Heuristic Algorithms using Tree Decompositions

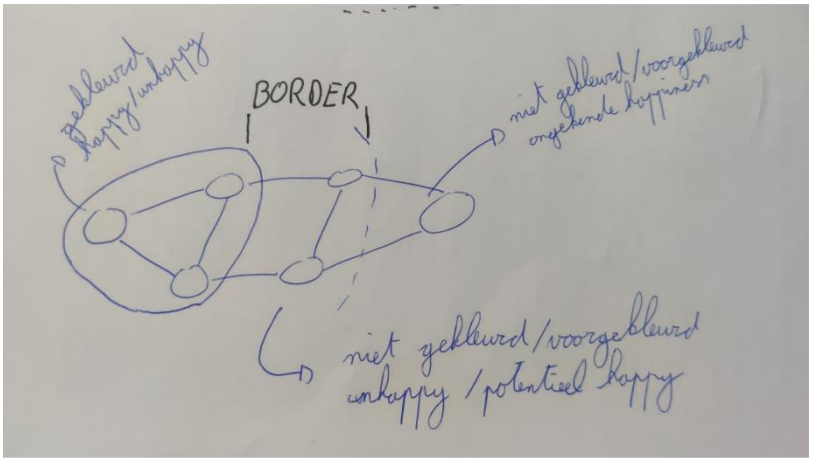
Report 15 31/03/2022

Ondertussen ben ik aan de ene kant redelijk ver met de implementatie van het nieuwe algoritme en aan de andere kant is er wel nog wat werk.

Ik zit al redelijk ver omdat ik al leaf nodes, introduce nodes en forget nodes volledig heb geschreven en toch redelijk overtuigd ben dat er geen optimalisaties meer mogelijk zijn (of toch geen die enorm veel verschil maken). Ik had nog extra dingen geprobeerd om verder te optimaliseren maar hier kon ik geen voordeel uit halen.

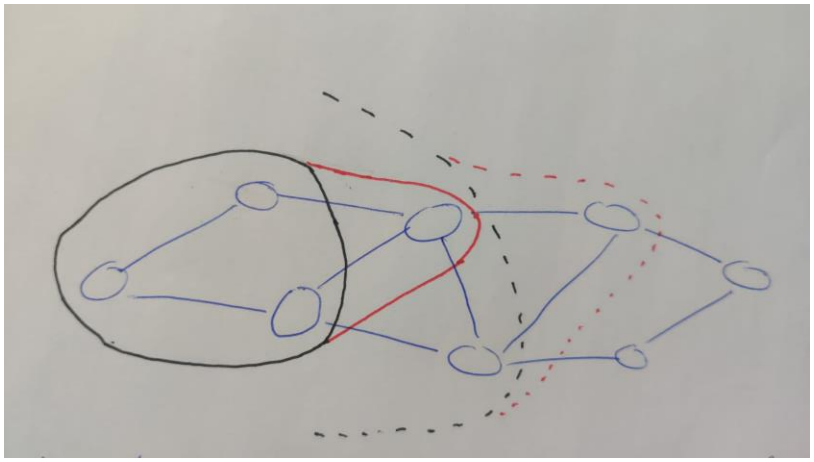
Aan de andere kant is er toch nog wel wat werk aangezien ik nog de join node code moet schrijven. Ik heb al een basic versie maar dit is volgens mij te beperkt. Uit vorige experimenten bleek dat de vertices in de bag één voor één te kleuren de beste resultaten gaf en dit ben ik ondertussen aan het implementeren. Deze code is wel iets complexer aangezien ik nu ook expliciet de happiness van de vertices bij houd.

Hieronder heb ik een tekening gemaakt van hoe het algoritme werkt. In de thesis zal ik een gelijkaardige tekening toevoegen, maar dan via tikz in plaats van gewoon uittekenen. Ik verdeel de graaf in 3 delen. Eerst heb je het deel dat gekleurd is (in de volle cirkel). Bij een specifieke node in de tree decomposition zijn dit de vertices in de bag van die node alsook de vertices in de bag van de descendants van die node. Vervolgens in het midden definieer ik een *border*. Dit zijn de buren van de vertices uit het eerste deel. Deze hebben nog geen kleur maar kunnen wel voorgekleurd zijn. Aan deze vertices werd nog geen concrete happiness aangegeven, maar je weet wel al of ze nog happy kunnen worden (en dus potentieel happy zijn) of al unhappy zijn aangezien ze aangrenzend zijn aan het eerste deel. Het laatste deel zijn dan vertices waarvan je nog geen happiness kunt bepalen.

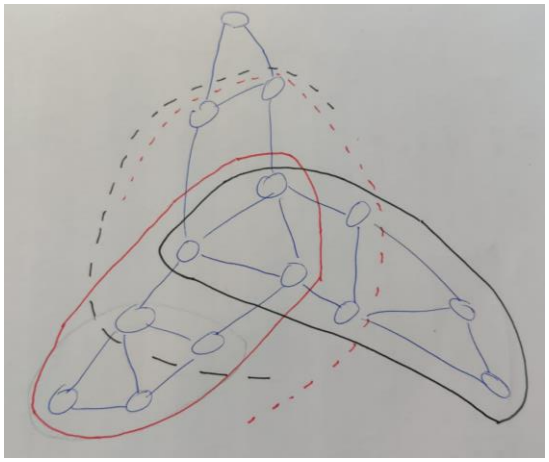


Aan de ene kant zorgt deze border er voor dat het algoritme complexer is (want je hebt meer mogelijke combinaties door de verschillende types van buren) maar ik kan de informatie in de border wel exploiten om het algoritme efficiënter te maken. Ondertussen heb ik dus al 3 van de nodes zo geoptimaliseerd en het algoritme is een stuk sneller (+-11 seconden nu voor een instantie dat vroeger +- 55 seconden duurde).

Om nog wat extra informatie te even over het algoritme heb ik nog een foto toegevoegd over introduce nodes en join nodes. Eerst de introduce node. Deze is an sich relatief eenvoudig, het deel dat gekleurd is moet groter worden en dan ook de border (dit groeien is aangeduid met rood in de foto). De implementatie zelf is toch vrij complex omdat je rekening moet houden met potentieel happy neighbours en unhappy neighbours en deze kunnen dan nog voorgekleurd zijn of niet. En de happiness van alle vertices in de border moeten dan nog correct gezet worden. Ik heb redelijk wat tijd gestoken in deze code, maar het werkt nu en zorgde voor het grootste deel van de tijdswinst eerder vermeld.



Nu ben ik nog bezig met de join node. Hieronder heb ik ook een foto toegevoegd, waarbij het ene kind de kleuring met rood geeft en het andere kind de kleuring met zwart. De bag bevat de 3 vertices in het midden, en de uiteindelijke kleuring moet alle vertices kleuren die met een volle lijn omcirkeld zijn door zwart of rood. De border is dan de twee vertices net boven de bag, en de ene vertex helemaal bovenaan zit bij het derde deel van voordien. In het exact algoritme zoek je naar een kleuring van de bag die in beide kinderen exact voor komt. Dit deel is ook al geïmplementeerd in mijn algoritme en werkt al. Maar aangezien ik slechts een beperkt aantal kleuringen bij houd is het niet zeker dat zo'n match bestaat. Ik zou kunnen zeggen dat er in dat geval geen kleuring kan gevonden worden, maar het is beter om dan toch twee niet-matching kleuringen te combineren. Dit is het deel waar ik momenteel mee bezig ben. Doordat er nu echter twee kleuringen zijn die overlappen en dus verschillende happiness geven aan de vertices en verschillende kleuren zijn er veel verschillende combinaties mogelijk, waardoor dit iets langer duurt.



Ik heb verder ook een check toegevoegd die aangeeft of het algoritme met zekerheid een exacte oplossing gevonden heeft of niet. Dit gebeurt wanneer het aantal partiele kleuringen nooit gelijk is aan de *width* parameter. Uit theoretisch perspectief kunnen we ook een bovengrens berekenen op deze *width* opdat het algoritme zeker een optimale oplossing kan vinden, namelijk het maximaal aantal mogelijke kleuringen in een node. In een bag zijn er maximaal tw+1 vertices (met tw de treewidth) die allemaal een kleur moeten krijgen en een happiness value (happy of unhappy). Dit komt dan neer op

$$width \leq k^{tw+1}2^{tw+1}$$

met k het aantal kleuren. In het exacte algoritme houd ik ook nog een set vertices waarin een vertex van elk mogelijk kleur zit (deze zijn voorgekleurde vertices). Dit is echter niet per se nodig voor het algoritme en zorgde er allemaal maar voor dat bovenstaande bovengrens groter werd, namelijk met een factor 2^k omdat al deze vertices ook of happy of unhappy kunnen zijn.

Verder zou ik nu nog dit laatste deel willen implementeren en dan nog verschillende experimenten uitvoeren:

- Een algemene test van het algoritme tegenover greedy en growth MHV. Hiervoor zal ik dan eerst het algoritme tunen (via SMAC) en vervolgens de gevonden parameters gebruiken om het algoritme te runnen op instanties die gekend zijn om moeilijk te zijn voor MHV
- Vervolgens wil ik dan ook nog kijken hoeveel het algoritme verbeterd naargelang ik de *width* groter zet. Ik zou dit dan ook bekijken in functie van het aantal voorgekleurde vertices want voor greedy en growth MHV geeft dit ook een verschil.
- Uiteindelijk zou ik dan ook nog de willen kijken wat de invloed is van de width van de tree decomposition op het algoritme. Normaal zou een kleine treewidth er moeten voor zorgen dat het algoritme sneller gaat, maar het is misschien ook zo dat de evaluatie beter is. Hiervoor zal ik de uiteindelijke evaluatie wel iets anders moeten doen. In voorgaande puntjes ben ik van plan om puur te kijken naar het aantal procent van de vertices dat happy zijn. In dit geval zal ik eerder kijken naar het verschil met greedy en growth MHV (eg greedy MHV geeft 60% happy vertices en mijn algoritme geeft 50% happy vertices dus een evaluatie van -10%). Dit is omdat ik andere grafen zal nemen en het is mogelijk dat grafen met een kleinere treewidth op natuurlijke wijze al procentueel meer happy vertices hebben (aangezien ze meer tree-like) zijn, waardoor puur het procentueel aantal happy vertices vergelijken tussen deze verschillende instanties ‘oneerlijk’ is volgens mij.