**Random Algoritme (geeft een oplossing op alle borden)**

Met het gebruik van het random-algoritme hebben we voor ieder bord een oplossing gevonden. Dit algoritme zoekt een willekeurige auto op het bord en verplaatst hem over een willekeurige afstand. Dit algoritme hoeft niks op te slaan (in tegenstelling tot de andere algoritmes in onze vergelijking) Dit zorgt er dan ook voor dat de random algoritme het snelst een oplossing vindt. Het random algoritme zet willekeurige stappen en hoeft alleen deze stappen (de route) te onthouden terwijl de breadth-first langzaam een *tree* afgaat die steeds groter wordt.

**Breadth-First Algoritme (geeft tot nu toe op de eerste drie borden een oplossing)**

Het breadth-first algoritme vindt gegarandeerd de best mogelijke oplossing. Met de beste oplossing wordt het minst aantal stappen bedoeld. Dit algoritme vormt een *tree* met alle mogelijke *moves*, waarbij hij elke generatie stap voor stap afgaat. Het nadeel van dit algoritme is dat deze vrij langzaam te werk gaat. Voor de eerste drie borden met een grootte van 6x6 vindt het algoritme nog binnen een paar minuten de beste oplossing, aangezien de *statespace* relatief klein is (kleiner dan een miljoen). Bij de grotere borden duurt het vinden van een oplossing significant langer (NOG 9X9 UITTESTEN MET BF VOOR N UUR OFZO). De *statespace* voor deze borden is dan namelijk een getal met bijna 20 nullen.

**Depth-First (geeft oplossing op … )**

Voor de borden groter dan 6x6 komt een depth-first algoritme beter van pas. Dit algoritme gaat net als de breadth-first een *tree* af, maar de depth-first slaat slechts één pad op in zijn geheugen. Hierdoor is het algoritme veel sneller dan de breadth-first, maar deze garandeert niet dat de beste oplossing gevonden wordt.

**Vergelijking algoritmes**

Het random algoritme kan men het best gebruiken als er gegarandeerd een oplossing gevonden moet worden. Het gaat hierbij niet om de beste, niet de meest efficiënte, maar wel de snelste. Verder kan dit algoritme alle borden (snel) oplossen. In vergelijking met het random algoritme vindt het breadth-first algoritme gegarandeerd de beste en kortste oplossing. Dit algoritme werkt in vergelijking met het random en depth-first algoritme het traagst en kan om deze reden niet alle borden oplossen. Tot slot zorgt het depth-first algoritme altijd voor een oplossing. Dit is niet de beste en kortste oplossing, maar hij vindt wel sneller een oplossing dan het breadth-first algoritme.

We kunnen de uitkomsten van het random-algoritme goed gebruiken om de depth-first te optimaliseren. Door de kortste uitkomst van het random-algoritme te nemen kunnen we dit gebruiken als upper bound. Als de depth-first een pad in de tree afgaat dat meer stappen nodig heeft dan die kortste oplossing kan hij doorgaan naar het volgende pad.