**College 3 Heuristieken**

Constructief zoeken: depth-first  
- recursief uitprogrammeren.  
- gevaar:

* dat het algoritme niet stopt. Er kunnen er loops inzitten
* Je kan dit controleren door elke x staa van het bord op te slaan en vervolgen te checken of je deze staat al hebt gehad.

- zuivere depth-first is brute-force  
- pruning: onmogelijke situaties gelijk vermijden.

* Geen heuristiek. Je zou sowieso alle niet mogelijke staten weg moeten gooien. Zuivere optimalisatie
* Heuristiek zit een gok aspect in

- compleet: Betekent dat ie of de oplossing teruggeeft, of aan geeft dat er geen mogelijke oplossing is.

Constructief zoeken: breadth-first  
- hoeveel geheugen nodig voor dit probleem. Hee veel  
-oook met breadth-first kan je prunen

* Gooie oplossingen eruit die sowieso niet kunnen. Pure optimalisatie

- ook checken op dubbele states

**Verschillen**- breadth-first werkt met een queue. First in, first out  
- depht-first werkt met een stack: Alles wat er in komt, komt op niveau erin. Je bewaart deze allemaal.

**Wat gebruik je voor minikakuro?**- waarom depth-first beter dan breadth-first.  
- de oplossing ligt op depth 8. Dit is zowel de minimale als maximale run time  
- je moet dus sowieso naar deze state toe te gaan. Zou je breadth-first gebruiken, dan moet je dus al die states opslaan in je geheugen.

**Schuifpuzzel**- we hebben nu verschillende kwaliteiten van oplossingen.  
- je moet in zo min mogelijk zeten de oplossing vinden

**Search tree**- alle nodes zijn states  
- degree d  
- diepte is geen diepte  
- #nodes = State-Space  
- state-space: een compound polyhedron  
- alle nodes (362 888)  
- degree 4 (11%), 3 (44%), 2 (44%).  
- maximale afstand (diameter)

* Je weet dan dat het maximaal aantal stappen dat je kan doen, dus de maximale diepte, de diameter is. Zo kan je iets wat je eerst met breadth-first wilde done opeens met depht first doen.

**8-puzzle: De invloed van loops**- hoe dieper je gaat, hoe meer duplicaten  
- bij iedere stap die je zet, krijg je meer baggage. Je neemt zeg maar de duplicaten mee van de stappen daarvoor en je behoudt ook gewoon de duplicaten van de stappen erna.

**Breadth-first met archief**

**Hill climbing**- 10 steden in een XY-vlak  
- vind de kortste route  
- hoe groot is de toestandsruimte

1. Random beginsituatie
2. Bepaalde score
3. Maak kopie, breng één wijziging aan
4. Bepaal de score
5. Is de kopie beter ? dan houden we die? Anders houden we het origineel
6. Ga terug naar 3 totdat er lange tijd gen verbetering is.   
   \*\* Monte carlo simulatie? Nee, want dat pakt alleen maar random start posities.   
   \*\* Maak kleine mutaties bij een hill climber.

**10 experiments**- 1000 iteraties, record alle data  
- iteratei: swap twee steden  
- hoeveelswaps voor stabilisatie  
- hoeveel swaps tav begin waarde

**Conclusie**- ocassionally: lokaal minimum  
- veranderd niets meer  
- goede score, maar geen globaal minimum.