

Forum:[Hack Forums](#)

Thread: [R5RS SCHEME, Linear Gene Expression Programming \(Al\)](#)

MiguelSanchez - February 20, 2022, 5:21 am

Linear gene expression programming in R5RS scheme.

I've written traditional Linear gene expression programming, as well as a version where each gene has it's own inherited preference based on how many generations it has survived in that position. A kind of dominant/recessive system a little beyond punnet squares (only barely)

All the definition are there for creating the populations, iterating them according to a given fitness function.

LGE is standard, LGEp is with genewise preference.

```
Code (#%require srfi/27)
(load "Essentials.scm")
(define (LGE base code) `(LGE ,base ,code)) ;;Linear Gene Expression

(define (LGEp base code) `(LGE ,base ,code))

(define LGEp-Start-Preference -.9)

(define (LGEp-increment current-preference) (+ current-preference (- .2 (/ (+ current-preference 1) 10))))

(define (RandomLGEp base min-length max-length)
  (LGEp base (maprange
    (lambda (x) (cons (random-integer base) LGEp-Start-Preference))
    (+ min-length (random-integer (+ 1 (- max-length min-length)))))))

(define (RandomLGE base min-length max-length)
  (LGE base (maprange
    (lambda (x) (random-integer base))
    (+ min-length (random-integer (+ 1 (- max-length min-length)))))))

(define LGEA (randomLGEp 4 10 10))
(define LGEB (randomLGEp 4 10 10))

(define (LGE-split a b)
  (let ((aexpr (lge-expr a))
        (bexpr (lge-expr b))))
  (let ((split-pos (random-integer (min (length aexpr) (length bexpr)))))
    (if (zero? (random-integer 2))
        (append (genehead a split-pos) (genetail b split-pos))
        (append (genehead b split-pos) (genetail a split-pos)))
    )
  )))
```

```
(define (preferential-exchange a b)(let ((fitness-a (cdr a))
                                         (fitness-b (cdr b)))
  (let ((preference (max -1 (min 1 (/ (- fitness-a fitness-b) 2)))))
    (if (< (random-integer 100) (+ 50 (* 50 preference)))
        (cons (car a) (LGEp-increment (cdr a)))
        (cons (car b) (LGEp-increment (cdr b))))
    )))
```

```
(define (LGE-expr ge) (caddr ge))
(define (LGEp-expr ge) (map car (caddr ge)))
```

```
(define (LGEp-Exchange a b)
  (let ((aexpr (lge-expr a))
        (bexpr (lge-expr b)))
    (append (map (lambda (c d) (preferential-exchange c d))
                (mcar aexpr (min (length aexpr) (length bexpr)))
                (mcar bexpr (min (length aexpr) (length bexpr))))
            (map (lambda (z) (cons (car z)
                                   (lgep-increment (cdr z))))
                (append (mcdr aexpr (min (length aexpr) (length bexpr)))
                        (mcdr bexpr (min (length aexpr) (length bexpr))))
                )))
```

```
(LGEp-Exchange (randomlgep 4 5 10) (randomlgep 4 5 10))
```

```
(define (Genehead ge n) (mcar (LGE-expr ge) n))
```

```
(define (Genetail ge n) (mcdr (LGE-expr ge) n))
```

```
(define (LGE-exchange a b) (LGE-split a b))
```

```
(define (replace-index l i n val) (append (mcar l (- n 1)) (cons '())))
```

```
(define (LGE-mutate ge n)
  (random-indices ge (round (+ (/ (length '(1 2 3 4 5 6 7 8)) 10) 1)))
  )
```

```
(define (population function popsize base minlength maxlength) (maprange (lambda (x) (function
base minlength maxlength)) popsize) )
```

```
(define (LGEp-population popsize base) (population randomlgep popsize base 2 15))
```

```
(define (fitness-sort l) l)
```

```
(define (iterate-population population fitnessfunction merge-function selection-function birthrate)
  (let ((fittest (fitness-sort (map
                                (lambda (x) (cons x (fitnessfunction x)))
                                population))))
    (maprange (lambda (n) (merge-function (car (selection-function fittest)) (car (selection-function
fittest)))) (* (length population) birthrate)))
  )
```

```
(define (grow-population population newcandidates) pass)
```

```
(define (quarterselect list) (if (= (random-integer 4) 1)
    (index list
      (floor (+ (/ (length list) 4) (random-integer (floor (* 3 (/ (length list) 4)))))))
    (index list
      (random-integer (round (/ (length list) 4) )))))
```

```
(define (lge-fitfunc f) (lambda (ge) (f (lge-expr ge))))
```

```
(define (lgep-fitfunc f) (lambda (ge) (f (lgep-expr ge))))
```

```
(define (list-difference a b) (if (or (null? a) (null? b)) 0 (+ (abs (- (car a) (car b))) (list-difference (cdr a) (cdr b)))))
```

;takes list as input returns function that take function as input, returns how different the function is from the list

```
(define (fitness-listgen l) (lambda (f) (list-difference l (maprange f (length l)))))
((fitness-listgen '(1 2 4)) (lambda (x) (expt 2 x)))
```

```
;iterate-population
; (LGEp-population 20 8)
; (lgep-fitfunc YOURFUNCTIONHERE)
; lgep-exchange quarterselect 4)
```

Please forgive me for not documenting everything here, people are welcome to ask questions, i wasn't 100% finished (I never am ?) so I wouldn't be surprised if something is broken.

Feel free to let me know, maybe if functional programmers become more visible we'll finally get our own board.

Seriously, everyone should learn emacs lisp and network our interpreters, we would be unstoppable (even with like 10 people).

I have a \$^#\$load more scheme to drop, but it's too big and uncommented for me to assume anyone would want to see it.