dots, v_k$ and a vertex v, we can add v if it is contained in $\sigma(\{v_1, v_j\}) \cup \sigma(\{v_{j+1}, v_k\})$ for all $1 \leq j \leq k$, because only then we can infer its label with every possible labelling using the convex hull of a subpath of same label. So, instead of covering the graph with shortest paths, we could cover it with the paths including these additional vertices, probably resulting in a smaller cover. Indeed, this is exactly the case for the hypercube graph where the labels of all vertices can be inferred knowing the labels of one single path, resulting in a cover of size 1. Further theoretical and algorithmic aspects of this idea pose a promising direction.

Empirical evaluation of the monophonic hull lower bound We mentioned in section 2.5 that the size of a minimum monophonic hull set is a lower bound on the needed number of queries and efficiently computable, but we did not perform any empirical evaluation of the quality of these bounds. In particular, it would be interesting to see by how much the lower bounds exceeds the simple bound based on the number of simplicial vertices we used in the experiments on the synthetic data.

Hardness of the minimum shortest path cover problem The hardness of the minimum shortest path cover problem remains open. For the weighted case we are convinced that we can reduce the

problem of covering points in the plane with the least possible number of lines [Dumitrescu and Jiang, 2015] to the computation of a minimum shortest path cover, resulting in an APX-hardness of our problem. The main challenge lies in the approximation of Euclidean distances, which involves computing square roots. For the unweighted case, there might be a reduction of related covering and partition problems using regular paths or induced paths, as described for example in the work of Manuel [2018].

Embeddings There is ongoing research on various ways to embed graphs into Euclidean space such that for example the distances [Asano et al., 2009] or the nearest neighbours of the vertices [Shaw and Jebara, 2009] are preserved. This is interesting for the machine learning community because if the embeddings preserve certain features of the graphs well enough, we can apply traditional algorithms developed for the Euclidean setting on these embedded graphs. A promising research direction would be to investigate the possibility of embeddings that preserve the induced geodesic convexity of a graph, in the sense that sets of vertices that are convex in the graph should be convex in the Euclidean space and vice versa.

Large-scale graphs Computing the shortest path cover roughly takes $\mathcal{O}(|V|^4)$ time, which is too long for a lot of practical settings with vertex set sizes in the millions. It is important to find ways of scaling, maybe relying on approximations, the computations to such large-scale graphs.

7 Bibliography

D. Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.

D. Artigas, S. Dantas, M. C. Dourado, and J. L. Szwarcfiter. Partitioning a graph into convex sets. *Discrete Mathematics*, 311(17):1968–1977, 2011.

T. Asano, P. Bose, P. Carmi, A. Maheshwari, C. Shu, M. Smid, and S. Wuhrer. A linear-space algorithm for distance preserving graph embedding. *Computational Geometry*, 42(4):289–304, 2009.

P. Auer and N. Cesa-Bianchi. On-line learning with malicious noise and the closure algorithm. *Annals of mathematics and artificial intelligence*, 23(1-2):83–99, 1998.

D. Balcan, V. Colizza, B. Gonçalves, H. Hu, J. J. Ramasco, and A. Vespignani. Multiscale mobility networks and the spatial spreading of infectious diseases. *Proceedings of the National Academy of Sciences*, 106(51):21484–21489, 2009a.

M.-F. Balcan, A. Beygelzimer, and J. Langford. Agnostic active learning. *Journal of Computer and System Sciences*, 75(1):78–89, 2009b.

A. Bendale and T. Boult. Towards open world recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1893–1902, 2015.

A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 19–26, 2001.

P. Bodesheim, A. Freytag, E. Rodner, and J. Denzler. Local novelty detection in multi-class recognition problems. In *2015 IEEE Winter Conference on Applications of Computer Vision*, pages 813–820. IEEE, 2015.

N. Cesa-Bianchi, C. Gentile, F. Vitale, and G. Zappella. Active learning on trees and graphs. In *Conference on Learning Theory*, pages 320–332, 2010.

D. Chakraborty, F. Foucaud, H. Gahlawat, S. K. Ghosh, and B. Roy. Hardness and approximation for the geodetic set problem in some graph classes. *arXiv preprint arXiv:1909.08795*, 2019.

O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

M. Chudnovsky, B. Reed, and P. Seymour. The edge-density for k2, t minors. *Journal of Combinatorial Theory, Series B*, 101(1):18–46, 2011.

V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3): 233–235, 1979.

E. M. M. Coelho, M. C. Dourado, and R. M. Sampaio. Inapproximability results for graph convexity parameters. *Theor. Comput. Sci.*, 600:49–58, 2015.

C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

G. Dasarathy, R. Nowak, and X. Zhu. S2: An efficient graph based active learning algorithm with application to nonparametric classification. In *Conference on Learning Theory*, pages 503–522, 2015.

S. Dasgupta. Coarse sample complexity bounds for active learning. In *Advances in neural information processing systems*, pages 235–242, 2006.

M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 17(8): 1036–1050, 2005.

I. Diakonikolas, T. Gouleakis, and C. Tzamos. Distribution-independent PAC learning of halfspaces with massart noise. In *Advances in Neural Information Processing Systems*, pages 4751–4762, 2019.

E. W. Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1 (1):269–271, 1959.

M. C. Dourado, F. Protti, and J. L. Szwarcfiter. Complexity results related to monophonic convexity. *Discrete Applied Mathematics*, 158(12):1268–1274, 2010.

P. Duchet. Convex sets in graphs, II. minimal path convexity. *Journal of Combinatorial Theory, Series B*, 44(3):307–316, 1988.

A. Dumitrescu and M. Jiang. On the approximability of covering points by lines and related problems. *Comput. Geom.*, 48(9):703–717, 2015.

E. Emamjomeh-Zadeh, D. Kempe, and V. Singhal. Deterministic and probabilistic binary search in graphs. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 519–532, 2016.

M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

T. Gärtner and G. C. Garriga. The cost of learning directed cuts. In *European Conference on Machine Learning*, pages 152–163. Springer, 2007.

A. Guillory and J. A. Bilmes. Label selection on graphs. In *Advances in Neural Information Processing Systems*, pages 691–699, 2009.

R. Hammack, W. Imrich, and S. Klavžar. *Handbook of product graphs*. CRC press, 2011.

A. Holzinger. Interactive machine learning for health informatics: when do we need the human-in-the-loop? *Brain Informatics*, 3(2):119–131, 2016.

T. Horváth, J. Ramon, and S. Wrobel. Frequent subgraph mining in outerplanar graphs. *Data Mining and Knowledge Discovery*, 21(3):472–508, 2010.

M. M. Kanté, T. Marcilon, and R. Sampaio. On the parameterized complexity of the geodesic hull number. *Theoretical Computer Science*, 791:10–27, 2019.

D. Kay and E. W. Womble. Axiomatic convexity theory and relationships between the carathéodory, helly, and radon numbers. *Pacific Journal of Mathematics*, 38(2):471–485, 1971.

B. Korte and J. Vygen. Combinatorial optimization: Theory and algorithms, 2018.

V. A. Kumar, S. Arya, and H. Ramesh. Hardness of set cover with intersection 1. In *International Colloquium on Automata, Languages, and Programming*, pages 624–635. Springer, 2000.

Y. LeCun. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/, 1998.

J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1): 29–123, 2009.

W. Maass and G. Turán. How fast can a threshold gate learn? In *Proceedings of a workshop on Computational learning theory and natural learning systems (vol. 1): constraints and prospects: constraints and prospects*, pages 381–414, 1994.

P. Manuel. Revisiting path-type covering and partitioning problems. *arXiv preprint arXiv:1807.10613*, 2018.

T. Marc and L. Šubelj. Convexity in complex networks. *Network Science*, 6(2):176–203, 2018.

O. Missura and T. Gärtner. Predicting dynamic difficulty. In *Advances in neural information processing systems*, pages 2007–2015, 2011.

M. Nielsen. Rmnist. https://github.com/mnielsen/rmnist, 2017.

A. B. Novikoff. On convergence proofs for perceptrons. Technical report, Stanford Research Institute, Menlo Park, California, 1963.

R. Nowak. Noisy generalized binary search. In *Advances in neural information processing systems*, pages 1366–1374, 2009.

J. Pan and G. J. Chang. Isometric path numbers of graphs. *Discret. Math.*, 306(17):2091–2096, 2006.

T. P. Peixoto. The graph-tool python library. 2020. URL https://graph-tool.skewed.de/.

I. M. Pelayo. *Geodesic convexity in graphs*. Springer, 2013.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

J. Sabater and C. Sierra. Reputation and social network analysis in multi-agent systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: Part 1*, pages 475–482, 2002.

F. Seiffarth, T. Horváth, and S. Wrobel. Maximal closed set and half-space separations in finite closure systems. *CoRR*, abs/2001.04417, 2020.

R. A. Servedio. On pac learning using winnow, perceptron, and a perceptron-like algorithm. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 296–307, 1999.

B. Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

B. Settles. From theories to queries: Active learning in practice. In *Active Learning and Experimental Design workshop in conjunction with AISTATS 2010*, pages 1–18, 2011.

B. Shaw and T. Jebara. Structure preserving embedding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 937–944, 2009.

J. L. Sobrinho. Algebra and algorithms for qos path computation and hop-by-hop routing in the internet. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 2, pages 727–735 vol.2, 2001.

The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.0)*, 2020. https://www.sagemath.org.

M. Thiessen. Active learning on convex graphs. https://github.com/maksim96/ActiveLearningOnConvexGraphs, 2020.

M. L. van De Vel. *Theory of convex structures*. Elsevier, 1993.

F. Wilcoxon. Individual comparisons by ranking methods. *Biometric Bull*, 1:80–83, 1945.

D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *Advances in neural information processing systems*, pages 321–328, 2004.

X. Zhu, J. Lafferty, and R. Rosenfeld. *Semi-supervised learning with graphs*. PhD thesis, Carnegie Mellon University, language technologies institute, 2005.

Statement of Authorship

I hereby confirm that the work presented in this master thesis has been performed and interpreted solely by myself except where explicitly identified to the contrary. I declare that I have used no other sources and aids other than those indicated. This work has not been submitted elsewhere in any other form for the fulfilment of any other degree or qualification.

Bonn, May 28, 2020

Maximilian Thiessen