Verslag CI 2022 - Sudoku

Justin Nguyen 7085842

Ik heb het hill climbing algoritme getest op 5 puzzels. Het hill climbing algoritme gebruikte bij mijn implementatie twee parameters:

- **Stopcriteria:** deze parameter geeft aan hoelang het algoritme mag doorgaan totdat er een plateau geconstateerd wordt. Als deze parameter te laag gekozen wordt dat zal het algoritme al snel zeggen dat er een plateau is (en dus een lokaal minimum heeft gevonden). Het nadeel hiervan is dat algoritme misschien niet werkelijk in een lokaal minimum zit. Het voordeel hiervan is dan natuurlijk dat er geen lange periodes zijn waar het algoritme vast zit in een lokaal minimum.
- **Random-walk stappen:** deze parameter geeft aan hoeveel willekeurige stappen het algoritme moet nemen als het een plateau gevonden heeft. Ook bij deze parameter keuze is het belangrijk om niet een te lage waarde te kiezen. Bij een te lage keuze is het mogelijk dat het algoritme nog steeds vast blijft hangen in hetzelfde lokaal minimum. Een te hoge waarde voor deze parameter kan ervoor zorgen dat het algoritme langer moet zoeken. Bij een hoge waarde ontsnap je wel uit het lokaal minimum maar dit betekent ook dat je ongericht gaat zoeken en dus misschien verder van je doeltoestand beland.

Hierbij heb ik twee soorten experimenten uitgevoerd. In de experimenten heb ik verschillende waarden voor de stopcriteria en random-walk stappen getest. Om de parameter waarden te testen heb ik het aantal wissels dat het algoritme heeft uitvoert bijgehouden (de random-walk wissels niet meegerekend). Dit heb ik per puzzel 25x gedaan, ook heb ik het gemiddelde en maximum berekent. Daarnaast heb ik ook het gemiddeld aantal random walks bijgehouden.

Stopcriteria:

Bij stopcriteria: 1**; Random-walk stappen:** 5**:**

Puzzel Nr.	Gemiddeld aantal stappen	Gemiddeld aantal random walks	Maximum aantal stappen	Minimum aantal stappen
1	14968	30326	49221	1264
2	38817	78206	144573	2724
3	202249	409505	700653	39182
4	39023	79656	119338	1121
5	706	1418	2314	135

Bij stopcriteria: 10**; Random-walk stappen:** 5**:**

Puzzel Nr.	Gemiddeld aantal stappen	Gemiddeld aantal random walks	Maximum aantal stappen	Minimum aantal stappen
1	4072	847	13640	224
2	10450	2189	43726	1280
3	38813	8202	134758	4471
4	9909	2107	28207	211
5	258	42	1164	84

Bij stopcriteria: 100**; Random-walk stappen:** 5**:**

Puzzel Nr.	Gemiddeld aantal stappen	Gemiddeld aantal random walks	Maximum aantal stappen	Minimum aantal stappen
1	7462	256	21174	621
2	7375	245	21651	299
3	39631	1370	198805	2095
4	21140	736	117792	384
5	248	6	1371	44

Bij stopcriteria: 1000**; Random-walk stappen:** 5**:**

Puzzel Nr.	Gemiddeld aantal stappen	Gemiddeld aantal random walks	Maximum aantal stappen	Minimum aantal stappen
1	21074	97	74246	153
2	60533	278	158701	3813
3	219886	1019	750300	4323
4	60909	282	242488	1356
5	331	1	3172	52

Random-walk stappen:

Bij stopcriteria: 5**; Random-walk stappen:** 1**:**

Puzzel Nr.	Gemiddeld aantal stappen	Gemiddeld aantal random walks	Maximum aantal stappen	Minimum aantal stappen
1	12861	1815	53601	540
2	10683	1494	57497	309
3	45261	6321	166691	331
4	38521	5462	196424	2927
5	576	76	3776	51

Bij stopcriteria: 5**; Random-walk stappen:** 10**:**

Puzzel Nr.	**Gemiddeld aantal stappen**	**Gemiddeld aantal random walks**	**Maximum aantal stappen**	**Minimum aantal stappen**
1	15961	9888	62946	1303
2	47759	29221	167725	2884
3	291603	183488	979112	12099
4	69864	43819	356088	1340
5	422	254	1114	66

Bij stopcriteria: 5**; Random-walk stappen:** 25**:**

Puzzel Nr.	**Gemiddeld aantal stappen**	**Gemiddeld aantal random walks**	**Maximum aantal stappen**	**Minimum aantal stappen**
1	174753	194036	1016119	865
2	904465	993455	2672467	70172
3	-	-	9212716	142669
4	678001	749723	2895213	4114
5	1674	1739	5995	50

Bij stopcriteria: 5**; Random-walk stappen:** 100**:**

Puzzel Nr.	**Gemiddeld aantal stappen**	**Gemiddeld aantal random walks**	**Maximum aantal stappen**	**Minimum aantal stappen**
1	820605	3167616	2674200	13964
2	-	-	-	-
3	-	-	-	-
4	-	-	-	-
5	3723	13164	19359	154

Conclusie:

Parameter stopcriteria: Bij het kiezen van de waarde van de stopcriteria was een waarde van 1 te klein, hierdoor had het algoritme een relatief lange rekentijd. Uit de resultaten is te concluderen dat waardes in het bereik van 10 / 100 redelijk goede resultaten opleveren. Bij een parameter keuze van 10 is het algoritme namelijk gemiddeld 3x sneller dan bij een parameter keuze van 1. Bij een parameter keuze van 100 lijkt het algoritme voor sommige puzzels even snel te zijn. Bij een parameter keuze van 1000 gaat het algoritme overduidelijk slechter presteren dan bij alle andere parameter keuzes. Dit komt doordat het algoritme altijd in een lokaal minimum blijft hangen. Dit betekent echter wel dat het algoritme veel minder vaak een random stap hoeft te maken.

De meest optimale waarde voor de stopcriteria zal ongeveer rond de 10 liggen. Dit komt omdat er in elke iteratie een blok van wissels bekeken wordt. Als je dus alle 9 blokken gaat bekijken en er blijkt geen verbetering te zijn, dan betekent dit 100% dat je in een lokaal minimum zit en dus random walks moet gaan uitvoeren om uit het lokaal minimum te komen. Als de parameterwaarde dus veel groter wordt e.g. 1000, dan zal er onnodig veel rekenwerk verricht worden, aangezien je na een bepaald aantal iteraties van geen verbetering al kan concluderen dat je in een lokaal minimum zit.

Parameter random-walk stappen: Bij het kiezen van het aantal random-walk stappen die het algoritme moet uitvoeren is het van belang om deze niet te groot te kiezen. Uit de experimenten is gebleken dat er al bij een parameter keuze van 25, enorm veel rekentijd en ruimte vereist is. De vereiste rekentijd bij waardes hoger dan 25 lag gemiddeld hoger dan 30 minuten. Daarnaast gebruikte het algoritme soms meer dan 20 gb aan geheugen voor het rekenwerk. Uit de resultaten is gebleken dat een parameterwaarde van 1 het snelste was voor alle puzzels.

Omdat het algoritme in dit geval optimaal is bij een waarde van 1, kunnen we mogelijk concluderen dat het algoritme makkelijk uit een lokaal minimum kan ontsnappen (in dit geval). Een keuze van een hogere waarde zal je dus vaak verder van je doeltoestand brengen dan nodig is. Een voorbeeld kan beschreven worden als volgt:

- Stel je loopt van punt A naar B. In 10% van de tijd dat je loopt kan je een duidelijke route zien. In de andere 90% van de tijd loop je geblinddoekt en gedesoriënteerd rond. Deze twee situaties gebeuren door elkaar heen, het is dus goed mogelijk dat je na een lange tijd geblinddoekt lopen, in een toestand terecht komt waar je mogelijk nog verder van de doeltoestand af bent dan je begintoestand. Als je dus niet vaak genoeg gericht loopt, zal je puur op geluk je doeltoestand moeten bereiken.

Dit is ook de situatie waar het algoritme zich in bevindt als het teveel random-walk stappen moet gaan zetten. Het algoritme kan niet meer gericht genoeg gaan zoeken, en moet dus puur op geluk de doeltoestand bereiken.

Puzzels

Uit de experimenten is overduidelijk gebleken dat de laatste puzzel het makkelijkst op te lossen was. In elk parameter keuze was puzzel 5 extreem snel klaar met rekenen, dit komt waarschijnlijk doordat de begintoestand erg gunstig was, e.g. meer en beter gefixeerde waardes. Puzzel 3 was in elk geval het moeilijkste op te lossen, deze kostte bij alle parameters keuzes significant meer rekenwerk dan andere puzzels. Puzzel 1 was over het algemeen langzamer dan puzzel 5 maar sneller dan de rest. En puzzel 4 en 2 vereiste een meeste situaties ongeveer dezelfde hoeveelheid rekenwerk.

Overig:

Zelfreflectie

Het programmeren van het algoritme zelf vereiste niet heel bijzonder veel werk. Het hill-climbing algoritme en het random-walk algoritme kostte ongeveer 1-2 uur aan werk. Het meeste programmeerwerk zat zich vooral in de representatie van het probleem zelf. Dus de Grid class en alle functie in de Solver class om het probleem te initialiseren. Daarnaast zat er wat tijd in het oplossing van bugs, e.g.:

- Bij het initialiseren van de beginscore van het grid, dit moest namelijk gebeuren nadat alle nullen een waarde hebben gekregen en niet daarna.
- Swaps moesten uitgevoerd worden op mijn kolom en rij, ik dacht dat elke waarde in een rij gepaard was met de juiste waarde in een bepaalde kolom aangezien ze bij het initialiseren naar hetzelfde object wezen.

Opmerkingen:

- Commentaar voor elke gebruikte functie staat in de code zelf
- Om een nieuwe puzzel te testen kan er in de combobox (waar al standaardwaarden in staan) ingevuld worden. Deze input moet wel van dezelfde vorm zijn als de gegeven puzzels. Daarna moet er eerste op reset gedrukt worden voordat er op Solve gedrukt wordt.