Proyecto Final

Algoritmos de Computación Evolutiva

Computación Concurrente

I.I.M.A.S.

U.N.A.M

Barajas Cervantes Alfonso
Cabello Figueroa Israel
Cerritos Lira Carlos
Franco López Benito Vicente

Dr. Óscar Alejandro Esquivel Flores

22 de enero del 2021

1. Requerimientos

2. Organizacion

Investigar algoritmos evolutivos aunque sea para una solución de un problema sencillo Propuesta para la primera Fase de los Rquerimientos:
1.-Seleccionar 5-10 ejemplos concretos que puedamos seguir
2.-Elegir alguno de ellos y entenderlo bien
3.-Trabajar sobre un colab y decidir que parte del programa podria empezar a desarrollar cada quien
4.-En paralelo ir llenando, la introducción y la explicación de nuestro problema
5.- Llenar las conclusiónes de nuestro proyecto

3. Requerimientos

1. Realizar la investigación correspondiente a los algoritmos evolutivos y su aplicación
2. Evaluar,elegir y analizar un algoritmo de interes que pueda ser ejecutado de manera concurrente
3. Considerar un problema de aplicación del algoritmo evolutivo elegido en el punto anterión
4. Plantear el problema y hacer el analisis de la implementación concurrente
5. Elaborar el análisis correspondiente de la implementación considerando
    - Validación de resultados
    - Tiempo de ejecución de la implementación secuencial
    - Tiempo de implementación de la ejecución concurrente

4. Introducción

Marco Histórico

En 1859, Darwin publica su libro El origen de las especies que levantó agrias polémica en el mundo científico por las revolucionarias teorías que sostenían que las especies evolucionan acorde al medio, para adaptarse a éste. De esta manera el universo pasaba de ser una creación de Dios estática y perfecta y se planteaba como un conjunto de individuos en constante competición y evolución para poder perpetuar su especie en el tiempo. Las especies se crean, evolucionan y desaparecen si no se adaptan de forma que solo los mejores, los más aptos, los que mejor se adapten al medio sobreviven para perpetuar sus aptitudes. De acuerdo con esta visión de la evolución, la computación ve en dicho marco un claro proceso de optimización: se toman los individuos mejores adaptados –mejores soluciones temporales –, se cruzan –mezclan–, generando nuevos individuos –nuevas soluciones– que contendrán parte del código genético –información– de sus antecesores, y el promedio de adaptación de toda la población se mejora.

La computación evolutiva se ha convertido en un concepto general adaptable para resolución de problemas, en especial problemas difíciles de optimización, ella encuentra fuerza combinada con otras técnicas como lo es la computación concurrente.

Algoritmos Evolutivos, Computación Evolutiva y más

La rama de la computación evolutiva es en si una rama que envuelve una gran comunidad de gente, ideas y de aplicaciones. Aunque sus raices genealogicas podemos tenerlos desde 1930, surgió como la emergencia de una relativa tecnologia computacional digital no costosa en la década de los 60, que sirvió como un importante catalizador para la rama. La viabilidad de usar esta tecnología fue de gran importancia para realizar simulaciones como herramiento para analizar sistemas mucho más complejos que aquellos que se podrían analizar matemáticamente.

Los algoritmos genéticos son herramientas que podemos usar aplicando aprendizaje máquina que algunas veces nos permite encontrar soluciones para problemas que podrían tener un número muy grande de potenciales soluciones. Cuando resolvemos un problema con un algoritmo genético en lugar de emplear una solución específica, necesitamos dar carácterísitcas y reglas a la solución para que estás sean aceptadas. Cuando estamos viendo por ejemplo si pensamos en la tarea de llenar un camión de carga, debemos establecer principios y reglas que restringan los posibles ordenes en que acomodemos las cosas, por ejemplo una primera regla podría ser meter las cosas más pesadas primero, las cosas frágiles al final, para colocarlas en sitios donde no se golpeen. Estás reglas restrigen el espacio de posibles soluciones.

Ahora pensemos que estamos resolviendo un problema en particular de adivinar un numero, entre 1 y 1000, y tenemos solo diez opciones para adivinar. Si la única retroalimentación que tienes del número, es si escogiste correctamente, o incorrectamente, entonces tu mejor oportunidad es de 1 en 100 de adivinar el número. Con los algoritmos genéticos recibimos información adicional, dan una retroalimentación acerca de que tan cerca o lejos están de la solución. Si en lugar de correcto o incorrecto recibimos este indicador, la probabilidad de encontrarlo es total, porque con búsqueda binaria podemos encontrar cualquier número entre 1 y 1000 en diez pasos.

4.1. Clasificación de los algoritmos evolutivos

1. Estrategias Evolutivas: Fue diseñada inicialmente con la meta de resolver problemas de optimización discretos y continuos, principalmente experimentales y considerados difíciles. Trabaja con vectores de números reales Dcon desviaciones estándarD que codifican las posibles soluciones de problemas numéricos. Utiliza recombinación o cruce (crossover aritmético), mutación y la operación de selección, ya sea determinística o probabilística, elimina las peores soluciones de la población y no genera copia de aquellos individuos con una aptitud por debajo de la aptitud promedio.

2. Programación Evolutiva: Inicialmente fue diseñada como un intento de crear inteligencia artificial. La representación del problema se realiza mediante números reales (cualquier estructura de datos), y emplea los mecanismos de mutación y selección. El procedimiento es muy similar a las estrategias evolutivas con la diferencia de que no emplea la recombinación, de tal forma que son denominadas en conjunto algoritmos evolutivos como una manera de diferenciarlas de los algoritmos genéticos.

3. Algoritmos Genéticos: Modelan el proceso de evolución como una sucesión de frecuentes cambios en los genes, con soluciones análogas a cromosomas. Trabajan con una población de cadenas binarias para la representación del problema, y el espacio de soluciones posibles es explorado aplicando transformaciones a éstas soluciones candidatas tal y como se observa en los organismos vivientes: cruce, inversión y mutación. Como método de selección emplean en mecanismo de la ruleta (a veces con elitismo). Constituyen el paradigma más completo de la computación evolutiva ya que resumen de modo natural todas las ideas fundamentales de dicho enfoque. Son muy flexibles ya que pueden adoptar con facilidad nuevas ideas, generales o específicas, que surjan dentro del campo de la computación evolutiva. Además, se pueden hibridar fácilmente con otros paradigmas y enfoques, aunque no tengan ninguna relación con la computación evolutiva. Se trata del paradigma con mayor base teórica.

4.2. Procesos Evolutivos Básicos

Un lugar bueno para empezar es preguntarse los componentes básicos de un sistema evolutivo. La primer nota que debemos tomar en cuenta es que existen al menos dos posibles interpretaciones de un *Sistema Evolutivo*. Es frecuentemente usado para describir un sistema que cambia con incrementos a lo largo del tiempo. La segunda interpretación es aquella que tiene un sentido biológico, es decir, que es un sistema evolutivo Darwiniano.

Para ello, precisamos saber lo que entendemos por un *Sistema*. Una manera es identificando lo que precisamente constituye al sistema, es decir todas sus partes. A continuación se presenta el conseso de lo que se entiende por un sisteme evolucionario Darwiniano.

- Una o más poblaciones compiten por limitados recursos
- Noción de que las poblaciones cambien dinámicamente debido al nacimiento y la muerte de individuos.

- El concepto de la aptitud en donde refleja la habilidad de un individuo de sobrevivir y de reproducirse
- El concepto de herencia variacional: la descendencia se parece mucho a sus padres, pero son no es identicos.

4.3. Un primer ejemplo: Un simple Sistema Evolutivo

La primera cuestión a afrontar es cómo representar a los individuos (organismos) que componen una población en evolución. Una técnica bastante general es describir a un individuo como un vector de longitud de las características L que se eligen presumiblemente debido a su relevancia (potencial) para estimar la aptitud de un individuo. Entonces, por ejemplo, los individuos podrían caracterizarse por:

$$< \text{color cabello, color ojos, color piel, altura, peso} >$$

Podríamos pensar libremente en este vector como que especifica la composición genética de un individuo, es decir, su genotipo especificado como un cromosoma con cinco genes cuyos valores dan como resultado un individuo con un conjunto particular de rasgos. Alternativamente, podríamos considerar tales vectores como descripciones de los rasgos físicos observables de los individuos, es decir, su fenotipo. En cualquier caso, por especificando adicionalmente el rango de valores (alelos) que tales características pueden asumir, se define un espacio de cinco dimensiones de todos los posibles genotipos (o fenotipos) que los individuos pueden tener en este mundo artificial.

```
EV:
    Generar una poblacion inicial de M individuos
    DO FOREVER:
        Seleccionar un miembro del la actual poblacion para que sea padre

        Usa el seleccionado padre para que produzca una descendencia que
        sea similar pero generalmente no una copia precisa del padre

        Selecciona a un miembro de la poblacion para que muera
        END DO
```

4.4. Algoritmos Evolutivos

4.4.1. Búsqueda Aleatoria

El método de búsqueda aleatoria es el primer método que basó su estrategia de optimización en un proceso totalmente estiocástico. Bajo este método, sólo una solución canidato x^k es mantenida durante el proceso de evolución. En cada iteración, la solución cadidato x^k es modificada añadiendo un vector aleatorio Δx. De esta manera, la nueva solución candidato es modelada bajo la siguiente expresión:

$$x^{k+1} = x^k + \Delta x$$

Considerando que la solución candidato x^k tiene d dimensiones $\left(x_1^k, x_2^k, \cdots, x_d^k\right)$, cada coordenada es modificada ($\Delta x = \{\Delta x_1, \Delta x_2, \cdots, \Delta x_d\}$) mediante una perturbación aleatoria Δx, ($i \in [1, 2, \ldots, d]$) mpde;ada por una distribución de probabilidad gaussiana $p(\Delta x_i) = N(\mu_i, \sigma_i)$, se considera que $\mu_i = 0$ dado que el valor de Δx_i añade una modificación alrededor de x_i^k.
Una vez calculado $x^k + 1$, se prueba si la nueva posición mejora la calidad de la solución candidato anterior x^k. De esta manera, si la calidad de $x^k + 1$ es mejor que x^k, el valor de $x^k + 1$ es aceptado como la nueva solución candidato, en caso contrario permanece x^k sin cambio. Esta prueba puede definirse para un caso de minimización como:

$$x^{k+1} = \begin{cases} x^{k+1} & \text{si } f(x^{k+1}) < f(x^k) \\ x^k & \text{si } f(x^{k+1}) \geq f(x^k) \end{cases}$$

A este criterio de sustitución, de aceptar solamente los cambios que mejoren la calidad de la solución es conocido como "*greedy*". En búsqueda aleatoria, la perturbación Δx impuesta a x^k podría hacer que el

nuevo valor x^{k+1} quede fuera del espacio de búsqueda denido X. Fuera del espacio de búsqueda X no existe denición de la función objetivo $f(x)$. Para evitar este problema, el algoritmo debe proteger la evolución de la solución candidato x^k, de tal forma que si x^{k+1} queda fuera del espacio de búsqueda X, se le debe de asignar una muy mala calidad (representado por u valor muy grande). Esto es $f(x^{k+1}) = \infty$ para el caso de la minimización o $f(x^{k+1}) = -\infty$ para el caso de la maximización.

Algoritmo 1.2 Método de búsqueda aleatoria para resolver 1.14	
1.	k $\leftarrow$ 0
2.	$x_1^k \leftarrow$ Aleatorio $[-3,3]$, $x_2^k \leftarrow$ Aleatorio $[-3,3]$
3.	**while** (k $<$ *Niter*) {
4.	$\Delta x_2 = N(0,1)$
5.	$x_1^{k+1} = x_1^k + \Delta x_1$, $x_1^{k+1} = x_1^k + \Delta x_1$
6.	$\mathbf{x}^k = (x_1^k, x_2^k)$, $\mathbf{x}^{k+1} = (x_1^{k+1}, x_2^{k+1})$
7.	**If**$(\mathbf{x}^{k+1}$ **x)** $\{ f(\mathbf{x}^{k+1}) = -\infty \}$
8.	**If** $\left(f(\mathbf{x}^{k+1}) < \left(f(\mathbf{x}^k)\right)\right)$ { $\mathbf{x}^{k+1} = \mathbf{x}^k$
9.	k $\leftarrow$ k + 1 }

Figura 1: Algoritmo de Búsqueda Aleatoria

4.4.2. Temple Simulado

El temple simulado o en inglés 'Simulated annealing' es una técnica de optimización que emula el proceso de templado en materiales metálicos. La idea del templado es enfriar el material controladamente de tal manera que las estructuras cristalinas puedan orientarse y evitar los defectos en las estructuras metálicas. El uso del templado como inspiración para la formulación de algoritmos de optimización fue por primera vez propuesto por Kirkpatrick, Gelatt y Vecchi en 1983. Desde entonces se han sugerido varios estudios y aplicaciones para analizar los alcances del método.A diferencia de algoritmos basados en gradiente, el método del temple simulado presenta una gran habilidad para evitar quedar atrapados en los mínimos locales.
En el templado simulado, el valor de la función objetivo que se intenta optimizar es análogo a la energía de un sistema termodinámico. En altas temperaturas el algoritmo permite la exploración de puntos muy distantes entre sí en el espacio de búsqueda, además de que la probabilidad de aceptación de soluciones que no mejoran su estado anterior es muy grande. Po el contrario, a bajas temperaturas, el algoritmo permite únicamente la generación de puntos muy cercanos entre sí, además de que la probabildiad de aceptación se reduce, por lo que ahora sólo las nuevas soluciones que mejoren su estado anterior, serán consideradas.
El *algoritmo de temple simulado* matiene urante su funcionamiento una sola solución candidato (x^k). Dicha solución es modificada en cada iteración utilizando un procedimiento similar al de método de búsqueda aleatoria, donde cada punto es modificado mediante la generación de un vector aleatorio Δx. Sin embargo, este algoritmo no solo acepta los cambios que mejoren su función objetivo, sino que incorpora un mecanismo probabilístico, que le permite aceptar incluso soluciones que le empeoren. Esto con el fin de salir de los mínimos locales.

Bajo estas circunstancias, una nueva solución será aceptada x^{k+1} considerando dos diferentes alternativas

- Si su calidad es superior a la de su antecesor x^k. Esto es si $f(x^{k+1}) > f(x^k)$
- Bajo una probabilidad de aceptación $p_a = e^{\frac{\Delta f}{T}}$, donde T representa la temperatura que controla el proceso de templado, mientras que f es la diferencia de energía entre el puntos $\Delta f = f(x^{k+1}) - f(x^k)$

A continuación mostramos el algoritmo de temple simulado.

Algoritmo 1.3 Método del temple simulado	
1.	Configura T_{ini}, T_{fin}, $\beta\sigma$ y *Niter*.
2.	$k \leftarrow 0, T = T_{ini}$
3.	$x^k \leftarrow$ Aleatorio $[x]$
4.	**while** $((T > T_{fin})$ **and** $(k < Niter))$ {
5.	**while** $\left(x^{k+1} \notin X\right)$ {
6.	$\Delta x \leftarrow N(0, \sigma \cdot \mathrm{T})$
7.	$x^{k+1} \leftarrow x^k + \Delta x$ }
8.	If $\left(x^{k+1}$es peor que $x^k\right)$
9.	$\{\Delta f \leftarrow f\left(x^{k+1}\right) - f\left(x^k\right)$
10.	Pa $\leftarrow$ e- $\frac{\Delta f}{T}$
11.	**If** $(Pa <$ rand $)\{x^{k+1} = x^k\}\}$
12.	$k \leftarrow k+1$
13.	$T \leftarrow \beta \cdot T\}$

Figura 2: Caption

4.4.3. Genéticos

Un algoritmo genético es un método de búsqueda que imita la teoría de la evolución biológica de Darwin para la resolución de problemas. Para ello, se parte de una población inicial de la cual se seleccionan los individuos más capacitados para luego reproducirlos y mutarlos para finalmente obtener la siguiente generación de individuos que estarán más adaptados que la anterior generación.

4.4.4. Implementación

Una premisa es conseguir que el tamaño de la población sea lo suficientemente grande para garantizar la diversidad de soluciones. Los pasos básicos de un algoritmo genético son:

- Evaluar la puntuación de cada uno de los cromosomas generados.
- Permitir la reproducción de los cromosomas siendo los más aptos los que tengan más probabilidad de reproducirse.

- Con cierta probabilidad de mutación, mutar un gen del nuevo individuo generado.
- Organizar la nueva población

Se puede fijar un número máximo de iteraciones antes de finalizar el algoritmo genético o detenerlo cuando no se produzcan más cambios en la población.

Parametros de los algoritmos geneticos.

- Tamaño de la población.
- Probabilidad de cruce.
- Probabilidad de mutación.

Al igual que en otros algoritmos metaheurísticos, los AG presentan algunas ventajas respecto a los métodos clásicos de optimización; por ejemplo, pueden trabajar con poca o ninguna información acerca de la función objetivo, y por ser poblacionales tienen mayores probabilidades de escapar de óp-timos locales, en comparación con los métodos determinísticos. Sin embargo, entre sus desventajas está el hecho de que no garantizan una convergencia al óptimo real, y en muchos casos son más tardados que las técnicas deterministas

Un ejemplo Veamos como aplicar un algoritmo genetico para minimizar la siguiente función . la cual es dificil desde el punto de vista computacional, pues contiene bastantes optimos locales

$$f(x_1, x_2) = a + exp(1) - a * exp(-b\sqrt{\frac{1}{d}(x_1^2 + x_2^2)}) - exp(\frac{1}{d}(\cos c_1 x_1 + \cos c_2 x_2))$$
$$x_1, x_2 \in [-15, 30]$$

```
function [temp, ind] = FUNCION(x, d, N) y=[];%
Ackley
for j=1:N
   a = 20; b = 0.2; c = 2*pi;
   s1 = 0; s2 = 0;
   for i=1:d;
      s1 = s1+x(j,i)^2;
      s2 = s2+cos(c*x(j,i));
end
y(j,1) = -a*exp(-b*sqrt(1/d*s1))-exp(1/d*s2)+a+exp(1);
end
[temp,ind]=sort(y);
```

Básicamente un AG es un programa que genera una población inicial aleatoria de padres, y durante cada generación selecciona pares de padres, considerando su valor de $f(x_i,)$, para realizar intercambios de material genético, o cruza, y generar pares de hijos; tales hijos serán mutados con una cierta probabilidad, y nalmente competirán por sobrevivir a la siguiente generación con los padres, proceso que se conoce como elitismo. Aunque dicho comportamiento produce la convergencia del algoritmo, también es cierto que provoca que la población sea 'arrastrada' por el mejor individuo de la población, lo que conduce en ocasiones un estancamiento de la población

La desventaja de esto es que podria llegarse encontrar un optimo local en ves de uno global

4.4.5. Estrategias Evolutivas

Las EE son algoritmos inspirados en la evolución y al igual que otros basados en tal metáfora, sus individuos realizarán una "evolución", o mejora, con respecto a alguna función objetivo y mediante operadores que imitan dicho proceso.

Podriamos por ejemplo considerar minimizar f tal que:

$$f(x_1, x_2) = -sen(x1)sen^{2m}(\frac{x_1^2}{\pi}) + sen(x_2)sen^{2m}(\frac{2x_2^2}{\pi} x_1, x_2 \in [0, \pi] m = 10$$

Para ello podemos implementar el siguiente codigo

```
function [f, i1] = FUNCION(x, d, mu)
y=[];
for i1=1:mu
m = 10;   s = 0;
for i2 = 1:d
s = s + sin(x(i1,i2))*(sin(i2*x(i1,i2)^2/pi))^(2*m);
end
y(i1,1) = -s;
end
[f, i1]=sort(y);
```

Esta función en particular tambien presenta minimos locales por lo que es dificil estudiarla al menos con las estrategias que conocemos hasta el momento Esencialmente, los operadores que se utilizan en las EE son muy parecidos a los de un AG, cuando menos nominalmente; sin embargo, la diferencia más notoria es que en las EE el operador de mutación no es un solo valor, sino una matriz de valores de mutación. Esta es una de sus características más interesantes, e incluso se considera como la más importante. La principal diferencia en la inicialización en las EE con respecto a otros algoritmos evolutivos radica en que se crean las matrices de covarianza y rotaciones (σ, α)

Se muestra a continuación las principales estrategias a seguir para generar un algoritmo evolutivo.

Algoritmo 3.1 Estrategias Evolutivas, versión$(\mu / \rho + \lambda)$ -EE	
1.	Configurar parámetros del algoritmo
2.	Inicializar y evaluar población inicial
3.	Mientras (no se cumpla criterio)
4.	**for** 1 : λ
5.	Seleccionar ρ padres, parámetros de estrategia
6.	Seleccionar ρ padres, variables de decisión
7.	Recombinar parámetros de estrategia
8.	Recombinar variables de decisión
9.	Mutar parámetros de estrategia
10.	Mutar variables de decisión
11.	Seleccionar de la población de hijos y de la población de padres anterior a los nuevos padres
12.	Mostrar resultado

Figura 3: Muestra de un algoritmo evolutivo

4.4.6. Evolución Diferencial

El algoritmo de Optimización Evolución Diferencial (Differential Evolution, por sus siglas en inglés), es un algoritmo poblacional de búsqueda directa y simple, el cual es capaz de optimizar hasta alcanzar el óptimo global en funciones multimodales, no diferenciales y no lineales. Fue propuesto por *Kenneth Price y Rainer Storn* en 1995. Desde entonces, el algoritmo DE ha probado sus capacidades en concursos Internacionales de la IEEE en Optimización Evolutiva, asi como con grandes aplicaciones en el mundo real.

El DE se basa en perturbar a los miembros de la población generada con diferencias escaladas de distintos miembros de una misma población. Lo hacen característico de los demás algoritmos evolutivos dado que esta inspirado en un fénomeno biológico, social, físico o químico. El algoritmo DE fue diseñado para que se cumpliera con (i) capacidad de lidiar con funcio-nes objetivo no diferenciables, no lineales y multimodales,

(ii) fácil implementación, (iii) pocos parámetros de ajuste, (iv) propiedades de convergencia consistentes en pruebas consecutivas in-dependientes y (v) capacidad de paralelizarse para lidiar con funciones de alto costo computa-cional. A continuación se presenta el algoritmo o un diagrama de flujo génerico de las operaciones llevadas acabo por el algoritmo evolución diferencial. De acuerdo cone studios reportados recien-

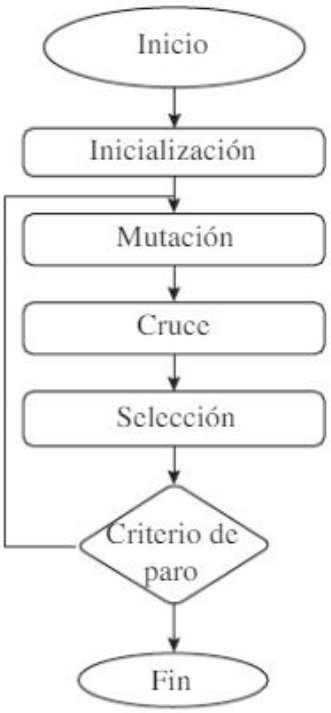


Figura 4: Caption

temente, el algoritmo DE ha mostrado un mejor desempeño que muchos algoritmos evolutivos en términos de convergencia, velocidad y robustez sobre distintas funciones de prueba.

4.4.7. Búsqueda Armónica

En este algoritmo se representa a una armonía como en vector n-dimensional de números reales.

- Un conjunto inicial de vectores de armonía son generados aleatoriamente y almacenados dentro de la memoría de armonías (HM).
- Una nueva armonía candidada es calculada a partir de los elementos contenidos en la HM, para esto existe un parámetro conocido como consideración de memoria.
- Se actualiza la memoria de armonías, por medio de la comparación de la nueva armonía candidata y el peor de los vectores de armonía.

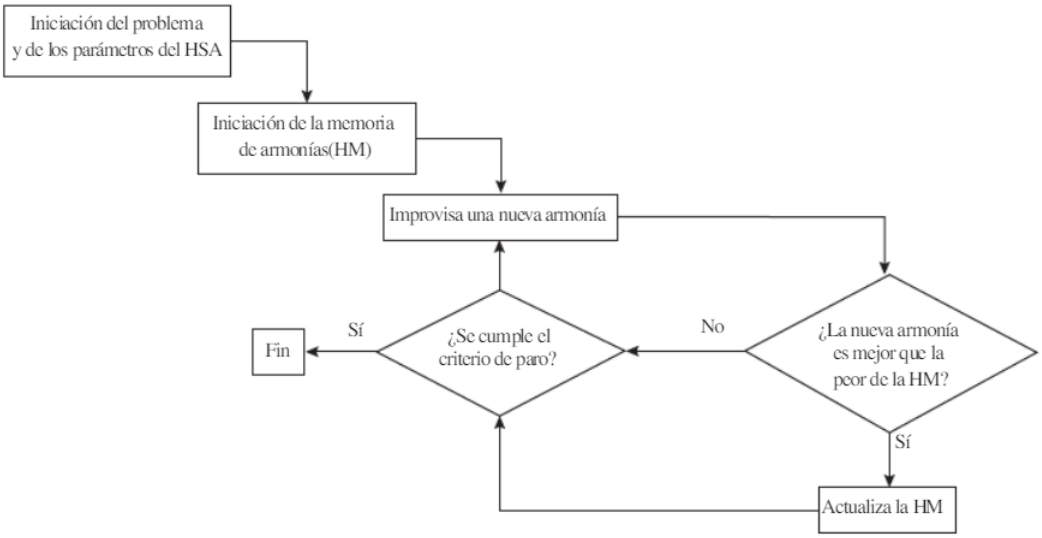


Figura 6.2. Diagrama de flujo del algoritmo HSA.

Un popularidad del algoritmo HSA se debe a un conjunto de características que lo distinguen del resto de los algoritmos metaheurísticos. Algunas de éstas son:
1. Para generar nuevas soluciones se consideran todas las soluciones existentes, no sólo dos de ellas como los padres en los algoritmos genéticos.
2. Se considera de forma independiente cada variable de decisión en un vector de soluciones.
3. Los valores de las variables de decisión son continuos, por lo que no se pierde precisión.

4.4.8. Sistemas Inmunológicos Artificiales

Para definir lo que es un sistema inmune artificial recurriremos a una de las definiciones más completa propuesta por Leandro Nunes de Castro y Jonathan Timmis.
Los sistemas inmunes artificiales son sistemas adaptativos, inspirados por la teoría inmunológica, funciones, principios y modelos inmunológicos observados, los cuales son aplicados a la solución de problemas.
Aunque la cantidad de modelos y aplicaciones del sistema inmune va en ascenso, no existe un esquema general de cuáles son los elementos esenciales que un sistema inmune artificial debe poseer.
Nunes De Castro y Timmis sugieren en su libro utilizar un esquema general de sistemas computacionales con inspiración biológica, tal como los algoritmos evolutivos o las redes neuronales. Este esquema consta de tres partes principales que deben denirse, y que a continuación se muestran:

- Representación de los componentes del sistema.
- Un conjunto de mecanismos para evaluar las interacciones de los individuos con el ambiente y entre ellos.
- Un proceso de adaptación que gobierne las dinámicas del sistema, es decir, el algoritmo en sí.

Representación de los componentes del sistema.
Un conjunto de mecanismos para evaluar las interacciones de los individuos con el ambiente y entre ellos.
Un proceso de adaptación que gobierne las dinámicas del sistema, es decir, el algoritmo en sí.

A diferencia de otras técnicas bio-inspiradas (por ejemplo los algoritmos genéticos), el sistema inmune artificial no tiene un algoritmo general único.

4.4.9. Optimización basada en electromagnetismo

El algoritmo EMO es un método basado en población propuesto por Birbil y Fang para resolver modelos de optimización continuos usando variables acotadas. Es decir, problemas de optimización. El algoritmo EMO replica el problema de las cargas, donde cada carga representa una solución a un campo dado, lo cual representa que cada cantidad de carga es una parte de la solución y refleja su cálidad. Este algoritmo, permite encontrar solución a problemas de minimización del tipo $f(X)X\epsilon[l,u]$ a través de los siguientes cuatro pasos.

- Inicialización: un conjunto de m partículas son tomadas aleatoriamente considerando un espacio R definido por el límite superior u y el límite inferior l
- Búsqueda local: se realiza la búsqueda de un valor mínimo en la vecindad de un punto X_p, donde $p\epsilon[m1, m2]$ y m es el número total de individuos de la población.
- Cálculo del vector de fuerza total: con base en el valor de la función objetivo se calculan las cargas y fuerzas para cada elemento de población de partículas. : cada partícula de población es desplazada de acuerdo a la fuerza total calculada con base en el valor de la función objetivo.

```
%Inicializacion, Algoritmo EMO Standar
%Prueba con función de Rosenbrock
function [x.fx] = inicializa (m,n,u,l)
%Se genera las partículas de forma aleatoria en el espacio de búsqueda
for i = 1:n
%Se obtiene el número aleatorio
lambda= rand(l,m);
x(i,:) = l(i) + lambda * (u(i) -l(i));
end
%Se evalúa cada partícula en la función objetivo (Rosenbrock)
for j =1:m
fx(j) = rosen(x(:,j));
end
```

El código anterior es una implementación del algoritmo EMO para la función de Rosenbrock.

4.4.10. Colonia Artificial en Abejas, Hormigas

El problema de optimización de colonia, es una tecnica probabilística para resolver problemas computacionales que pueden ser reducidos a encontrar un camino bueno en una gráfica.

La forma más fácil de entender como el algoritmo de optimización por colonia de hormigas funciona, es a través de un ejemplo. Consideremos el problema del agente viajero, consiste en, dado un conjunto de ciudades y la distancia entre estas, encontrar la ruta más corta que visite a todas y regrese al lugar de partida.

Las hormigas construyen la solución de la siguiente manera. Cada hormiga selecciona una ciudad aleatoria. Después, en cada paso se mueve a un nuevo vértice queno haya sido visitado de forma probabilística, esta decisión esta basada en información como la feromona. Cada vez que una hormiga finaliza su recorrido, la feromona de cada arista es actualizado de acuerdo a la calidad de la solución resultante.

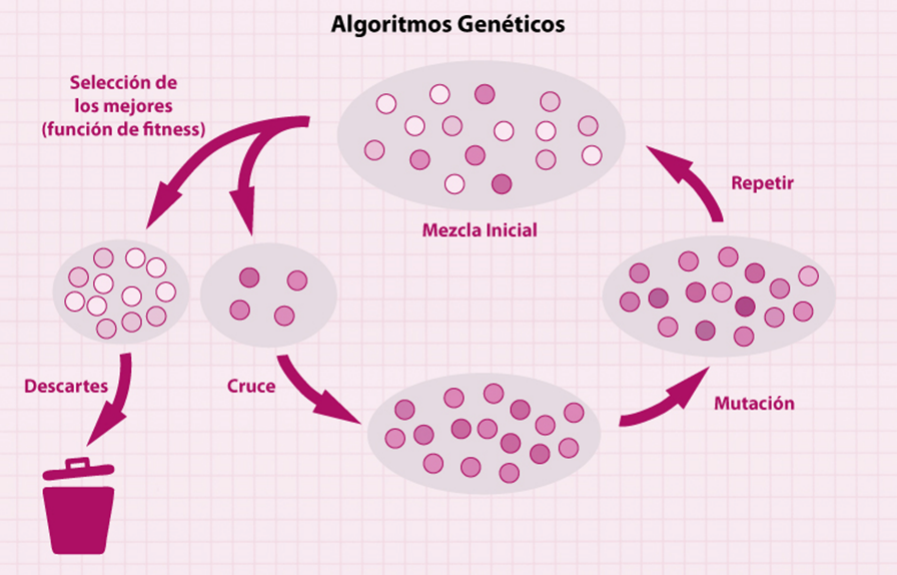
5. Estudiar, evaluar y argumentar la elección de algún algoritmo evolutivo.

5.1. Estudio.

El algoritmo genético como ya mencionamos en la sección anterior, es un algoritmo que restringe una búsqueda con los siguientes pasos.

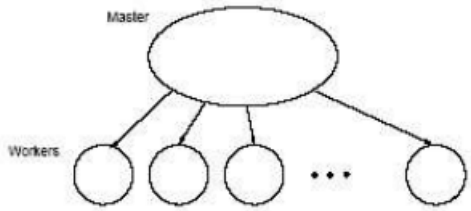
- Generar población inicial.
- Seleccionar de esta población individuos más capacitidados.
- Reproducir y mutar estos individuos.
- Obtiene una nueva generación de individuos mejor adaptados

El proceso se repite hasta llegar a una solución aceptable. El algoritmo genético podemos notar que se basa de un sistema de selección natural, en el cual tras cada iteración solo los individuos más aptos sobreviven, y que por tanto serán base para las generaciones posteriores. Esta capacidad les permite resolver muchos problemas que no son de un capo en específico. Solo debemos definir un problema y una función de evaluzación de los individuos para poder resolver cualquier tipo de problema. Al traterse de una técnica heurística el tiempo de ejecución puede volverse un obstáculo si el espacio de búsqueda de soluciones es muy amplio, además de que si el factor de mutación no es elegido con cuidado, podríamos hacer un algirtmo que no convergiese a el óptimo búscado. Por eso es necesario garantizar la diversidad de los individuos al realizar la técnica, y también es esta complicación por la cual es natural aplicar paralelización, precisamente en la sub sección siguiente discutiremos esto con más profundidad, para dar la justificación a la elección de un algoritmo genético.



5.2. Evaluación

Justificación Decidimos elegir el algoritmo genético dado que la labor de selección de los individuos, es una tarea que se puede paralelizar y que además no afecta la solución final del algoritmo. La selección debe ser realizada de manera global si queremos que el algoritmo produzca el mismo resultado que el secuencial original. Otro punto del algoritmo donde se puede realizar la paralelización de la población, podemos en lugar de generar una población muy grande, grupos pequeños de poblaciones que se comuniquen entre ellas, esté punto es el que de acuerdo a la bibliografía más apremia realizar una paralelización contra el secuencial ya que incluso puede ser algo favorable, comenzar con sub poblaciones para la búsqueda, incluso este proceso de paralelización se conoce como generar nichos.

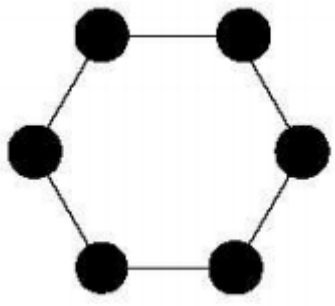


Esta es una representación esquemática de un algoritmo genético paralelizado, con la aplicación del proceso paralelo en la evaluación de los individuos, ya que suele ser también la que toma más tiempo.

El intercambio de información entre los nodos realiza los siguientes pasos

- Nodo maestro envia subconjunto de individuos a cada nodo.
- Cada nodo retorna el valor de evaluación al nodo maestro.

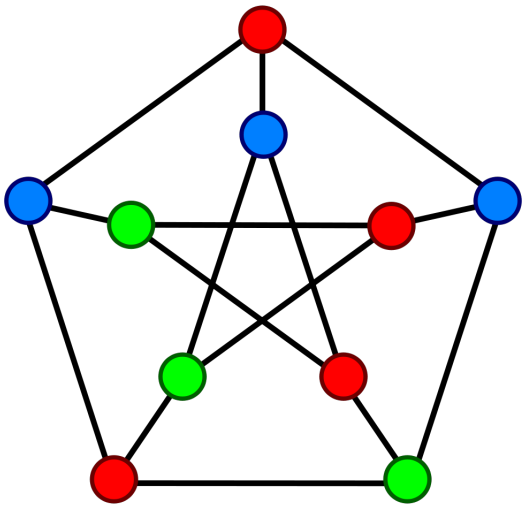
De forma tradicional, el nodo maestro espera a recibir los valores de adecuación de todos los individuos para hacer la siguiente generación, pero también podemos hacer que los nodos más lentos no se comuniquen con el nodo maestro, aunque aquí perderíamos información del comportamiento original.



Una segunda forma de generar paralelización, es generar una arquitectura de muchas poblaciones, este es el caso en donde cada sub población es repartida hacía un diferente procesador y es evaluada por separado. Se encontró que en está arquitectura, las poblaciones por separado convergen más rápido hacía valores óptimos, aunque en contraste, estos valores no son tan buenos como los del proceso secuencial.

6. Problema de aplicación elegido en el punto anterior

El problema de aplicación es colorear una gráfica de n vértices con 3 colores, esto es, dado $G(E, V)$ debemos asignare un color a cada vértice de tal forma que no haya dos vértices adyacentes que sean del mismo color.



En este caso se tienen las siguientes definiciones:

- Person: Una coloración arbitraria de la gráfica, esta se puede ver como un array que nos indica el color de cada vertice $[1, 0, 2, 1, ..., 0]$.
- Fitness: Número de aristas que conectan a dos vértices del diferente color.
- Crossover: Dadas 2 gráficas (padres) elegimos un número aleatorio entre 0 y n y creamos una nueva gráfica que tenga los primeros k colores iguales a los de la primera y el resto igual a los de la segunda.

El procedimiento es el siguiente:

1. Generar población aleatoria (primer generación).
2. Obtener fitness.
3. Obtener a los individuos con mayor fitness.
4. Hacer crossover para obtener nueva generación.
5. Repetir paso 2

El modelo tiene los siguientes hyper-parámetros:

- population_size: Número de personas en la población.
- n_nodes: Número de nodos aleatorios de la gráfica
- n_generations: Número de generaciones que se usarán para evolucionar a la población.
- percentage_keep: Porcentaje de la población con mejor fitness que se copia de a la siguiente población.

7. Programa - Simulación

Para la simulación se utilizara python y la libreria networkx junto con matplotlib asi como multiprocessing lo que nos permitira implementar nuestro algoritmo de manera paralela.

`ds`

8. Conclusiones

Referencias

[1] Coley, D. A Introduction to Genetic Algorithms for Scientists and Engineers. (First ed.).

[2] Goldberg, D Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional.

En la referencia .ºptimización. Algoritmos programdos con MATLAB", Cuevas et al., Alfaomega, México, 2016, podrán encontrar los siguientes algoritmos evolutivos: