Changes repo, repos.

תיעוד מלא לקוד ולכל אחד מהקבצים

לעשות עריכה לקבצי האופטימיזציה

Our goal

Model description

The concept of MIP-Net is explained in detail in Dr. Amir research paper, which is the basis to this project, as mentioned before.  
In this project, we used a (slightly modified) version of the graph,

Differentiation for original MIP-graph description

**Similarity metric**:

We’ve identified a potential problem with the adamic-Adar-Proximity in the case where the two nodes (s,t) are connected by a third node (c) s.t. total weight of the edges (c,s),(t,c) is 1.0. in that case, the formula should return inf (0 in the denominator).

We decided to use the “simple” Proximity metric instead.

**Handling of deleted objects:**

Objects which were deleted in the code are kept in the graph until the weight of all connected edges is 0. This allow recovery in cases were one commit deletes a file and another quickly re-instate it and maintaining the “information” stored in the object (derived from past activity) about connections between other nodes.

The only pitfall is in the case where a file is deleted, and shortly after, a new file is created with the same name. we assume that this case is rare, and either way, identical name can imply similar functionality so the transfer of the “history” between the files isn’t completely useless.

**Incrementing edge weight by action weight:**

Previously always increment by 1, resulting in just negating the decay (in the case where decay=1). Right now, all action weights are 1, so the change is meaningless, but we put it in place in case the action weight differ.

**Ranking all objects**

The original model is trying to highlight the most important (according to a dynamic definition) changes to a given user. In this version, we’re trying to gage what he’s likely to change. we assume that whether a file was changed by others isn’t conclusive on to itself to the question of whether a user will change is, so we rank all the object’s in the graph (according to the DOI metric) to predict the next changes.

**Centrality**

It the research, it was meant to be an a-priori component. In the implementation we worked from, it was an “information flow” matric for determining the nodes importance in the graph. Because of compatibly issues, we defined the centrality of a node to be its degree (number of connected edges).

Project modules (in a nutshell)

Optimizing and evaluating using Github data

First, it’s worth mentioning that the optimization part is crucial. Since the weighted parameters (centrality, proximity and change extent) aren’t normalized, we needed for the ratio between the weights to work as a normalization factor, as well as a measurement for the parameter’s importance.

**Evaluation data**

We decided to use existing GitHub repositories as a bench-mark, using the model to predict the files that a given user will modify in his\her next session.

We downloaded a bunch of Github repos (full description can be found in the project files) and filtered out all the commits where the user didn’t modify existing files. We also had to filter Out the repos where the number of files exceed a curtain threshold (arbitrarily: 2000) since they proved too time-consuming.

At first, we tried predicting on all the data in the following method:

1. Build a base-line graph comprised of the first 80% of the commits in the repo.
2. For the remaining 20%, at each iteration, predict the most likely objects to change, compute the score, and update the graph with the real changes.

הבנו שזה לא משנה ואפשר להכניס את כל הקבצים ברצף.

ניסינו לעשות אופטימיזציה למשהו מסוים: רק ליוזר ספציפי.

**Evaluation matric**

We looked at different ways of evaluating. At first, we tried remaining close to the evaluations described in the paper and maintaining a notion of “top-3”, “top-5” etc. quickly we realized that it’s not applicable to this case, first because of the data (some - if not most – of the commits modified only one or two files making it kind of a binary prediction) and also because of the lack of the internal order between the modified files.

Instead, we decided to try and maximize the proportional sum of the DOI of the changed objects compared to the other objects in the graph. Meaning, choosing the weights s.t.

The idea is, we try to find the weights where the changes objects get a “larger” proportion of the total DOI, then individual changed object will be more likely to score high DOI compare to non-changed objects.

We also tried to find ways to expand the metric beyond the “next commit” harsh threshold, but couldn’t a way to do it without compromising the consistency of the graph and prediction (since commits of the same user aren’t always aggregated together, this could’ve caused silly situations where we combine two commits that were omitted a month apart because there wasn’t a commit of another user in the middle, while separating two others from the same day).

Results

**“Stupid” classifiers**

**What else we tried**

we tried maybe inferring other conclusions from the graph “shape” and the ways the graph changes over time.

We couldn’t discover any new information. Other than the obvious (at the beginning, the graph grows until all the files are committed and from this point on the changes are just in the connections\weights. Dah), the graphs looked to us completely descriptive. The changes were according to the pre-determined formulas, and nothing in the structure looked like it contains any extra information.

Why it didn’t work

**Not a good proxy**

Our main conclusion from the results is that this proxy undermines the entire premise of the original MIP-Net model, since “interest” doesn’t equal “desire to modify”. For example, let’s assume that our repo consists of a main file and many others utility files (a structure that resembles this of most actual repos). And let’s assume that each team member is working on one of the sub-modules (again, very common in practice), a change in the main file is very relevant to every user, but our model isn’t built to recognize that fact.

This alone isn’t detrimental onto itself. It just means that we are trying to apply the MIP-Net model for a different purpose, instead of measuring “imprtance of changes” we’re trying to predict “what a person will change”. It’s not clear if the same model will work for this different task, and we couldn’t even come up with a hypothesis because of the second problem:

**Really bad idea for a data set**

In fact, realizing that after examining the result, made any attempt to better our model feels futile, since we strongly believe that measuring it using the “next file to change” proxy with the Github data will result in just noise.

Problems we hypothesize will arose from the Github data:

1. The life cycle of a repo is hardly one-piece. After a while, a repo is “stabilized”. Meaning, the product\library is done and so is most of the “heavy leafting”. From that moment on, changes are made based on bugs and
2. There is one or more person that control the commits omitted into the “main” branch.
3. Roles definitions in the group.
4. Open-source projects are notoriously problematic. Long tail…
5. Linear order doesn’t really represent “workflow order” – the most prevalent problem.  
   the decay is too harsh because commits can be minimal or very invacive.  
   Couple commits in a row – huge problem. Even trying to fix with aggregating isn’t enough because what if one commit today and one tomorrow?  
   working on something for a while before committing.

Commit and “pull” aren’t the same.

Not clear how much does a person “care” about other changes.

We tried isolating some of the differences and looking at a specific case. For example, we looked at a repos with “long tails” in comparison to looking at the most dominant users in a given repo.

In short, nothing is “generalizable”, and this problem can’t be solved using a one-size-fit-all set of weights.

From all of these and other theories, we strongly believe that using Github repos as data points require deeper understanding of their structure and policies, and that different “phases” and differences between users’ roles and actions are large enough to not be “smoothed” even in a large enough sample.

**Important point of emphasis**

זה לא שהשיטה של מיפס לא מתאימה למשימה של עבודה משותפת על פרויקט גדול, אלא שהצורה שבה אנחנו ניסינו ליישם אותה ובייחוד השימוש בספריות גיטהאב לצורך שיערוך לא מתאימים בכלל למודל המתואר.

אנחנו סבורים שהמודל יתאים יותר למבחן עם צוות ממוקד באמצעות ניסוי ישיר. בצורה כזו ניתן לשלוט בכל הבעיות שתוארו למעלה, לוודא שמקבלים את שטף העבודה בסדר ליניארי, לתשאל בנוגע ל"רמת ענין" ולא להשתמש בפרוקסי, להכניס מידע א-פריורי בנוגע למטרת הקוד, תפקידי אנשי הצוות וכדו'.

Further exploration

**Problem solving**

As mentioned above, further advancement is dependent on getting a reliable data for measuring performance. But even with a close study with a controlled group, some problems still stand. Mainly, defining what is an “important” change to flag, how to measure success or failure, and more.

**Trying it with namespaces (functions)**

The objects in our project are defined to be documents (files) in each project. The same principles can be applied in a more granular approach, by looking at each function\class as an object.

We didn’t really put much of an effort in this route, after early tries made it look like a quite taxing task (from the perspective of parsing and structuring the Github data), and we didn’t saw much of an added value since we didn’t used the content of the objects anyway.

The *CsrCode* class in the CSR.py file can be used as a base class for this exercise. All other modules should generalize without changes.

**Using the code-structure for extra information**

How many other functions call or being called from the changes function, etc.

**Diving into the code**

Analyzing the code changes could uncover more information. Particularly, devising a policy to determine each action weight according to the type and size of the changes in the commit. Examples of parameters for such policy: the number of lines in the “diff” string of the changed lines, the file where the changes were made, the “type” of the change (renaming a variable vs added input parameters etc.).

**Centrality could contain actual a-priory information**

We think that degree-based centrality should have a pretty week significance to determine DOI, since in a big enough project, the graph is so large that the values of this component are insignificant. Instead, it’s possible to assign the nodes real a-priori information, either from the code-structure as discussed before or from other known information and features.

**The time-stamp could tell us a lot**

One of the problems we mentioned above was the non-linearity of the Github data. It’s possible to remove some of the haze by looking at the time-stamp of the commit and changing the model’s behavior accordingly. One of the things we tried was adding a “time-based decay” to the model, but that’s not enough. It’s easy to think of many ways where the time passed between iterations is as (or even more) important than other parameters.

**Change Extent**:

Currently defined to be : . This only considers the frequency of change, not the amount of it. Maybe this metric should also incorporate the weights of the changes made to the object since the last visit of the user.

What else

**Machine-learning approach**

Although we like to think of this model as pretty “smart”, the described task looks perfect for trying a machine-learning approach. Our work on the infrastructure for the retrieval and cleaning of the data, combined with some hand-crafted features, some of which already embedded in the model, make it very easy to try and predict the next code changes using classic machine learning algorithms. Results and conclusions from these experiments could also be used to update this version of the model. For example: the weights calculated by a logistic regression could help determine the weights for the parameters in the Mip-model, etc.

**Re-purposing the model**

Our model is fairly general and can easily be applied to Wikipedia edits (for example). It’ll be interesting to see how it does in other domains. This kind of exercise could also shed light on the performance on the git-hub task and help uncover if there is some problem with the model itself, the evaluation method, or is it just bad fit for this domain.

**Can I recommend something else?**

Although the original paper described a model for improving collaborative team-work, our mission seems more like a typical recommendations problem. We used the collaborative nature as a tool for discovering connections, but eventually used those connections and predicted “what this person will find the most interesting”. This is a well-studied problem and possible solutions can be found in all areas of AI. We think it could be beneficial to infuse between the two problem. Meaning, using recommendations algorithms to predict code changes (as mention above), and using the MIP approach on classic recommendations data-sets (the most obvious application that come to mind is something like the Facebook newsfeed, where stories are objects and profiles are the users).

What we’ve learned

Well, as a research project, this can be described as a complete failure. Not just we didn’t solve the problem we set out to solve, the model we created looks to be pretty much useless.

As a learning exercise, we took in a lot. Diving into the research paper gave us new perspective onto itself problem solving in the AI domain (not everything is deep learning, apparently). Using graphs to model interactions in a dynamic environment and quantifying and measuring something somewhat “amorphic” as “Degree of interest” are examples for things we didn’t encounter until this project.

From the discouraging results we took a lesson to be more prepared and work completely different in future projects. We rushed to the code and spent too much time on the “production” problems such as generating data, optimizing resources and so on, that we barely took the time to think of the methodology itself. Looking back, we could have predicted at least some the problems with using Github repos. We should’ve spent more time on properly defining what we are trying to achieve and assessing whether this model combined with the available resources is best suited for this goal.