

Practicumverslag Computatieve Intelligentie

Cornelis Bouter (studentnummer) en Roel Brouwer (3976866)

27 juni 2014

Abstract

Chapter 1

Implementatie algoritmen

1.1 Hill-climbing

1.1.1 Algemeen

Zoekoperator

Het bepalen van de volgende toestand in het *hill-climbing* gebeurt aan de hand van de zoekoperator (hieronder in pseudo-code uitgewerkt). Deze geeft een nieuwe sudoku terug, die van de oude verschilt op twee vakjes: de eerste verwisseling die een verbetering opleverde in de evaluatiewaarde van de sudoku.

```

ZoekOperator (Sudoku)
    // eval bevat de waarde die de evaluatiefunctie aan de sudoku geeft.
    eval = Sudoku.evaluateer();
    // Verwisselen gaat per nxn-blok. g en h representeren het blok.
    // Het zijn constantes die bij de variabelen die de positie van
    // de vakjes binnen het blok representeren (i, j, k en l) worden
    // opgeteld om het juiste vakje te kunnen identificeren.
    // Tot is exclusief. In de volgende geneste loops worden alle
    // wissel-mogelijkheden afgelopen tot er een gevonden is die de
    // huidige toestand verbetert.
    for g = 0 tot n*n (ophogen per 3)
        for h = 0 tot n*n (ophogen per 3)
            for i = 0 tot n
                for j = 0 tot n
                    for k = 0 tot n
                        for l = 0 tot n
                            if ((i,j) en (k,l) representeren niet
                                hetzelfde vakje EN
                                (i,j) en (k,l) zijn variabele vakjes)
                                // Verkrijg de nieuwe sudoku door de vakjes
                                te verwisselen.
                                Nieuw = Sudoku.verwissel(g + i, h + j, g +
                                    k, h + l)
                                // Als de nieuwe beter is zijn we klaar.
                                if (Nieuw.evaluateer() > eval)
                                    return Nieuw;

    // Als er niks verwisseld kan worden, zitten we in een lokaal
    // optimum. We geven de oorspronkelijke sudoku terug.
    return Sudoku;

```

1.1.2 Random Restart

1.1.3 Iterated Local Search

Zoekoperator

Voor het implementeren van *iterated local search* is het nodig geweest de zoekoperator iets aan te passen, of beter gezegd: een variant op de zoekfunctie toe te voegen.

De zoekoperator die voor het beklimmen van de heuvel, het *hill-climbing* deel, gebruikt wordt, loopt alle swap-mogelijkheden in een bepaalde volgorde door, en kiest de eerste die een verbetering is. Voor *iterated local search* is een *random walk* nodig. Een *random walk* geeft niets om het antwoord op de vraag of een opvolger-toestand al dan niet een verbetering is. Het moet een willekeurige opvolger-toestand kunnen kiezen, ongeacht de waarde die de evaluatiefunctie aan die toestand geeft.

Wanneer de oude implementatie gebruikt zou worden, zou het dan steeds voor de eerste optie kiezen: het swappen van de eerste twee niet-vaste vakjes in

het blok in de linksbovenhoek van de sudoku. Dat willen we niet, daarom veranderen we de zoekoperator-functie zodat deze twee willekeurige vakjes (binnen hetzelfde blok - natuurlijk) van plaats verwisseld worden:

```
RandomZoekOperator (Sudoku)
    // g en h representeren het blok waarin het verwisselen wordt
    // uitgevoerd.
    g, h = randomInt(0, n) * n;
    // i, j, k en l representeren de twee vakjes die verwisseld worden.
    i, j, k, l = randomInt(0, n);
    while ((i,j) en (k,l) representeren hetzelfde vakje OF
        (i,j) of (k,l) is een vast vakje)
        // Nieuwe waarden voor i, j, k en l.
        i, j, k, l = randomInt(0, n);
    // Verwissel de genoemde vakjes in de sudoku en return het resultaat
    return Sudoku.verwissel(g + i, h + j, g + k, h + l);
```

1.2 Constraint Satisfaction

1.2.1 Backtracking

1.2.2 Forward Checking

Chapter 2

Experimentele resultaten

2.1 Hill-climbing

2.1.1 Algemene observaties

Accepteren gelijke toestanden

2.1.2 Random Restart

2.1.3 Iterated Local Search

2.2 Constraint Satisfaction

2.2.1 Backtracking

2.2.2 Forward Checking