# Taxonomy of Evasion Problems in Graphs

The name “Evasion” already reveals somehow a bias, as we will see that many other problems can be seen from this same perspective, besides that of Evading Community Detection, which is the one in the paper by [1] that motivated these thoughts. However, we will focus mainly on Evading Community Detection, and for the time being we will keep that title.

The general idea is that of trying to change, to fool, to spam, or to evade the results of a (Community Detection, CD) algorithm by strategically altering the structural properties of the input graph. Similar, but not identical approaches are taken in the so-called counterfactual reasoning [2].
But also in older papers that refer to similar ideas under the names of

- Adversarial attacks
- Robustness
- Sensitivity Analysis

One important thing to do here would be to analyze and clarify differences and similarities with those references.

We have found also some relation with the subjects:

- Local search. There are many similarities. In LS we define a neighborhood and apply local changes that improve the value of the objective function. Here we do something similar, we apply local changes according to some rules and following the gradient descent.
- Algorithms with predictions (algorithms that use predictions from machine learning applied to the input to circumvent worst-case analysis, Mitzenmacher)
- Discovering faster matrix multiplication algorithms with reinforcement learning (where they use ML techniques to derive the optimal parameters of a known algorithm, Fawzi). Summary of this paper: It is a deep reinforcement learning approach based on AlphaZero1 for discovering “efficient and provably correct algorithms for the multiplication of arbitrary matrices.” In practice, what they do is to train an agent AlphaTensor to play a game with the objective of finding the best decomposition of different size matrices to be used by Strassen’s algorithm. For example, they found a set of 47 multiplications that is enough for multiplying two 4x4 matrices (instead of the 49 multiplications required by

Strassens classic algorithm (7 in the first level and 7 in the second level). The connection would be: could their approach be used to make the agent play a game with the objective of hiding the community of a node, in the same way as [1] but with their framework?

- Community shifting users (analysis of the characteristics of users (nodes) that change community over time). They probably didn't use one of these algorithms, they just changed, the paper explains their characterization. Albanese & Feuerstein).

References

[1] Bernini, Andrea, Fabrizio Silvestri, and Gabriele Tolomei. "Evading Community Detection via Counterfactual Neighborhood Search." *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2024.

[2] Ana Lucic, Maartje A. ter Hoeve, Gabriele Tolomei, Maarten de Rijke, and Fabrizio Silvestri. 2022. CF-GNNExplainer: Counterfactual Explanations for Graph Neural Networks. In International Conference on Artificial Intelligence and Statistics, AISTATS 2022, 28-30 March 2022, Virtual Event (Proceedings of Machine Learning Research, Vol. 151), Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera (Eds.). PMLR, 4499–4511. https://proceedings.mlr.press/v151/lucic22a.html).

Taxonomy

We can define a bunch of dimensions to study the framework, potentially each combination of values in these dimensions gives rise to a different interesting problem (probably some of the problems coincide, some have no sense, some may be trivial), but I think this is useful for the full comprehension of the matter.

I first give a list of the dimensions I have thought of, then explain and propose values for each of them

Dimensions:

- Problem/property considered
- Characteristic of the input (features)
- Modifications allowed
- Metrics used

- Knowledge of the algorithm
- Complexity of the problem

Dimensions & Values:

- Problem/property considered (as I said before, any graph problem could, a priori, be subject of this approach)
    - Hide (leave) your community
    - Change the number of communities
    - Want to belong to the same community as x \in V
    - Want to belong to the same community as C \included V
    - Don’t want to belong to the same community as x \in V
    - Don’t want to belong to the same community as C \included V
    - Break a community in 2 or more parts
    - Join communities C1 and C2
    - Hide your (some) community inside a bigger one
    - Be connected to x \in V
    - Not be connected to x \in V
    - Reduce/increase the shortest path from x to y
    - Reduce/increase x’s betweeness centrality
    - Reduce/increase x’s Page Rank
    - Reduce/increase the diameter of the graph
    - Reduce/increase a maximum flow
    - ANY GRAPH problem/property
    - Non graphs but geometrical

- Characteristic of the input (features in the graph)
    - No features
    - Nodes have features
    - Edges have features
    - Nodes and Edges have features
    - Particular case of directed and undirected graphs

- Modifications allowed (what are the “legal moves” that can be used to change the output)

    - Add
        - Vertices
        - Edges (*)
        - Both
    - Remove
        - Vertices
        - Edges
        - Both
    - Add & remove
        - Vertices
        - Edges
        - Both
    - Modify features
        - Any
        - Increase
        - Decrease
        - ???
    - Edge's features, nodes features, both
    - Sets of modifications in one move (analogy with Local Search, how we define the neighborhood).

(*) The version of the problem in [1] with only insertion of edges has sense, sometimes the user cannot undo things

- Metrics used (Different metrics used to measure the distance between the instances of the input and of the output, the objective function to minimize could be a combination of both)
    - Input
        - # added vertices
        - # added edges
        - # deleted vertices
        - # deleted edges
        - #features changed
        - \sum(\delta(features))
        - Other?
    - Output
        - #communities added
        - #of nodes changed community
        - Difference between the communities before/after

        - Deception score (reachability, spreadness and hiding, as in the paper)
        - Ad hoc for any problem

- Knowledge of the algorithm (do we know what algorithm/model is being used by the "owner" of the graph)
    - Yes
    - No
    - Know the algorithm but no the parameters
    - Algorithm can change from time to time?

- Characteristic of the model: This is crucial. The algorithm may be fixed and known, but the output/class of the rest of the points may change or not when "our" point is modified. It seems that if we fix the classification of the rest of the points the problem may be much easier.

- Complexity of the problem (what is the intrinsic complexity of finding the optimal change that could be introduced to the input to alter the output? Does it matter?)
    - Original problem is in P
    - Original problem not known to be in P

---

Application in a geometrical setting.

Idea: we could try the approach in [1] in a geometrical setting, where the clustering problem is interesting enough. There are many versions that are NP-hard even in 2 dimensions. The definition of the problem would be similar to that in [1] but instead of a graph on n nodes and adding or removing edges, we have n points in a d-dimensional space, and the specified point u could move some distance in some direction, to leave the current cluster. The goal is to minimize the distance moved.

A classical algorithm would choose some other point (maybe the nearest that is part of another cluster?) or orthogonal to the line that separates u's cluster from another cluster. How much to move? An intuitive heuristic would be to start with a skip of length 1 unit, then 2, then 4, until it changes cluster. Or a probabilistic algorithm that chooses the distance with some probability…(remember Barabasi and small world models?).

As for the DRL algorithm, one nice situation is that in this continuous setting, the #states is \infty, but of course not all the directions have sense, and we could reduce its cardinality in some intelligent way. Examples: possible directions could be the n-1 other points v, and the distances moved could be 1,2,4,8….dist(u,v), or similar exponentially increasing (may be dynamically changing? After moving some distance, the new distance is smaller, so some possibilities are not allowed anymore). Moreover, it could be interesting to add some stochastic ingredient to choose how far we want to move in one move).

Then, we should define a loss function that would be rather similar to the one in the paper. Maybe because of the nature of the problem the only important thing is that u changes cluster (as the other clusters should not change)?

Associated problem: Robust Clustering – Provide a clustering such that the distance that a node (all the nodes? Some nodes? A particular node?) must move to change cluster is maximized.

---

What about the (weighted) graph problem associated to this and to [1]? I.e. the edges that can be added (or eliminated) have weights. So, the choice is not only to add or remove edges but to choose the weights….It seems nice! The budget would be, not a number of edges added or removed but, for example, the sum of the weights added or eliminated, or only added, or…whatever.

---

Differences with Function/Active Learning

Luca proposed a Problem1 that consists in designing an algorithm that eventually produces a z that maximizes certain function F known by an oracle, after submitting a sequence of inputs and receiving the value of F for those inputs.

Underlying Problem 1 is the problem of learning a predictor of the behaviour of an oracle by submitting a suitable sequence of inputs and observing the results. This problem is called **function learning (FL)**, and (binary) classification is the special case where F's codomain is {0,1}.

As for **active learning (AL)**, it is connected with FL but they are not exactly the same. In AL, a model tries to discover an underlying function F. Based on a set of labeled examples, it starts with a prediction F', and is allowed to query an oracle about F(x) for some other examples (and hence observe the results) with the goal of improving the prediction for the other cases.

So, in both cases, when/if the model learns the underlying F, we are done. But for problem 1 above, that is not necessarily the case: from a computability/complexity point of view, having learned F does not mean that we know it's maximum.

As for the counterfactual/evading problem, the differences are bigger: there is a model F' that tries to predict a certain F, but we do not really care about F. F' is more important than F, because we want to "cheat" F' (not the underlying F). If we knew F it would be (almost) useless. Even if we knew F' we would still have to look for a z such that F(z) has some property.

From this point of view, it seems to me that problem 1 is somehow in the middle: even if there is an F, we want to maximize F'(z), not F(z)