Optimización usando heurísticas

Dr. Luis Gerardo de la Fraga

E-mail: fraga@cs.cinvestav.mx
Departamento de Computación
Cinvestav Zacatenco

27 de enero, 2021

Contenido

1. ¿Qué es optimización?
2. Problemas de optimización
3. Heurísticas: búsqueda exhaustiva, búsqueda aleatoria
4. Algoritmo genéticos
5. Optimización multiobjetivo
6. El algoritmo de la heurística NSGA-II

Optimización

- Es encontrar la mejor solución a un problema dado
- Miminizar una función f es igual a maximizar $-f$

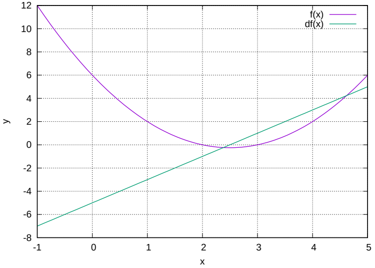
Tipos de problemas de optimización

1. **Problemas lineales**. Se resuelven invirtiendo una matriz (o mejor usando la descomposición QR)
2. **Problemas no lineales**. Se resuelven iterativamente, se necesita una solución inicial muy cercana a la solución final para que el algoritmo de solución converja.
3. **Problemas uniobjetivo** (con n dimensiones).
4. **Problemas multiobjetivo**, estos tienen 2 o 3 objetivos y n dimensiones.
5. Con más de 3 objetivos son **problemas de muchos objetivos**

1. Los problemas también pueden ser:
2. Continuos (relativamente son fáciles de resolver)
3. Discretos (pueden ser imposibles de resolver)

¡Un problema!

- Minimizar:
  $f(x) = (x-2)(x-3)$
- Solución:
- Expandemos
  $f(x) = x^2 - 5x + 6.$
- Calculamos su derivada:
  $f'(x) = 2x - 5.$
- La igualamos a cero:
  $2x - 5 = 0,\ x = 5/2$



- En más dimensiones este problema se resuelve con una inversión de una matriz (o mejor usando la descomposición QR)
- $A\mathbf{x} = \mathbf{y}$, A es una matriz, $\mathbf{x}$ y $\mathbf{y}$ son vectores columna,
- $\mathbf{x} = A^{-1}\mathbf{y}$.
- Usando la descomposición QR:
- $A\mathbf{x} = QR\mathbf{x} = \mathbf{y}$,
- $Q^{\mathsf{T}}QR\mathbf{x} = Q^{\mathsf{T}}\mathbf{y}$,
- $R\mathbf{x} = Q^{\mathsf{T}}\mathbf{y}$.

Un problema no lineal

Encontrar los valores para x y y que minimizen:

$$f(x,y) = (a-x)^2 + b(y-x^2)^2$$

- Se le conoce como la función de Rosenbrock
- Es no lineal y tiene dos variables (dimensión 2)
- Se tiene que linealizar su derivada, usando su descomposición en series de Taylor alrededor de un punto inicial (¡Es por esto que se necesita una solución inicial!)
- Se iguala a cero.
- Se itera hasta obtener convergencia.
- Se debe poner en el código un número máximo de iteraciones para evitar que se cicle el programa, y no termine, si no existe convergencia.

- $f(x) = \sum_{i=0}^{n} \frac{(x-a)^i f^{(i)}(a)}{i!}$, la serie de Taylor para f.
- $f(x) \approx f(a) + (x-a)f'(a)$, la linealización de f.
- Para la derivada:
- $f'(\mathbf{x}) \approx f'(a) + f''(a)\mathbf{\Delta a}$.
- Igualándola a cero:
- $f'(\mathbf{x}) \approx f'(a) + f''(a)\mathbf{\Delta a} = 0$,
- $\mathbf{\Delta a} = -f'(a)/f''(a)$.
- En más dimensiones:
- $\mathbf{\Delta a} = -H^{-1}(\mathbf{a})J(\mathbf{a})$.

Para la función de Rosenbrock: $f(x,y) = (a-x)^2 + b(y-x^2)^2$

- $\frac{\partial f}{\partial x} = 2(x-a) - 4bx(y-x^2)$
- $\frac{\partial f}{\partial y} = 2b(y-x^2)$
- $\frac{\partial^2 f}{\partial x \partial x} = \frac{\partial f}{\partial x}(2x - 4bxy + 4bx^3)$
- $\quad = 2 - 4by + 12bx^2$
- $\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial f}{\partial x}(-2bx^2) = -4bx$
- $\frac{\partial^2 f}{\partial y \partial x} = \frac{\partial f}{\partial y}(-2bx^2) = -4bx$
- $\frac{\partial^2 f}{\partial y \partial y} = \frac{\partial f}{\partial y} 2by = 2b$

La jacobiana J se calcula como:

$$J = \begin{bmatrix} \frac{\partial f'}{\partial x} \\ \frac{\partial f'}{\partial y} \end{bmatrix}$$

La matriz hesiana se calcula como:

$$H = \begin{bmatrix} \frac{\partial^2 f'}{\partial x \partial x} & \frac{\partial^2 f'}{\partial x \partial y} \\ \frac{\partial^2 f'}{\partial y \partial x} & \frac{\partial^2 f'}{\partial y \partial y} \end{bmatrix}$$

Método de Newton

- Entrada: valor inicial $\mathbf{a}_0$, valor de la precisión ϵ, J, H
- Salida: el mejor valor que optimize f
- for(i=0; i<20; i++) {
- $\quad H\mathbf{\Delta a} = J$
- $\quad \mathbf{a}_1 \leftarrow \mathbf{a}_0 - \mathbf{\Delta a}$
- $\quad$ if ($\|\mathbf{a}_1 - \mathbf{a}_0\| < \epsilon$)
- $\qquad$ break
- $\quad \mathbf{a}_0 \leftarrow \mathbf{a}_1$
- }
- print $\mathbf{a}_1$

```
$ python newton2D.py -0.5 0.5
0 -0.5 0.5
1 -0.530612244898 0.280612244898
2 0.758409151103 -1.08639171887
3 0.759133963011 0.576283848444
4 0.999974694753 0.941945132081
5 0.999976702971 0.99995340648
6 1.0 0.999999999457
7 1.0 1.0
```

- El método de Newton tiene una convergencia cuadrática.
- O dicho de otra forma, el método de Newton obtiene el resultado en el menor número de iteraciones.
- ¿Qué valor para ϵ usarían?

El método de descenso de gradiente (1/2)

- Una buena dirección de búsqueda del valor mínimo de una función es hacia el negativo de su derivada
- $\boldsymbol{\Delta}\mathbf{a} = -\alpha J$
- α se podría calcular[1] como $\alpha = \frac{\|\mathbf{a}_{i+1} - \mathbf{a}_i\|}{\|\mathbf{J}_{i+1} - \mathbf{J}_i\|}$
- Las iteraciones pueden terminar cuando $\|\mathbf{J}_{i+1}\| < \epsilon$ (la derivada se hace cero en el mínimo o máximo local)
- Necesitamos dos puntos de entrada para el algoritmo.

[1] J. Barzilai and J. Borwein, Two-Point Step Size Gradient Methods, IMA Journal of Numerical Analysis (1988) 8, 141-148

El método de descenso de gradiente (2/2)

```
$ python descensoG.py -0.5 0.5 -0.49 0.49
0 -0.5 0.5
0 -0.49 0.49
1 -0.0391998336615 0.979799921601
2 -0.0728633853724 0.483117124935
3 -0.131365985624 0.0085392257564
4 -0.118145788989 0.0170110064419
5 -0.10818927257 0.0141054050777
.
.
.
53 0.999994802412 0.999989591696
54 0.999994807552 0.99998959433
55 0.999994809615 0.999989598487
```

¿Cuándo usar una heurística?

- Cuando no se tiene la expresión matemática (se necesitan las expresiones para la primera y segunda derivadas para un Newton, o la primera derivada para el descenso de gradiente)
- Cuando el valor de la función objetivo es el resultado de un experimento (en nuestro caso podría ser una simulación con spice)
- Cuando tenemos una función multimodal

Dos heurísticas

1. La búsqueda exhaustiva
2. La búsqueda aleatoria

La búsqueda exhaustiva (1/3)

- Supongamos que queremos resolver un problema con 10 variables
- Supongamos que el espacio de búsqueda para cada variable es $[-10, 10]$
- Supongamos que buscamos la solución al problema de optimización con una resolución de $0.001 = 10^{-3}$
- Para una variable tenemos que buscar $20/10^{-3} = 20 \times 10^3$ valores de la función objetivo
- Para las 10 variables son $(20 \times 10^3)^{10}$ valores, igual a
- $20^{10} \times 10^{30} =$
- $= 2^{10} \times 10^{10} \times 10^{30} = 1024 \times 10^{40}$

La búsqueda exhaustiva (2/3)

- Si suponemos que se obtiene el valor de la función objetivo en 1 segundo,
- entonces tardaríamos en resolver el problema:
- 1024×10^{40} seg $\times \frac{1 \text{ min}}{60 \text{ seg}} \times \frac{1 \text{ hr}}{60 \text{ min}} \times \frac{1 \text{ día}}{24 \text{ hr}} \times \frac{1 \text{ año}}{365 \text{ días}}$
- $= \frac{1024 \times 10^5 \times 10^{35}}{31{,}536{,}000}$ años
- $\approx 3 \times 10^{35}$ años

La búsqueda exhaustiva (3/3)

- Esta heurística nos garantiza encontrar la solución global (dentro de la resolución de la búsqueda).
- Se puede usar hasta en 3 dimensiones.

La búsqueda aleatoria (1/2)

- Entrada: La amplitud de búsqueda $[\text{mín}_i, \text{máx}_i]$ para cada variable a_i
- Salida: La mejor solución encontrada aleatoriamente

```
vmin = 1e6
i = 0
while i < iteraciones :
    Para cada variable :
      a_i = rand() * (máx_i − mín_i) + mín_i
   v = f(a)
   if v < vmin :
      vmin = v
      solucion = a
   i += 1
print vmin, solucion
```

La búsqueda aleatoria (2/2)

- Es totalmente ineficiente
- Nunca debería de usarse

Algoritmo genético (1/2)

- Utiliza el conocimiento almacenado en un conjunto (población) de soluciones
- Es una mejor aproximación que la búsqueda aleatoria

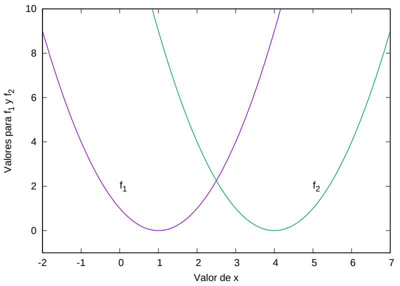
Algoritmo genético (2/2)

- Entrada: El número de iteraciones, el tamaño de la población, el espacio de búsqueda para cada variable
- Salida: La mejor solución que esperamos esté cerca del mínimo global

```
1  Inicializa aleatoriamente la población
2  i = 0
3  while i < iteraciones :
4      Para toda la población/2 :
5          Seleccionar dos hijos
7          Cruzar los hijos
8          Mutar los hijos
8          Aplicar elitismo
9      La población generada reemplaza a la anterior
10     i += 1
```

Optimización multiobjetivo

- Resuelve el problema de optimizar dos o tres funciones al mismo tiempo.
- Se tiene un conjunto de soluciones
- Se podría resolver el problema agregando las funciones como:
- $f = w_1 f_1 + w_2 f_2 + w_3 f_3$, donde $w_1 + w_2 + w_3 = 1$

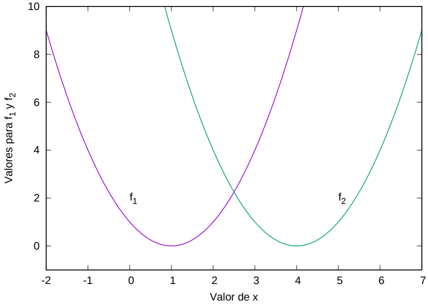


Problema de optimización multiobjetivo (POMO)

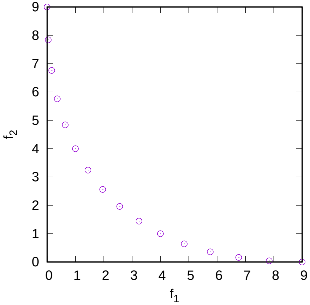
Un POMO es un problema que considera más de una función objetivo (y que están en conflicto):

$$\text{POMO} \begin{cases} \text{Optimizar } \ \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})]^{\mathsf{T}}, \\ \text{sujeto a: } \ h_k(\mathbf{x}) \geq 0, \ \text{ para } \ k = 1, 2, \ldots p, \\ \text{con } \ \mathbf{x} \in D \subset \mathbb{R}^n, \end{cases}$$

donde m es el número de funciones objetivo a optimizar, $\mathbf{x} = [x_1, x_2, \ldots, x_n]^{\mathsf{T}}$ es el conjunto de variables de decisión en el conjunto de soluciones factibles D; h_k son p restricciones que el problema debe cumplir.

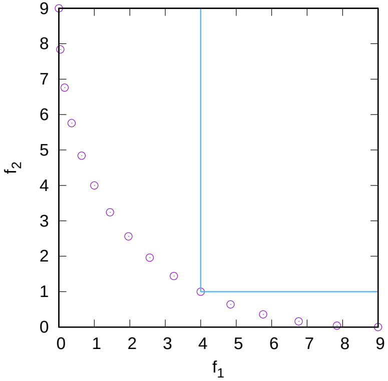


Frente de Pareto:



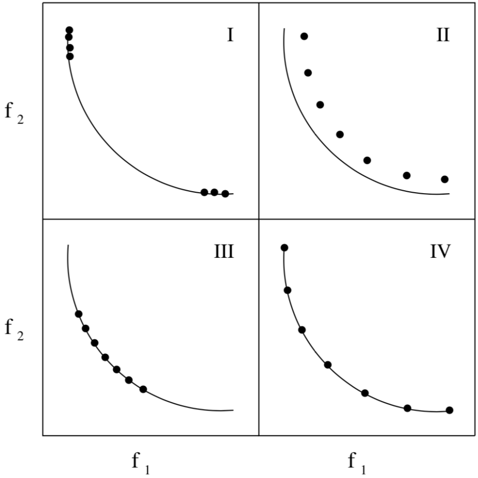
¿Cómo se comparar dos soluciones de un POMO?

- Se tienen dos soluciones $\mathbf{x}$ y $\mathbf{y}$
- Se dice que $\mathbf{x}$ *domina a* $\mathbf{y}$, $\mathbf{x} \prec \mathbf{y}$, si
- $f_i(\mathbf{x}) \leq f_i(\mathbf{y})$, para todos los índices $i \in 1, 2, \ldots, m$, y
- $f_i(\mathbf{x}) < f_i(\mathbf{y})$, para al menos uno de los índices $i \in 1, 2, \ldots, m$.



¿Cómo son las soluciones $[4.0, 1.0]^T$ y $[4.84, 0.64]^T$?

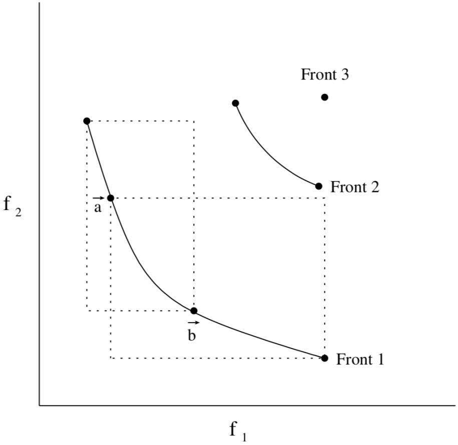
¿Qué queremos de un buen algoritmo que resuelva un POMO?



Queremos tanto una buena convergencia como una buena distribución.

- A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II
- Presentado por Deb[2] en el 2002.
- La población se particiona en capas o frentes usando el criterio de nodominación
- La diversidad se mantiene usando la distancia de apiñamiento

[2] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. IEEE Transactions on Evolutionary Computation, 6(2):182–197, April 2002.



La solución $\vec{b}$ tiene una mayor distancia de apinamiento que la de la solución $\vec{a}$

El algoritmo NSGA-II

Require: POMO, condición de terminación, tamaño de la población N,
Ensure: Una aproximación al frente de Pareto

$t \leftarrow 0$
Initializar la población P_t
Evaluar la población P_t
Obtener $\vec{f}^{\text{mín}}$ y $\vec{f}^{\text{máx}}$
ordenamiento-nodominados(P_t)
while no se cumpla la condición de terminación **do**
    Selección por torneo binario
    Generar los hijos P'_t usando operadores de variación
    Evaluar la población P'_t
    Actualizar $\vec{f}^{\text{mín}}$ y $\vec{f}^{\text{máx}}$
    $F \leftarrow$ ordenamiento-nodominados$(P_t \cup P'_t)$
    $P_{t+1} \leftarrow \varnothing$
    $i \leftarrow 1$
    while $|P_{t+1}| + |F_i| \leq N$ **do**
        distancia-apiñamiento$(F_i,\ \vec{f}^{\text{mín}},\ \vec{f}^{\text{máx}})$
        $P_{t+1} \leftarrow P_{t+1} \cup F_i$
        $i \leftarrow i + 1$
    end while
    Ordena los individuos del frente F_i según su distancia de apiñamiento
    $P_{t+1} \leftarrow P_{t+1} \cup F_i[1 : (N - |P_{t+1}|)]$
    $t \leftarrow t + 1$
end while

- El código fuente en C de NSGA-II está disponible en: `http://www.iitk.ac.in/kangal/codes.shtml`

¡Gracias!