Universidad Nacional de Costa Rica
Facultad de Ciencias Exactas y Naturales
Escuela de Informática

Paradigmas de Programación

Proyecto 2.

Prof.

Georges Alfaro

Estudiantes:

Riccardo Bove

116310708

Edwin Fernández

304560080

Github:

https://github.com/DarthKazar/Proyecto2Paradigmas.git

II Ciclo 2016

1. Descripción.

El problema a resolver se presentó de la siguiente manera.

“El problema consiste en elaborar un programa que pueda aproximar la forma de una función arbitraria, por medio de un algoritmo genético… El programa debe aproximar una función real de dos variables… La función a aproximar es una composición arbitraria de funciones definidas por operadores binarios. Los operadores a considerar corresponden con cada una de las operaciones aritméticas fundamentales y la exponenciación… La función a aproximar es una composición de las funciones anteriores hasta una profundidad determinada, por medio de un valor definido como parámetro. Si considera el árbol de evaluación de la función, el parámetro determinará la profundidad máxima del árbol de evaluación… El programa deberá generar una población inicial de un tamaño determinado (indicado por un parámetro del programa), que contenga funciones generadas de manera aleatoria, usando los operadores indicados y un árbol de evaluación de una altura máxima especificada… Las funciones serán seleccionadas según su valor de ajuste, considerando primero aquellas que tengan un valor menor (que minimicen el valor de la función de ajuste). El programa conservará un porcentaje (predeterminado en los parámetros del programa) de los individuos de cada generación (elitismo). El resto de los individuos en cada nueva generación se incluirán por medio de cruces y mutaciones de las funciones originales. Para cruzar dos funciones, se tomarán los subárboles izquierdo y derecho de cada función respectivamente. Se conserva la operación en el nodo raíz de la primera función… Las mutaciones pueden aplicarse sobre las funciones originales o sobre las funciones obtenidas por cruces. Puede considerar ambas técnicas o una combinación de ambas, comparar los resultados y decidir la forma más adecuada… ”

Tomado del enunciado del proyecto.

2. Solución.

A continuación se detalla brevemente la solución planteada para el problema a resolver.

a. Abordaje.

Dado que el objetivo del proyecto es implementar técnicas básicas de la programación funcional, se decidió abarcar el problema dividiendo la solución en módulos más pequeños los cuales se encargaban de funciones simples. Luego se esperaba poder reunir esta distribución modular en una única función general configurable por medio de una lista de parámetros definida al inicio del código. Se quiso dividir el programa en 3 secciones: definiciones, funciones base y funciones derivadas. Esto con el fin de tener una mejor legibilidad de la solución y así darle un mejor mantenimiento a lo largo de la implementación. En la sección de *definiciones* se establece la lista de parámetros para

configurar la aplicación. También se enlista los operadores y operandos necesarios. Seguidamente se encuentra la sección *funciones base*, la cual contiene la distribución modular que sería utilizada por las *funciones derivadas*, donde se reunirían todas las implementaciones en un alcance más amplio. Como sección adjunta se encuentra la parte de *pruebas* en donde se colocaron a modo de comentarios las líneas de código que comprobaban el funcionamiento de lo implementado.

b. Funciones.

En términos generales se visualizaron 4 funcionalidades principales, las cuales son: Mutación, Inicialización, Ajuste y Cruces. Fueron implementados basándonos en métodos vistos en clase.

Para la Mutación se definió implementar la de manera sencilla modificando el nodo raíz, el cual siempre sería un operador, y sustituyendo lo aleatoriamente. Esto debido a que se consideró como el nodo más importante y el que representa la identidad de la expresión. Para esto se creó una función base llamada *toList* que se encarga de generar un operador al azar y presentarlo como una lista de un único elemento para poder ser unido con el resto de la expresión utilizando la función derivada *mutar*.

Para el Cruce se necesitó únicamente una función base que tomara las ramas izquierda y derecha de sus dos parámetros respectivamente.

La Inicialización contó con una función de carga de lista, una de generador aleatorio de operadores y operandos, un generador de expresiones y uno final generador de la función. Todas funciones base.

Finalmente el Ajuste comprende la función *fitness* que toma dos listas y calcula el cuadrado de sus diferencias para cada elemento para cada par de puntos dados, de forma recursiva.

c. Limitantes.

Pese a ser un proyecto relativamente corto, el nivel de complejidad resultó terminar siendo un impedimento para avanzar a la velocidad deseada. La principal dificultad se presentó al tratar de comprender la funcionalidad total y el alcance general de lo solicitado en el proyecto. Debido a lo mencionado nos fue imposible lograr implementar la totalidad de lo requerido y algunos requerimientos menores. Las 4 funciones principales están implementadas en forma general, sin embargo no cuentan con una estructura que engloba sus alcances en conjunto. Debido a esto la aplicación es incapaz de ser configurada utilizando la lista de parámetros y también no cuenta con la capacidad de cargar los datos por medio de un llamado inicial. Las posibles soluciones a lo expuesto en este apartado se detallan en el siguiente.

d. Resultados.

En base a lo detallado en el apartado anterior, se obtuvo como resultado más notorio que la complejidad del proyecto recayó principalmente en la dificultad para comprender la totalidad de funcionalidades que requería la aplicación. Por lo tanto, creemos que con un manejo iterativo de los objetivos del proyecto se hubiese podido abarcar de una mejor manera la solución al problema planteado. Así también, al no estar presente la función general que controlaría todas las capacidades del programa, se estima que crear una función en la cual un llamado secuencial de todas las funcionalidades implementadas permitiría visualizar la interacción entre cada parte y poder, de esta manera, detectar inconsistencias y errores en el flujo de datos como un conjunto. Actualmente la mayoría de las funciones implementadas funcionan correctamente por separado, brindando los parámetros adecuados.

3. Conclusión.

Enfrentar los retos de un nuevo paradigma de programación para resolver un problema es una gran experiencia para ampliar las capacidades como ingenieros. Pese a que se presentaron algunas dificultades, fue de gran aprendizaje el enfrentarse a problemas con este tipo de alcance. Se alcanzó claramente un mejor entendimiento de la programación funcional y su utilidad en ciertos ámbitos. Finalmente, podemos decir que el curso de Paradigmas de Programación resultó paralelamente provechoso.

4. Anexos.

Se adjunta código fuente completo.

```
#lang racket
(define ns (make-base-namespace))
#|
Proyecto 2 Paradigmas de Programación.
Generador de funciones por medio de algoritmo genético.
II Ciclo 2016.
Prof. Georges Alfaro.
Estudiantes:
-Riccardo Bove - 116310708.
-Edwin Fernández Fernández -304560080.
|#

;**********DEFINICIONES*******************************
(define operadores '(+ * - / expt))
(define operandos '(x y a b))
;Operadores y operandos básicos utilizados por el generador.
(define parametros '(5 3 2 1 4))
;Parámetros: (población,generaciones,desviación,elitismo,mutaciones).
;*****************************************************************

;-----------------------------------------------------------------

;**********FUNCIONES BASE*****************************
(define elemento (lambda (L) (list-ref L (random (length L))) ) )
;Función base para obtener un operador u operando al azar.

(define expresion (lambda (n) (cond ((zero? n) (elemento operandos)) (else
   (list (elemento operadores) (expresion (random n))(expresion (random n)) ) ) ) ) )
;Generador base de expresiones aritméticas (polinomios).

(define generador (lambda (n p)            (if (zero? n) empty
                                  (cons (list 'lambda '(x y a b) (expresion p)) (generador (- n 1) p)) )))
;Generador base de funciones aritméticas y potencia.

(define cruces (lambda (f1 f2) (list (car f1)(cadr f1)(caddr f2)) ) )
;Cruza dos expresiones, lado izq. de una con el der. de la otra.

(define toList (list (elemento operadores)) )
;Convierte un elemento al azar en una lista de un elemento.
;*****************************************************************

;-----------------------------------------------------------------

;**********FUNCIONES DERIVADAS*************************
(define mutar (lambda (expr)
   (append toList (cdr expr)) ))
```

```
;Mutación simple de una expresión que reemplaza el nodo raíz.

(define q
  (lambda (f L)
    (map
     (lambda (p)
       (list p (f (car p) (cadr p))))
     L)
    ))
;Generador de valores de entrada a la función.

(define fitness
  (lambda (L1 L2)
    (cond
      [(or (null? L1) (null? L2)) '()]
      [(and (null? (cdr L1)) (null? (cdr L2))) (expt (- (car L2) (car L1)) 2)]
      [(+ (expt (- (car L2) (car L1)) 2) (fitness (cdr L1) (cdr L2)))]
        )
    )
  )
;Función de ajuste para funciones derivadas, determina viabilidad.

;*********************************************************************

;---------------------------------------------------------------------

;************************PRUEBAS**********************************
;(elemento operadores)
;(elemento operando)
;(expresion 2)
;(expresion 3)
;(generador 2 2)
;(generador 2 3)
;(cruces (expresion 2) (expresion 3) )
;(define expr '(+ (* (expt x 2) (- 1 y)) 1))
;expr
;(mutar expr)

;**********Funcion de ejemplo**********
#|(define f
  (lambda (x y)
    (+ (* (expt x 2) (- 1 y))
       (/ (* 4 (expt y x)) (+ 3 (* 2 y)))
       )
    )
  )|#
;***************************************
;(generador 2 2)
;(q f '((1 1) (1 2) (2 2) (1 3)))
;((eval (car (generador 2 2)) ns) 1 1 (random) (random))
;(define l1 (list 3 4 7 6))
;(define l2 (list 3 3 6 6))
;(fitness l1 l2)
;*********************************************************************
```