Manual de Usuario del vgGA1.3

Manuel Valenzuela Rendón

Escuela de Ingeniería y Ciencias
Tecnológico de Monterrey, Campus Monterrey

21 de octubre de 2014

1. Introducción

Este documento explica la forma de utilizar la implementación del vgGA (virtual gene Genetic Algorithm) en MATLAB vgGA1.3. El vgGA (Valenzuela-Rendón, 2003) está basado en la idea de mapear las operaciones sobre el genotipo a operaciones sobre el fenotipo, viendo éste como un vector de números enteros o reales.

2. Directorios `Ifuns` y `Gfuns`

Asumimos que el vgGA está instalado en un directorio llamado `vgGA1.3/`. Dentro de este directorio se encuentran los directorios `Ifuns/` y `Gfuns/` que contienen funciones que operan sobre fenotipos enteros para dígitos tradicionales y para digítos generalizados, respectivamente. Las funciones en los directorios `Ifuns/` y `Gfuns/` sirven solamente para mostrar cómo las ideas que dieron origen al vgGA fueron generalizándose, y no se requieren para la operación usual del vgGA, por eso, no se explicará su uso en este documento. La figura 1 muestran los listados de los directorios `Ifuns/` y `Gfuns/`.

```
Ifuns/                  Gfuns/
   iMutate.m               gdigit.m
   icrossAt.m              ghigh.m
   idigit.m                ghighh.m
   ihigh.m                 glow.m
   ihighh.m                gmutValues.m
   ilow.m                  gsegment.m
   imutValues.m
   isegment.m
```

Figura 1: Listado de los directorios `Ifun/` y `Gfuns/`.

3. Funciones básicas

El directorio `vgGA1.3/` contiene funciones básicas para la operación del vgGA y la idea de realizar operaciones de cruce y mutación sobre el fenotipo. Estas funciones operan sobre fenotipos reales (y por lo tanto también operan sobre fenotipos enteros) para dígitos tradicionales y generalizados:

`low(r,k,rMin=0)` Regresa la parte baja de un número `r` hasta el valor `k`. El número `r` tiene un valor mínimo `rMin`.

`high(r,k,rMin=0)` Regresa la parte alta de un número `r` a partir del valor `k`. Opcionalmente se puede dar el valor mínimo de `r` como `rMin`.

`highh(r,k,rMin=0)` Regresa el valor de la parte alta la parte alta de un número `r` a partir del valor `k`. Opcionalmente se puede dar el valor mínimo de `r` como `rMin`.

`segment(r,k,delta=2,rMin=0)` Regresa el segmento de ancho `delta` a partir, e incluyendo, el valor `k` del número `r`. `delta` debe tomar un valor entero mayor a `1`. Se puede especificar el valor mínimo de `r` como `rMin`.

`digit(p,n,B)` Obtiene el dígito de peso `n`/`B` de base `B` del entero `p`.

`[h1,h2]=crossAt(r1,r2,k,rMin=0)` Cruce los números `r1` y `r2` de valor mínimo `rMin` en el valor `k`, y regresa los hijos `h1` y `h2`.

`[h1 h2] = crossAtUniform(p1,p2,template,B=2)` Cruce uniforme de los números enteros `p1` y `p2` de base `B` de acuerdo a la plantilla binaria `template`. Solamente funciona para números enteros y dígitos tradicionales.

`mutate(r,k,delta,rMin=0)` Regresa el resultado de mutar el número `r` con valor mínimo `rMin` en el valor `k` que debe ser un dígito tradicional.

`mutateGen(r,k,delta,rMax,rMin=0)` Regresa el resultado de mutar el número `r` con valor mínimo `rMin` en el valor `k` que puede ser un dígito generalizado. La función realiza la corrección gama.

`mutValues(r,k,delta,rMin=0)` Regresa todos los resultados posibles de aplicar la función `mutate`.

`crossPoint(type,digits,dist,N,B,rMin,rMax)` Obtiene un punto de cruce dentro del vgGA.

`traditional(m)` Trunca el número `m` hacia dígitos tradicionales.

`generalized(m,B=2)` Trunca el número `m` hacia dígitos generalizados.

Falta descripción de `uniform`.

4. Estructura de la clase `population`

El vgGA está programado orientado a objetos. En el directorio raíz `vgGA1.3/`, se encuentra el directorio `@population/` que define la clase `population` que es la clase de la cual es instancia la población en el vgGA. En el directorio `@population/` se encuentran todos los métodos de esta clase.

La estructura de la clase population tiene los siguientes campos:

- `evals`
- `params`
- `best`
- `individual`

- `mutclock`
- `trace`

El campo `evals` contiene el número de evaluaciones de la función objetivo que se han realizado hasta el momento; los demás campos se describen en las siguientes secciones. En la figura 2 se lista la estructura de la clase population.

4.1. `params`

El campo `params` contiene parámetros que afecta a la operación del vgGA y se detallan a continuación.

`max` Bandera que determina si se está resolviendo un problema de maximización (1) o minimización (0).

`type` Tipo de los cromosomas puede ser `'integer'` o `'real'`.

`m` Número de segmentos en el cromosoma.

`N[m]` Arreglo que contiene el número de dígitos por segmento.

`rMin[m]` Vector que contiene los mínimos de los segmentos.

`rMax[m]` Vector que contiene los máximos de los segmentos.

`DeltaR[m]` `rMax` − `rMin`.

`pm[m]` Vector de las probabilidades de mutación por segmento.

`pc` Probabilidad de cruce.

`B[m]` Vector de bases de los segmentos.

`delta[m]` Vector de anchos de mutación por segmento.

`dist` Tipo de distribución de puntos de cruce, puede tomar los valores de `'exponential'` o `'uniform'`.

`digits` Tipo de dígitos, puede tomar los valores de `'traditional'` o `'generalized'`.

4.2. `best`

El campo `best` guarda información del mejor individuo encontrado hasta el momento (*best-found-so-far*), y tiene los siguientes campos:

`r[m]` Fenotipo del mejor individuo. Éste es un vector de reales (o posiblemente enteros) de tamaño `m`.

`fitness` Aptitud del mejor individuo encontrado.

`evals` Número de evaluaciones cuando se encontró al mejor individuo.

```
evals        : number of function evaluations so far

params
          max: 0, 1 maximization flag
         type: 'integer', 'real'
            m: number of segments
         N[m]: digits per segment
       rMin[m]: min value of segments
       rMax[m]: max value of segments
     DeltaR[m]: rMax - rMin
         pm[m]: mutation probability
            pc: crossover mutation
         B[m]: base of segments
      delta[m]: mutation delta (width)
         dist: 'exponential', 'uniform'
        digits: 'traditional', 'generalized'

best         : best individual found so far
         r[m]: phenotype
       fitness: evaluation
         evals: number of evaluation when found


individual[<population size>]
         r[m]: phenotype
       fitness: evaluation

mutclock
     DeltaI[m]: individuals to next mutation
      kPlus[m]: value where next mutation will occur
       mMax[m]: maximum digit for mutation
      mPlus[m]: value where next mutation will occur

trace
        nMuts: number of mutations
       nCross: number of crossovers
         nIPC: number of Inter Parameter (segment) Crossovers
```

Figura 2: Estructura de la clase population.

4.3. individual

El campo individual es un arreglo de tamaño que contiene todos los individuos en la población. Cada elemento del arreglo tiene los siguientes campos:

r[m] Fenotipo del individuo. Éste es un vector de reales (o posiblemente enteros) de tamaño m.

fitness Aptitud del individuo.

4.4. mutclock

El campo mutclock contiene información necesaria para implementar el reloj de mutación del vgGA.[1]

4.5. trace

El campo trace contiene información útil para rastrear y monitorear la operación del vgGA. Se tienen los siguientes campos:

nMuts Número de mutaciones que se han realizado.

nCross Número de cruces que se han llevado a cabo.

nIPC Número de cruces que han ocurrido en la frontera entre segmentos.

5. Métodos de la clase population

El vgGA está programado orientado a objetos. El directorio @population/ contiene los métodos que definen la clase population.

5.1. Métodos básicos

Los métodos básicos para la clase population son los siguientes:

population Este método construye una población vacía con los parámetros que especifican cómo va a correr el vgGA. Se puede llamar dos maneras, dependiendo de si se desea una población de enteros o reales:

```
population('i(nteger)',N,pm,pc,B,delta,dist,digits)
population('r(eal)',R,N,pm,pc,B,delta,dist,digits)
```

Donde R es una matriz $m \times 2$ que contiene los valores mínimos y máximos de cada segmento y N es un vector con el número de dígitos por segmento. Los demás parámetros se describieron en la sección 4.1, son opcionales, y toman los siguientes valores de default: pm $= 0$, pc $= 1.0$, B $= 2$, delta $=$ B, dist $=$ 'exponential', digits $=$ 'traditional'.

display Éste es un método que es necesario para que MATLAB sepa cómo desplegar una instancia de la clase population. Dentro del método display se definen las siguientes constantes que determinan la forma en que se despliega una población:

[1] Se está preparando un documento que explicará la operación del reloj de mutación del vgGA.

`CHROMS` Desplegar cromosomas. Default es 1.

`PHENOS` Desplegar fenotipos. Default es 1.

`MUT_CLOCK` Desplegar variables del reloj de mutación. Default es 0.

`TRACE` Desplegar infromación de rastreo. Default es 0.

`BEST` Desplegar mejor encontrado. Default es 1.

`setDisplay(pop,displayList)` Permite fijar algunos de las constantes que determinan la forma en que `display` despliega una población.

`get, set` En MATLAB no es posible accesar campos de un objeto fuera de los métodos de su clase. Construí los métodos `get` y `set` para obtener y asignar valores a algunos campos.

`random(pop,n,add='N')` Crea `n` individuos aleatorios. Si `add` es `'Y'` se agregan estos individuos.

`init(pop)` Inicializa una población borrando todos los individuos y borrando la información relacionada con el mejor encontrado.

`fill(pop,value=0)` Llena el campo `r` de todos los individuos de la población `pop` con el valor `value` que toma el default de 0. Junto con el método `set`, este método es útil para hacer pruebas.

`max(pop)` Carga el valor de `1` a la bandera `max` del campo `params` para que se realice maximización.

`min(pop)` Carga el valor de `0` a la bandera `max` del campo `params` para que se realice minimización.

5.2. Métodos de cruce y mutación

`crossover(pop,lastTwo='N')` Aplica cruce de un punto a la población `pop` haciendo parejas aleatorias. Si `lastTwo` es `'Y'` el cruce se realiza únicamente a los dos últimos individuos de la población.

`mutation(pop,last='N')` Aplica mutación a la población `pop`. Si `last` es `'Y'` la mutación se aplica únicamente al último individuo de la población.

`uniformCrossover(pop,p0,lastTwo=0)` Aplica cruce uniforme a la población `pop` haciendo parejas aleatorias. Cada cruce se realiza con una plantilla binaria aleatoria con probabilidad de cero de `p0`. Solamente funciona para números enteros y dígitos tradicionales.

5.3. Métodos de selección

`evaluate(pop,objective_function,start=1,setBestFlag=1)` Evalúa a la población `pop` con la función objetivo `objective_function` que espera un vector del mismo tamaño que el campo `r` de los individuos en `pop`. La población se evalúa a partir del individuo `start`. Si la bandera `setBestFlag` es verdadera, se actualiza el mejor individuo encontrado.

`evaluateLast(pop,objective_function,N=1,setBestFlag=1)` Evalúa los últimos `N` individuos de la población con la función objetivo `objective_function` que espera un vector del mismo tamaño que el campo `r` de los individuos en `pop`. Si la bandera `setBestFlag` es verdadera, se actualiza el mejor individuo encontrado.

`multiEvaluate(pop,objective_functions,start=1)` Evalúa la población `pop` con las funciones objetivo definidas en la lista `objective_functions`. La población se evalúa a partir del individuo `start`.

`eraseWeak(pop,N=1)` Borra los `N` individuos de peor evaluación en la población.

`roulette(pop,normalized=0)` Realiza selección de rueda de ruleta sobre la población `pop`.

`sus(pop,normalized=0)` Realiza muestreo universal estocático (SUS) sobre la población `pop`.

`linRanking(cMult=2)` Realiza selección por ranqueo lineal. La constante `cMult` toma el valor de `2.0` como default.

`tournament(pop,m=2)` Realiza selección de torneo de tamaño `m` sobre la población `pop`. `m` es opcional y toma el valor de `2` como default.

`btournament(pop,compare_function)` Realiza selección de torneo binario por comparación sobre la población `pop`. El usuario debe proveer la función `compare_function` que reciba el campo `r` de dos individuos y regrese `1` si el primer individuo es mejor o igual al segundo, y `0` si no.

`ngeneration` Implementa selección no generacional (*de estado estable*). No está terminado este método: No usar.

5.4. Métodos para escalamiento, compartición, y convergencia

`scale(pop,Cmult=2)` Realiza escalamiento lineal sobre la población `pop`. La constante de escalamiento `Cmult` toma el default de `2`.

`share(pop,sigma,type,alpha=1)` Realiza compartición (*sharing*) de tipo `type` con constantes `sigma` y `alpha` sobre la población `pop`. El tipo de la compartición puede ser `'phenotypic'` (fenotípica) o `'fitness'` (por aptitud).

`convergence(pop,type='phenotypic')` Calcula la diversidad de la población `pop` utilizando un tipo `type` de convergencia.

5.5. Inyecciones y herencia

Estos métodos están todavía siendo desarrollados.

`injection(pop,fraction,objective_function)` Conserva la fracción `fraction` de los mejores individuos en la población, y genera individuos aleatorios que son evaluados con `objective_function`.

`startInheritance(pop,weighted=0,alpha=0.5,beta=0.7)` Inicializa herencia. Una fracción `alpha` de la población es evaluada directamente con la función objetivo. La aptitud del resto de la población es estimada con herencia.

`crossInheritance(pop)` Realiza el cruce con herencia.

`evaluateInheritance(pop,objective_function)` Evalúa a la población con herencia.

5.6. Métodos para desplegar y ordenar

`sort(pop,type='phenotype',order='direct')` Ordena la población `pop` por tipo `type` que puede ser `'phenotype'` o `'fitness'` en orden `order` que puede ser `'direct'` (mejor a peor) o `'inverse'`.

`report(pop,fileID)` Reporta la población `pop` a la pantalla o a un archivo.

`plot(pop)` Grafica una población de un segmento. (Bajo construcción: no usar).

5.7. Reloj de mutación

El vgGA utiliza un reloj de mutación para determinar cuándo debe ocurrir la siguiente mutación.

`nextMut(pop,i)` Calcula cuándo debe ocurrir la siguiente mutación para el segmento `i` de acuerdo a un reloj de mutación.[2]

`startMut(pop)` Inicializa o reinicializa el reloj de mutación.

6. Uso del vgGA

En esta sección haremos un breve tutorial del uso del vgGA. El vgGA está programado orientado a objetos. El vgGA está basado en la clase `population` definida en el directorio `@population/`. En MATLAB todos los métodos de una clase deben colocarse en un directorio que empiece con el caracter @ (arroba) y que tenga el mismo nombre de la clase.

El constructor de la clase debe ser una función con el mismo nombre de la clase, y que regrese una instancia de la clase que se contruya con la función `class`. La forma en que se despliega una instancia de una clase queda determinada por el método `display` que el usuario debe definir.

MATLAB no permite accesar los campos de un objeto fuera de los métodos de ese objeto. Es usual contruir métodos específicos para obtener y modificar campos de una clase. La clase `population` tiene los métodos `get` y `set` para este fin; sin embargo, están limitados a los campos que se requirieron para pruebas, y no son muy generales.

En MATLAB, a menos que se declaren globales, todas las variables son locales a las funciones que las utilizan. Los métodos de una clase son simplemente funciones que deben recibir la instancia de la clase sobre la que van a trabajar, y deben regresarla para su asignación si es que deben modificar la instancia.

6.1. Creación de una población vacía

Para crear un objeto `p` de la clase population se utiliza el constructor de la clase. Por ejemplo, para crear una población de enteros, con dos segmentos (uno de 6 dígitos binarios, otro de 3 dígitos base 5), con probabilidades de mutación de 0.4 y 0.2, y probablidad de cruce 0.9, se utiliza la siguiente instrucción:

```
>> p = population('integer',[6 3],[0.4 0.2],0.9,[2 5])

p =
```

[2] Se está preparando un documento que explica la operación del reloj de mutación del vgGA.

```
   vgGA population:

   parameters:

    type: integer
      pm: 0.4 0.2
      pc: 0.9
       m: 2
       B: 2 5
   delta: 2 5
    dist: exponential
  digits: traditional
       N:   6   3

   Best (max) found at evaluation 0, after 0 evaluations
```

Esta instrucción crea una población vacía (sin individuos).

6.2. Individuos aleatorios

Para crear 10 individuos aleatorios se utiliza el método `random` como se muestra a continuación:

```
>> p = random(p,10)
p =

   vgGA population:

   parameters:

    type: integer
      pm: 0.4 0.2
      pc: 0.9
       m: 2
       B: 2 5
   delta: 2 5
    dist: exponential
  digits: traditional
       N:   6   3

   individual:

       1: 011011 232      27      67  ->         NaN
       2: 101100 000      44       0  ->         NaN
       3: 110101 421      53     111  ->         NaN
       4: 000001 022       1      12  ->         NaN
       5: 010111 440      23     120  ->         NaN
       6: 100111 313      39      83  ->         NaN
       7: 011101 404      29     104  ->         NaN
```

```
      8: 110100 232      52      67  ->         NaN
      9: 000101 121       5      36  ->         NaN
     10: 100110 042      38      22  ->         NaN

   Average fitness:       NaN,   std dev:       NaN
   Best (max) found at evaluation 0, after 0 evaluations:
    BEST: 000000 000     0     0  ->        -Inf
```

Para cada individuo se lista su genotipo, fenotipo, y evaluación. Por ejemplo, el individuo 3 tiene un cromosoma de `110101 421`, un fenotipo de `[53 111]`, y una evaluación de `NaN` (*Not a Number*) porque todavía no ha sido evaluado.

6.3. Evaluación

La población se evalúa con el método `evaluate` que recibe la función objetivo como parámetro:

```
>> p = evaluate(p,@x2y2)
p =

   vgGA population:

   parameters:

    type: integer
      pm: 0.4 0.2
      pc: 0.9
       m: 2
       B: 2 5
   delta: 2 5
    dist: exponential
  digits: traditional
       N:   6   3

   individual:

      1: 011011 232      27      67  ->   5218.0000
      2: 101100 000      44       0  ->   1936.0000
      3: 110101 421      53     111  ->  15130.0000
      4: 000001 022       1      12  ->    145.0000
      5: 010111 440      23     120  ->  14929.0000
      6: 100111 313      39      83  ->   8410.0000
      7: 011101 404      29     104  ->  11657.0000
      8: 110100 232      52      67  ->   7193.0000
      9: 000101 121       5      36  ->   1321.0000
     10: 100110 042      38      22  ->   1928.0000

   Average fitness: 6786.7000,   std dev: 5637.2850
   Best (max) found at evaluation 3, after 10 evaluations:
    BEST: 110101 421    53   111  ->  15130.0000
```

El individuo 3 tiene ahora una evaluación de 15130.0000. El método `display` despliega información general de la población: aptitud promedio, desviación estándar de la aptitud, mejor encontrado, etc. También se despliega cuántas evaluaciones se han realizado, en cuál evaluación se encontró el `mejor encontrado`, y el mejor encontrado. En este ejemplo la población se evalúa con la función `x2y2` que es una función que recide en el directorio raíz `vgGA/`, y que regresa la suma de los cuadrados de los segmentos. Para el individuo 3 se tiene que la evaluación es $53^2 + 111^2 = 15130$. El método `evaluate` espera una función que reciba un vector que corresponde al fenotipo (campo `r`) de cada individuo, y regresa su evaluación.

6.4. Selección

Para realizar selección de rueda de ruleta sobre la población se aplica el método `roulette` (hay otros métodos de selección implementados en el vgGA, véase su descripción en la sección 5.3):

```
>> p = roulette(p)
p =

   vgGA population:

   parameters:

    type: integer
      pm: 0.4 0.2
      pc: 0.9
       m: 2
       B: 2 5
   delta: 2 5
    dist: exponential
  digits: traditional
       N:   6   3

   individual:

       1: 011101 404      29     104  ->  11657.0000     1.7176
       2: 010111 440      23     120  ->  14929.0000     2.1997
       3: 110101 421      53     111  ->  15130.0000     2.2294
       4: 110100 232      52      67  ->   7193.0000     1.0599
       5: 110101 421      53     111  ->  15130.0000     2.2294
       6: 011011 232      27      67  ->   5218.0000     0.7689
       7: 011101 404      29     104  ->  11657.0000     1.7176
       8: 010111 440      23     120  ->  14929.0000     2.1997
       9: 010111 440      23     120  ->  14929.0000     2.1997
      10: 110100 232      52      67  ->   7193.0000     1.0599

   Average fitness:11796.5000,   std dev: 3906.7196
   Best (max) found at evaluation 3, after 10 evaluations:
    BEST: 110101 421    53   111  ->  15130.0000
```

Los métodos `sus` y `tournament` implementan muestreo universal estocástico y selección de torneo, respectivamente.

6.5. Cruce y mutación

Cruce y mutación se realizan con los métodos `crossover` y `mutation`:

```
>> p = crossover(p)
p =

   vgGA population:

   parameters:

    type: integer
      pm: 0.4 0.2
      pc: 0.9
       m: 2
       B: 2 5
   delta: 2 5
    dist: exponential
  digits: traditional
       N:   6   3

   individual:

       1: 011101 404      29     104  ->  11657.0000     1.7176
       2: 010111 440      23     120  ->  14929.0000     2.1997
       3: 110101 421      53     111  ->  15130.0000     2.2294
       4: 110100 232      52      67  ->   7193.0000     1.0599
       5: 110101 421      53     111  ->  15130.0000     2.2294
       6: 011011 232      27      67  ->   5218.0000     0.7689
       7: 011101 404      29     104  ->  11657.0000     1.7176
       8: 010111 440      23     120  ->  14929.0000     2.1997
       9: 110100 230      52      65  ->         NaN        NaN
      10: 010111 442      23     122  ->         NaN        NaN

   Average fitness:       NaN,   std dev:       NaN
   Best (max) found at evaluation 3, after 10 evaluations:
    BEST: 110101 421    53   111  ->  15130.0000

>> p = mutation(p)
p =

   vgGA population:

   parameters:
```

```
    type: integer
      pm: 0.4 0.2
      pc: 0.9
       m: 2
       B: 2 5
   delta: 2 5
    dist: exponential
  digits: traditional
       N:   6   3

   individual:

       1: 010101 404      21     104  ->         NaN        NaN
       2: 010111 440      23     120  ->  14929.0000     2.1997
       3: 110111 421      55     111  ->         NaN        NaN
       4: 111100 232      60      67  ->         NaN        NaN
       5: 010101 021      21      11  ->         NaN        NaN
       6: 010010 432      18     117  ->         NaN        NaN
       7: 110001 424      49     114  ->         NaN        NaN
       8: 010111 440      23     120  ->  14929.0000     2.1997
       9: 110100 230      52      65  ->         NaN        NaN
      10: 010111 442      23     122  ->         NaN        NaN

   Average fitness:       NaN,   std dev:       NaN
   Best (max) found at evaluation 3, after 10 evaluations:
    BEST: 110101 421    53   111  ->  15130.0000
```

Nótese que la mayoría de los individuos tienen una evaluación de `NaN` porque no han sido evaluados. El individuo 2 tiene la evaluación de 1429.000 porque no fue modificado por cruce ni mutación.

6.6. Algoritmo genético

Con los métodos `population`, `random`, `evaluate`, `roulette`, `crossover`, y `mutation` es posible crear un algoritmo genético que corra por un número de generaciones y que regrese el individuo `mejor encontrado` contra el número de evaluaciones de la generación. El método `runGA` hace lo anterior. En las siguientes instrucciones se corre la población `p` por 50 generaciones y se grafica el mejor encontrado contra el número de evalauciones de la función objetivo:

```
>> [p,B] = runGA(p,@x2y2,50);
   10     3    53   111  ->  15130.0000
   20    14    19   122  ->  15245.0000
   30    29    55   124  ->  18401.0000
   40    29    55   124  ->  18401.0000
   50    29    55   124  ->  18401.0000
   60    29    55   124  ->  18401.0000
   70    70    63   123  ->  19098.0000
   80    70    63   123  ->  19098.0000
   90    70    63   123  ->  19098.0000
```

```
  100    70    63   123  ->  19098.0000
  110    70    63   123  ->  19098.0000
  120    70    63   123  ->  19098.0000
  130    70    63   123  ->  19098.0000
  140    70    63   123  ->  19098.0000
  150    70    63   123  ->  19098.0000
  160    70    63   123  ->  19098.0000
  170    70    63   123  ->  19098.0000
  180    70    63   123  ->  19098.0000
  190    70    63   123  ->  19098.0000
  200    70    63   123  ->  19098.0000
  210    70    63   123  ->  19098.0000
  220    70    63   123  ->  19098.0000
  230    70    63   123  ->  19098.0000
  240    70    63   123  ->  19098.0000
  250    70    63   123  ->  19098.0000
  260    70    63   123  ->  19098.0000
  270    70    63   123  ->  19098.0000
  280   276    63   124  ->  19345.0000
  290   276    63   124  ->  19345.0000
  300   276    63   124  ->  19345.0000
  310   276    63   124  ->  19345.0000
  320   276    63   124  ->  19345.0000
  330   276    63   124  ->  19345.0000
  340   276    63   124  ->  19345.0000
  350   276    63   124  ->  19345.0000
  360   276    63   124  ->  19345.0000
  370   276    63   124  ->  19345.0000
  380   276    63   124  ->  19345.0000
  390   276    63   124  ->  19345.0000
  400   276    63   124  ->  19345.0000
  410   276    63   124  ->  19345.0000
  420   276    63   124  ->  19345.0000
  430   276    63   124  ->  19345.0000
  440   276    63   124  ->  19345.0000
  450   276    63   124  ->  19345.0000
  460   276    63   124  ->  19345.0000
  470   276    63   124  ->  19345.0000
  480   276    63   124  ->  19345.0000
  490   276    63   124  ->  19345.0000
  500   276    63   124  ->  19345.0000
  510   276    63   124  ->  19345.0000
>> plot(B(:,1),B(:,2))
```

La última instrucción produce la gráfica de la figura 3.

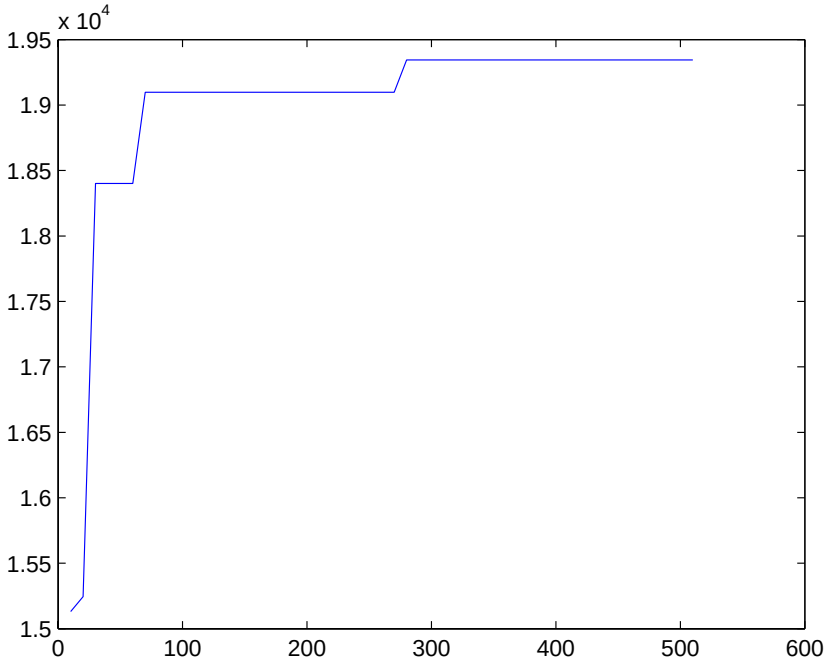

Figura 3: Gráfica del mejor individuo encontrado contra el número de evaluaciones de la función objetivo para el ejemplo del tutorial.

6.7. Curva del mejor encontrado

Para obtener la *curva del mejor encontrado* existe la función `plotGA` en el directorio raíz `vgGA`. Esta función crea una población y llama repetidamente a `runGA` para luego graficar el promedio, más y menos la desviación estándar, del mejor encontrado de una serie de corridas de un algoritmo genético. Con la siguiente instrucción se corre el algoritmo genético de este tutorial 20 veces por 50 generaciones.

```
plotGA(20,50,10,@x2y2,'random','integer',[6 3],[0.4 0.2],0.9,[2 5]);
```

La figura 4 muestra la gráfica de la *curva del mejor encontrado* que se produce.

6.8. Funciones objetivo implementadas

En el directorio raíz se encuentran implementaciones de la siguientes funciones objetivo:

`debf1` Implementa la función 1 de Deb para probar compartición *sharing)*.

`debf2` Implementa la función 2 de Deb para probar compartición *sharing)*.

`onemax` Regresa el número de unos en el cromosoma.

`basemax(x,B=2)` Para un cromosoma de base `B`, regresa el número de dígitos igual al dígito $B-1$.

`x` Regresa la suma de los segmentos del cromosoma.

`x2y2` Regresa la suma de los cuadrados de los segmentos.

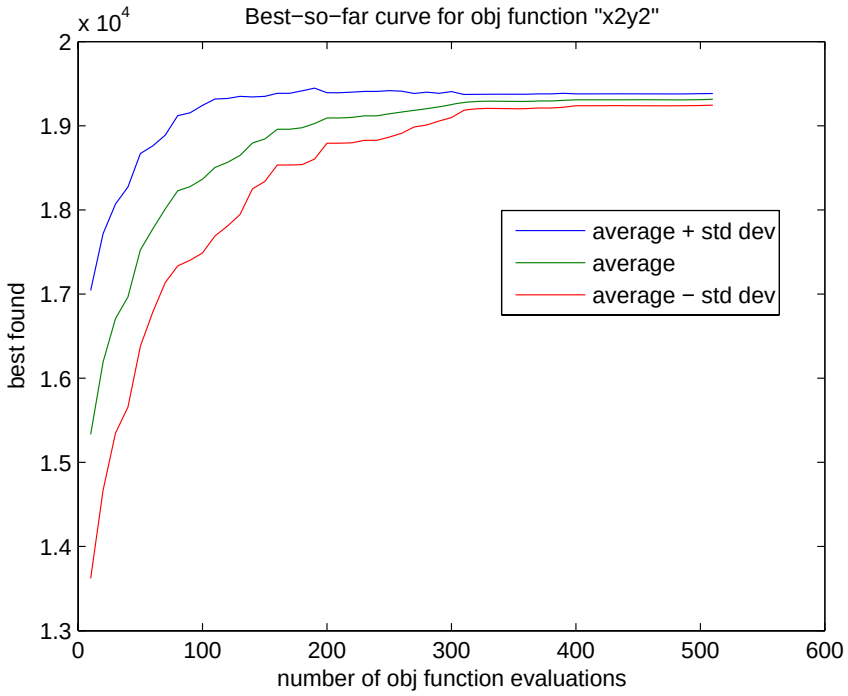

Figura 4: Gráfica de la curva del mejor encontrado para 20 corridas del algoritmo genético del tutorial.

7. Uso del vgGA mediante scripts

En lugar de utilizar el método `runGA` y la función `plotGA` es posible, y quizás más conveniente, hacer scripts que llamen a los métodos que se desean aplicar. En el directorio `Documentation/` se encuentrar los scripts siguientes como ejemplos de diferentes tipos de algoritmos:

`AGsimple` Se corre un algoritmo genético simple con selección de torneo sobre la función `basemax` y se obtiene la curva de mejor encontrado.

`Escalamiento` Se demuestra la necesidad de utilizar escalamiento para evitar falta de presión selectiva al final de la corrida.

`Inyecciones` Muestra el uso de los métodos `convergence` e `injection` para reinicializar un algoritmo genético mediante inyecciones de diversidad.

`Herencia` Se muestra la forma de utilizar herencia junto con inyecciones para disminuir el número de evaluaciones de la función objetivo.

`Comparticion` Se hace una breve demostración de compartición.

`AGnogeneracional` Se implementa un algoritmo genético de estado estable, y se compara su desempeño sobre la función `onemax`.

`Engano` Este script utiliza las funciones `deceptive` e `interDeceptive` para probar un algoritmo genético simple sobre problemas completamente engañosos concatenados y con los bits intercalados.

8. Optimización de permutaciones

El directorio `TSP/` contiene la implmentación de operadores especiales de cruce y funciones para permutaciones.

`fcnTSP` Calcula el costo (distancia) total de una ruta.

`distance=tableTSP(coordinates)` Obtiene la matriz de distancias (costos) a partir de las coordenadas de las ciudades.

`plotTSP` Grafica una ruta de un TSP.

`[v,n1,n2]=invTSP(permutation)` Realiza una inversión aleatoria en una permutación. Regresa el resultado de la inversión `v`, y las posiciones `n1` y `n2` sobre las que se hizo la inversión.

`[h1,h2]=pmx(p1,p2)` Implementa cruce PMX de las permutaciones `p1` y `p2`.

`[h1,h2]=cx(p1,p2)` Implementa cruce CX de las permutaciones `p1` y `p2`.

`[h1,h2]=ox(p1,p2)` Implementa cruce OX de las permutaciones `p1` y `p2`.

`h=erx(p1,p2)` Implementa cruce ERX de las permutaciones `p1` y `p2`.

`[h1 h2]=erx2(p1,p2)` Manda llamar dos veces a la función `erx` para producir dos hijos de las permutaciones `p1` y `p2`.

`ordonez(chromosome)` Toma un cromosoma en codificación de Ordóñez[3] y regresa la permutación representada.

`fcnOrdonez(chromosome)` Evalúa un cromosoma que representa una permutación utilizando la representación de Ordóñez; para ello, llama a la función `fcnTSP`.

Se tienen los siguiente métodos para optimización de permutaciones dentro del directorio `vgGA1.3/`:

`randomPerm(pop,n)` Genera una población aleatoria donde los individuos son permutaciones. Los elementos de la permutación están numerados a partir de `0`.

`permCrossover(pop,@xoperator)` Aplica el operador de permutaciones `xoperator` a la población.

Dentro del directorio `Documentation/` Se encuentra el script `Permutaciones` que muestra el uso de las funciones y métodos anteriores para resolver problemas de vendedor viajero dentro del vgGA.

9. Gene Expression Programming

Gene Expression Programming es una forma de programación genética en la cual

[3] La codificación de Ordóñez representa una permutación como un vector binario. Cuando se utiliza la codificación de Ordóñez, cruce y mutación tradicionales no producen individuos inválidos.

10. Scripts de aspectos teóricos

Los siguientes scripts sirven para mostrar o probar algunos aspectos teóricos del vgGA.

`Cruces-Continuos` Se muestra el resultado del cruce de dos números reales, 0.8 y 0.5, para todos los puntos válidos en $[0, 1)$. Se muestran los resultados con dígitos tradicionales, generalizados, y continuos; con distribución exponencial y uniforme; sin y con corrección gamma.

`reloj` Demuestra las fórmulas que utiliza el reloj de mutación. Se compara la densidad de probablidad que se obtiene cuando se hace mutación en la forma tradicional (decidiendo si mutar bit por bit) con la densidad de probabilidad que producen las fórmulas del reloj de mutación.

Referencias

Valenzuela-Rendón, M. (2003). The virtual gene genetic algorithm. *Genetic and Evolutionary Computation Conference (GECCO-2003)*, 1457–1468.

Índice de métodos, funciones, y campos