1

Variablene er er henholdsvis startposisjonen for hvert segment i hver rad, og startposisjonen for hvert segment i hver kolonne. Det gir oss samme antall variabler som vi har kolonner og rader til sammen.

Domenet blir kalkulert i starten av algoritmen og er alle kombinasjoner som er lovlige på rad/kolonne-nivå. Her blir det ikke tatt hensyn til samspillet mellom rader og kolonner, men bare på at segmentene er atskilt av minst et hvitt felt.

Restriksjonen er samspillet mellom rader og kolonner. Det vil si at hvis alle lovlige kombinasjoner for en rad har et segment i posisjon x, må tilsvarende kolonne også ha et segment i posisjon y. (hvor posisjon x,y er skjæringspunktet mellom rad og kolonne). Dette gjelder også for situasjoner hvor man vet det ikke kan være et segment tilstede i en blokk.

2

Både h og valg av assumpsjonsvariabel var vanskelig å teste siden alle problemene ble løst på bare en node. Derfor har vi valgt å holde oss til simpel utgave som i alle fall burde fungere greit i en god del tilfeller.

h = summen av (alle lovlige kombinasjoner for hver rad/kolonne -1).

Dette gir oss antall overflødige lovlige kombinasjoner som må renses ut før vi har et svar. (Et svar må nødvendigvis ha en lovlig kombinasjon per rad derfor trekker vi fra 1, for hver rad kolonne)

Variabel for neste assumpsjon velges som den radvariabelen som har færrest mulige potensielle posisjoner. Dette gjøres for å minimere antallet barn som genereres (siden det er dyrt å kjøre GAC på hvert barn), samt ivareta friheten til problemet lengst mulig så man unngår mange søk som ender i ulovlige brett.

3

A*-klassen er den samme som ble brukt i forrige modul. GAC-implementasjonen ligger som en del av wrapperen for tilpasning av A* til nonogrammer. Dette er den eneste særklassen som er

brukt for å implementere GAC systemet. Wrapperens sentrale metoder er: generate_children, redo, arc_cost, is_solution, calculate_h og generate_line_alternatives.

De eneste sentrale submetoden som er lagt til kontra normal A* er ”redo_all” og ”generate_line_alternatives”.

- “redo_all” utfører samme oppgave som ”domain-filtering loop” i materialet for modul 2. Den benytter seg av “redo” som utfører samme oppgave som REVISE.
- ”generate_line_alternatives” initialiserer domenet for hver variabel. Den tilsvarer initialize i materialet for modul 2.
- “Generate_children” velger en rad hvor man gjør en antakelse og generer alle med antakelser på den raden
- “Arc_cost” er alltid 0 da det ikke innebærer noen kostnad å være dypt i søketreet
- “Is_solution” sjekker domenet til hver variabel og returnerer True hvis alle domenene har størrelse 1
- “calculate_h” er er beskrevet tidligere.

4

Et sentralt designvalg var måten vi representerte problemet og restriksjonene på. Vi har valgt å representere problemet som en 2-dimensjonal tabell med startposisjoner for hver av segmentene for hver rad.

Formålet med dette var å lett kunne implementere en restriksjon på at rader og kolonner måtte ha samme celleverdi samt å kjøre en restriktiv initialisering. Dette viste seg å være to kraftige restriksjoner som har vært i stand til å løse alle problemene uten å kreve antakelser/gjett fra A*.

Hvordan kjøre koden

For å kjøre koden skrives “python3 NonogramGUI.py” i terminalen. Deretter spør programmet om filnavn. Det oppgis uten ”.txt” f.eks. ”cat” eller “elephant”

GUI’et viser det ferdige brettet. Hvis det er generert mer enn en node, kan man bla frem og tilbake for å se de forskjellige nodene som er en del av den endelige løsningen.

Hvis man vil se alle nodene som er utforsket i deres gitte rekkefølge kan man trykke pil opp og bla igjennom fra rotnoden.