Nanopond – genetisch programmeren

Practische sessie 2
Computationele intelligentie
Academiejaar 2010-2011

Pieter Pareit

18 november 2010

1 Program X

Het volgende programma zal met een minimale verzameling van instructies reproduceerbaar zijn:

```
READG     % zet register ≠ 0
LOOP      % als register = 0 spring voorwaarts naar bijhorende REP
READG     % plaats waarde van gnome op huidige positie in register
WRITEB    % schrijf waarde van register naar buffer op huidige positie
FWD       % verhoog de positie met één
REP       % als ≠ 0 spring terug naar bijhorende LOOP instructie
ZERO      % zorg dat het programma een 0 in het register plaatst
```

Voor de eerste instructie zijn een aantal verschillende mogelijkheden. De instructies `INC`, `DEC`, `READG` en `READB` plaatsen respectievelijk de waarden `0x1`, `0xf`, `0x5` en `0xf` in het register. In het verdere verloop van het programma wordt nog eens `READG` gebruikt. Dus hier is het minimale programma één die 6 verschillende instructies nodig heeft, 7 instructies plaatst in het genoom, en het verbruikt $2+7\cdot4+1=31$ eenheden energie tijdens het uitvoeren.

2 Mutation

De volgende code implementeert variatie in de wereld door tijdens het uitvoeren van een genome soms een genome te wijzigen, soms het huidige register te wijzigen, soms een instructie tweemaal uit te voeren of soms een instructie over te slaan.

```
370 if (rg.nextDouble() < MUTATION_RATE) {
371     int type = rg.nextInt(4);
372     switch (type) {
373         /* replacement */
374         case 0:
375             c.genome[instructionIndex] = (byte) rg.nextInt(16);
376             break;
377         /* change register */
378         case 1:
379             reg = (byte) rg.nextInt(16);
380             break;
381         /* duplicate instruction execution */
382         case 2:
```

```
383                 if (instructionIndex == 0) {
384                     instructionIndex = POND_DEPTH;
385                 }
386                 instructionIndex--;
387                 break;
388             /* skip instruction */
389             case 3:
390                 instructionIndex++;
391                 instructionIndex %= POND_DEPTH;
392                 break;
393         }
394 }
```

De code om een instructie te dupliceren, dupliceert eigenlijk niet de huidige instructie, maar de vorige. Dit maakt niet echt een verschil uit. Er wordt instructie-wrap voorzien, anders zou de laatste instructie nooit dubbel uitgevoerd worden. Alternatief zou de code nooit een instructie-duplicatie kunnen doen wanneer de index gelijk aan nul is.

De volgende code introduceert sporadisch een fout tijdens het kopiëren van een cel zijn buffer naar één van zijn buren:

```
570 int i = 0, j = 0;
571 while (i < POND_DEPTH && j < POND_DEPTH) {
572     if (rg.nextDouble() < MUTATION_RATE) {
573         int type = rg.nextInt(3);
574         switch (type) {
575             /* replacement */
576             case 0:
577                 neighbor.genome[i++] = (byte) rg.nextInt(16);
578                 j++;
579                 break;
580             /* duplicate instruction execution */
581             case 1:
582                 neighbor.genome[i++] = outputBuf[j];
583                 i %= POND_DEPTH;
584                 neighbor.genome[i++] = outputBuf[j++];
585                 break;
586             /* skip instruction */
587             case 2:
588                 i++;
589                 j++;
590                 break;
591         }
592     } else {
593         /* no mutation */
594         neighbor.genome[i++] = outputBuf[j++];
595     }
596 }
```

Een alternatief dat sneller loopt, maar waarvan dan een extra mutatie snelheid moet worden bepaald is de volgende. Eerst wordt de output buffer met een snelle `arraycopy` gekopieerd naar het buur genoom. Daarna wordt willekeurig (met een te bepalen mutatie snelheid) de kopie geselecteerd om mutaties te ondergaan. Enkel indien geselecteerd, wordt het tragere algoritme uitgevoerd.

3 World domination

Programma's die succesvol zijn moeten een korte uitvoeringstijd hebben, zichzelf in de buffer plaatsen, energie van naburige programma's overnemen en naburige programma's verwijderen. Door de vereisten van korte uitvoeringstijd en zichzelf kopiëren zijn alle succesvolle programma's kleine varianties van *Programma X*. Energie overnemen van buren kan door `SHARE` instructies binnen loops uit te voeren. De `KILL` instructie is verraderlijk omdat de eigen energie wordt gehalveerd bij falen. De kans tot falen wordt beïnvloed door de waarde van het register en door de waarden van de eerste instructie van het naburige programma dat aangevallen wordt, hoe meer verschillende bits tussen het register en de eerste instructie, hoe beter de kans.

Volgend programma *zou* daardoor succesvol moeten zijn:

```
READG
INC**4     % voer INC viermaal na elkaar uit
KILL
LOOP
READG
WRITEB
SHARE
FWD
REP
ZERO
```

De hex-code van dit programma is `0x53333d958e1a0f`. Volgende variantie's bleken beter: `0xd7d9581ea0f` en `0xdd7de9581a0f`.

Verder werd ook een programma ontwikkeld dat in twee fases werkt. De countdown fase, en de payload fase. Bedoeling van de countdown fase is om zo snel mogelijk een weg af te leggen, de bedoeling van de payload fase is om zich ergens te settelen en daar zo min mogelijk energie te verbruiken en erg defensief te zijn. De payload fase was echter efficiënter (minder energie verbruiken) dan de countdown fase en daardoor haalde de tweede fase vaak de eerste fase in.

```
READB      % maak register ≠ 0
LOOP       % zoek naar XCHG in eigen code
FWD
READG
INC**4
REP
FWD
READG      % lees counter
DEC        % tel één af
LOOP       % als register = 0, spring naar de payload fase
XCHG
STOP
LOOP       % ga met programma-pointer terug naar begin programma
BACK
READG
INC
REP
INC
LOOP       % kopieer nu eigen programma in buffer
READG
KILL
WRITEB
```

```
FWD
SHARE
REP
STOP      % countdown fase volledig, stop nu
REP
DEC       % wrap counter, dus herbegin tellen vanaf STOP
WRITEG
LOOP      % plaats programma-pointer op nul
BACK
READG
INC
REP
FWD       % dump payload in buffer
INC**3
WRITEG
FWD
DEC**6
WRITEG
FWD
DEC**4
WRITEG
FWD
DEC**4
WRITEG
FWD
INC**3
WRITEG
FWD
DEC**7
WRITEG
FWD
DEC**3
WRITEG
FWD
DEC**4
WRITEG
FWD
INC**6
WRITEG
FWD
ZERO
```