* „Variable sink functionality“
* DAG structure is being abused bc I treat self loops as a special case
* The self loop should be a separate parameter, not a self loop, like a variable „sink amount“
* => so arrows in the graph should only be delegations
* In a way, each node has two nodes, delegation node and a sink node
* So self loops are kind of „no-ops“
* Global iterations instead of the mixture between graph traversal and optimisation, so we traverse the entire graph once, and then again, and then again, until the movement is below a threshold (non-optimised algorithm of my version)
* Does it make sense to vote a bit yourself and a bit delegating? We could discuss this (confidence in your own vote, camaraderie e.g.)
* Try LP without my little sink functionality
* Objective: max weights while keeping the bounds (inflow, outflow constraint)
* Variables: weight per node (w\_i)
* Try simple graphs first
* Per iteration, inject a new vote per each node, and propagate each other votes in the system
* At the k-th iteration, votes k had no time to propagate, k-1 has 1 step to propagate, etc. If you do this a lot, the sinks collect p\_i per node, at the limit, p\_i will be the amount of votes you put into the system, then you need to divide those or normalise them top get the „real voting power“
* Explain diagrammatically and textually this LP thing
* Maybe start on the document

25.02.2025

* We can assume that the algorithm has at least one sink (or each clique has at least one sink)
* Check the second algorithm to find a case where the first one doesn’t break, but the second does
* Difference between first and second algorithm:
  + First one can prove xxx using the harmonic series
  + If you just do delegations of 100%, the first one might not oscillate around which the second one does
* Regarding the P\_G question
  + P\_G should not vote, that is ok, since he is not a voting sink
  + All outflows must add up to one, that is something we can say
* For linear programming:
  + Compute the standing pool of voting power, not the final power
  + E.g. compute the maximum power that fit the delegation constraints,
* Formulae:
  + If you carefully keep the iterations separate, so the values are all dependent on all past iterations, then the formulae should be similar to the LP formulae that we are going to solve

For next meeting:

* LP works, the way that I found, but I also noticed its more efficient if you just solve a system of linear equations. Is this like what you imagined? It’s basically unravelling the simulation and then solving the equations
* I have not yet (?) proven or shown that the no source effect one is flawed. Could you please specify why you think it would not work?

Meeting 11.03.2025:

* We want the limit, while here its just solving for the nth iterations
* Ford proposed a delegation which might not work without the “source effect”, there only C should vote in the steady state
* We want to avoid an oscillation between 0 and something, 0 and something

Meeting 20.03.2025:

* The algorithm works. Yay
* Next steps:
  + Write up the algorithm
  + Benchmarking
  + Try some synthetic and “fun” graphs
  + E.g. create graphs in Python automatically (there are libraries for this)
  + We need directed not-acyclic graphs that might resemble a “real” delegation
  + Maybe we can even scrape or find a dataset (e.g. Twitter follow relations)
    - The IMC often scrape these kinds of measurements from crawling the internet

For next meeting (01.04.2025):

* What have I done:
  + Some benchmarking on custom graphs
  + Set up the visualization of graphs
  + Done some reading
    - Fractional Delegation
    - https://markus-utke.github.io/files/Anonymous\_and\_Copy-Robust\_Delegations\_for\_Liquid\_Democracy.pdf
* Timeline update (end by June)
* What to do with the benchmarks?
  + What to do with the benchmarks
* Currently, I don’t really have a sophisticated way to prove that the results are correct. Should I maybe create one?
* I will start writing ig
* Title ideas (I want to send a preliminary one to TUM):
  + Resolving delegations using Linear Programing

Notes before the meeting:

* Instead of building the graph iteratively, start off with all nodes already there, and choose a delegate from these nodes, this way its way more realistic imo. We can also make their popularity drop off, so choosing node 1 is super likely, choosing node 2 is less likely, etc., to replicate super voters
* Have a look at Liquidfeedbacks algorithm, and maybe compare it with mine?

Meetig notes:

* Use this <https://networkx.org/documentation/stable/reference/generators.html> to generate graphs
  + [**barabasi\_albert\_graph**](https://networkx.org/documentation/stable/reference/generated/networkx.generators.random_graphs.barabasi_albert_graph.html#networkx.generators.random_graphs.barabasi_albert_graph) maybe
  + Keyword: Preferential attachment
* Benchmark: Compare them: When is what algorithm faster, scalability
  + Find maybe certain cases which LP Solvers are bad at solving, or the opposite, where iterative one is bad but LP good
  + Sparsity vs density of the graph, what does that change…
    - almost no one delegates and the chains are short vs everyone delegates very far
    - bc in practice, there might not be a lot of delegations in a real system
  + Don’t shy away from unrealistic scenarios
* Try to explain the algorithm (intuitively why it is correct), and then maybe prove correctness, but that is only optional
* Deadline in June is ok
* My own notes:

Notes before the meeting:

* Benchmarks are going quite bad. LP is generally slower than doing Iteratively, except when you need to iterate very small amounts. Also, I am afraid that LP is slower than solving a system of linear equations.
* <https://arxiv.org/pdf/1412.4039> has done the same thing we have but using systems of linear equations. I haven’t tried it yet, but that is probably faster than what we are doing. Only benefit I can see atm is that we have the sink node constraint, which ensures no voting power is lost, but I think you can prove this and thus no longer need the constraint…
* This paper doesn’t prove anything though… although this one does, they prove something else, but mention that their proof also proves what the paper above has done
  + <https://www.researchgate.net/publication/363920340_A_Voting_Power_Measure_for_Liquid_Democracy_with_Multiple_Delegation>

Meeting Notes 10.04.2025

* Cite the other two papers definitely
* Benchmarks are still a good topic
  + Sparse vs dense graphs
  + Big circle
  + Separate between “set-up costs” and “solving costs” in the LD thingy
  + Use the same precision for the LD solver and the iterative thing
  + Instead of generating a perfect graph, build a postprocessing algorithm that that takes any graphs and cleans them, e.g.
    - Normalize edges
    - Remove edges to create sinks
    - Add weights to unweighted edges (such as 1/k)
* TODO check the second paper more closely
* Show the eight or whaever properties an LD system should have (<https://liquid-democracy-journal.org/issue/3/The_Liquid_Democracy_Journal-Issue003-01-Preferential_Delegation_and_the_Problem_of_Negative_Voting_Weight.html>)
* Maybe add the fractional self-delegation features to my algorithm, but first check the second paper (https://www.researchgate.net/publication/363920340\_A\_Voting\_Power\_Measure\_for\_Liquid\_Democracy\_with\_Multiple\_Delegation) what algorithm they use
* Collect statistics on how open perfect loops appear in a graph (maybe we can do something where we also check how often they appear in graphs where we only allow non-fractional delegation

Notes for before next meeting:

* The setup of the LP takes quite little time, runtime changed from 17.4s to 17.3s
* The second paper does no benchmarking, and their ld model is the one that I implemented in the beginning. They prove their model, and use some game theory too, to show that using their implementation of LD, there exists a pure strategy Nash equilibrium. Is that something that we might want to do?
* I did the graph transformer

Things I want to do:

* Check how the performance changes if a use an LE solver
* Try out the graph transformer, “torture testing”
* Try to find graphs (however synthetic) that work better with LP than iterating
* Algorithmic runtime, computational feasibility
* Document that a high precision has very poor performance for the iterative approach (cutoff= 0.0000001 or something like that)

Meeting:

* Find out how long the setup changes
* Game theory might be interesting, but out of the scope of the project. Nevertheless, it might be interesting and useful to understand their results, and maybe building on them