UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA

Alumna: M.C. Alma Eliza Guerrero Sánchez

Doctorado en ingeniería

Seminario de tesis III

Método de recocido simulado (Simulated Annealing).

Inspirado en el recocido en la metalurgia, donde el metal se calienta rápidamente a una temperatura alta y luego se enfría lentamente, lo que aumenta su resistencia y facilita el trabajo. Este algoritmo se considera una versión modificada y mejorada del algoritmo Hill-Climbing, sin embargo, se considera una versión optimizada del mismo por el proceso que vemos en la figura 1. El algoritmo de Montañismo o Hill Climbing tiende a caer en máximos locales, puntos silla. Esto debido al proceso característico de este algoritmo. Sin embargo, el algoritmo de simulado recocido al implementar la Distribución de probabilidad de Boltzmann, el cual más adelante hablaremos a profundidad, evita o disminuye la probabilidad que este algoritmo caída en valores locales, puntos de silla, entre otros..

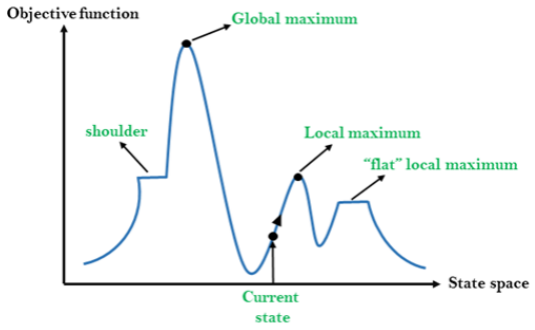


Figura 1.

Es bien sabido que el proceso de un algoritmo metaheurístico o de optimización combinatoria tiene ciertas fases importantes. Como lo son soluciones aleatorias, costo, posibles soluciones o soluciones vecinas, criterios de paro y solución final. Esto facilita el desarrollo de algoritmos a partir de procesos de la naturaleza o en este caso procesos dentro de la metalurgia. Esto lo presentamos en la figura 2, el cual es la analogía y comparación

Proceso de Recocido	Optimización Combinatoria
Estados Del Sistema	Soluciones
Energía	Costo
Cambio de estado	Soluciones vecinas
Temperatura	Criterios de paro
Estado congelado	Solución Final

Figura 2. Analogía y comparación de algoritmo de recocido simulado proceso de optimización combinatoria.

Reporte de algoritmo metaheurístico: Recocido Simulado.

Como podemos observar en la figura 2, el proceso inicia al tener estado de l sistema de manera aleatoria o generar soluciones aleatorias que pudieran encajar en la resolución de nuestro problema. Por otro lado, tenemos los cálculos el costo, fitness o en el caso del algoritmo de recocido simulado Energía, que básicamente es que tan buena o mala es la solución en potencia que se está generando. La soluciones vecinas o cambio de estado es la exploración de nuevas soluciones en potencia en el caso de algoritmos genéticos tenemos cruza, mutación. El criterio de paro puede ser iterativo, es decir por número de iteraciones o hasta que la temperatura llegue a un cierto valor. Y finalmente estado de congelado o la solución final que necesitamos.

Pseudocódigo

A continuación, se presenta el Pseudocódigo del sistema de recocido simulado.

Algoritmo: Recocido Simulado

Procedure Recocido_Simulado
Returns A STATE sk
Entrada:
T0, Temperatura inicial
j, función de costo o energía
s0, Estado inicial
temp-schedule, horario de enfriamiento
neighbor, función de estado vecino

T ← T0
Sk ← S0
for t=1 **to** tmax **do**
s_{k+1} ← neighbor(s_k)
(ΔE ← J(s_{k+1})-J(s_k)
if min(1, $e^{\frac{\Delta E}{T}}$)≥ rand(0,1) **then**
s_k <- s_{k+1}
end if
T ≤ temp-schedule(t)
end for
end procedure

Solución del problema

Problema:

Minimizar:

$$f(\vec{x}) = (x_1 - 10)^3 + (x_2 - 20)^3$$

Sujeto a:

$g_1(\vec{x}) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0$

$g_2(\vec{x}) = (x_1 - 6)^2 + (x_2 - 5)^2$-82.81 ≤ 0

Reporte de algoritmo metaheurístico: Recocido Simulado.

Donde 13 ≤ x_1 ≤ 100 **y** 0 ≤x_2≤ 100

En la figura 3 se muestra el diagrama UML del desarrollo del programa, donde podemos ver un panorama del desarrollo del algoritmo de manera general. Métodos, atributos y funciones matemáticas utilizadas para el desarrollo del programa.

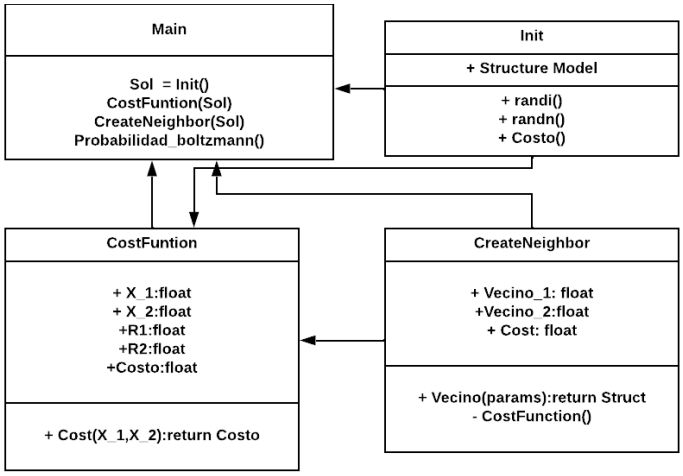
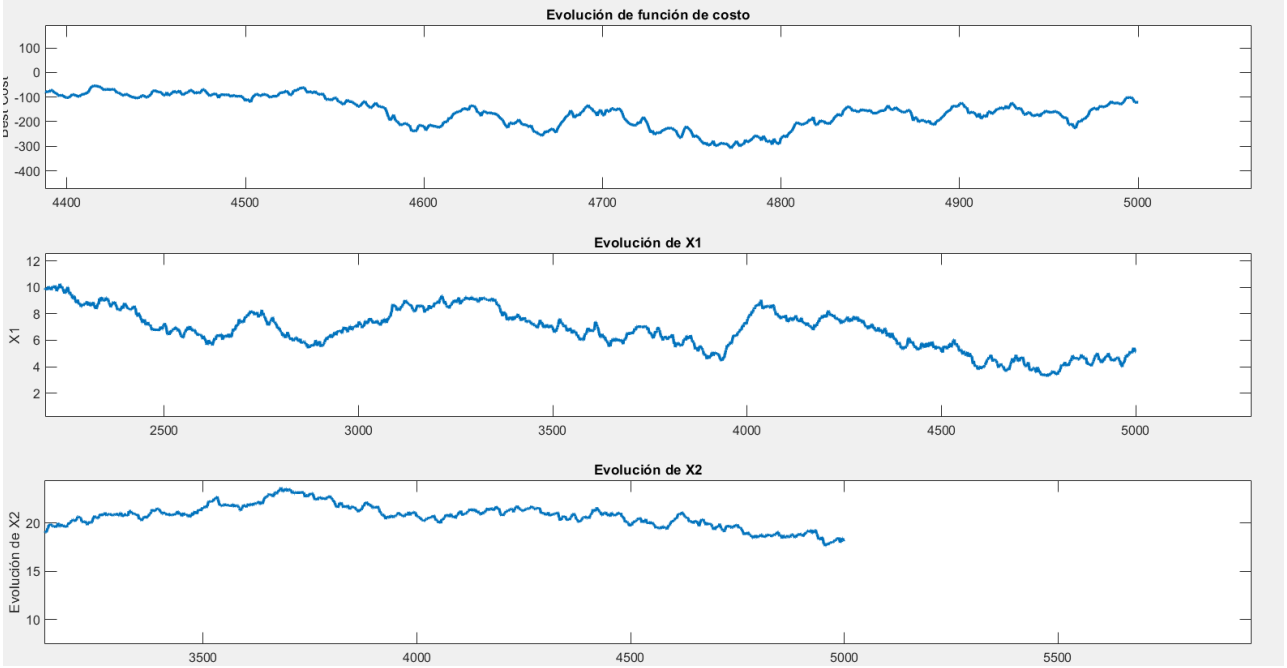


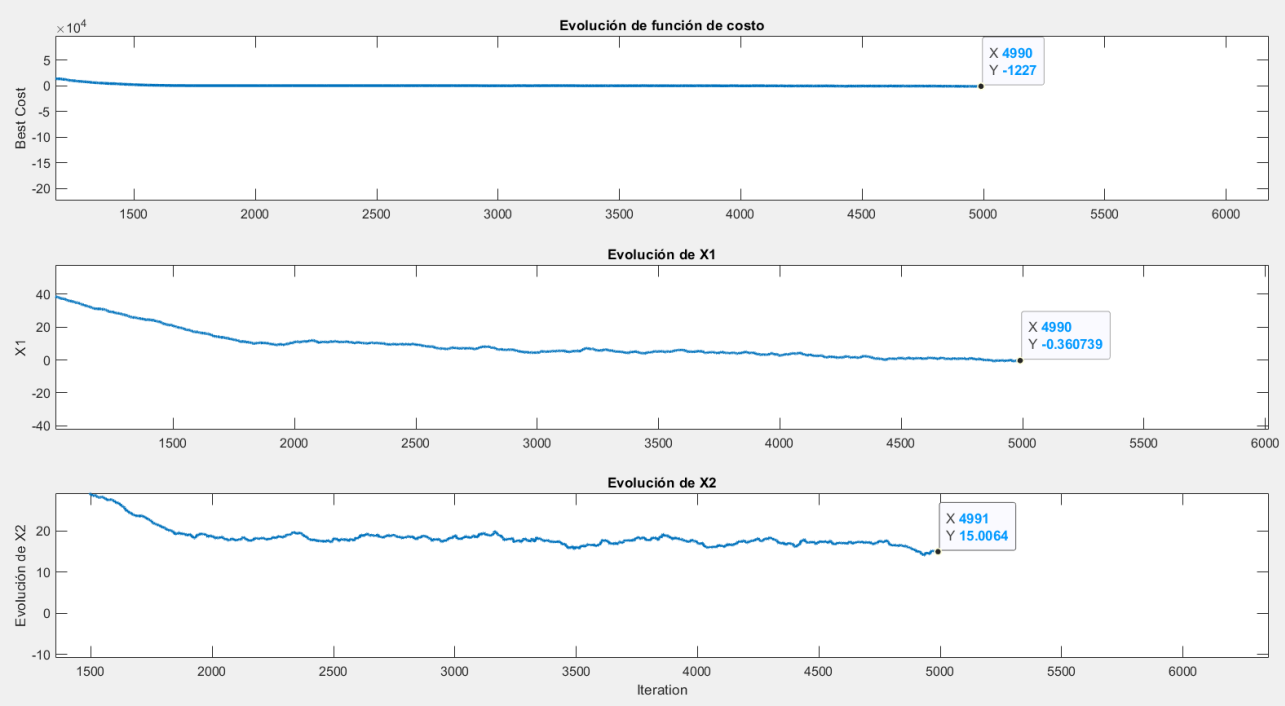
Figura 3. Diagrama UML de algoritmo metaheurístico **.**

Es importante mencionar que un elemento clave para la evolución y que este algoritmo funcione es la probabilidad de Boltzmann también conocida como función de distribución de energía. La distribución de Maxwell-Boltzmann es la función de distribución clásica, para la distribución de una cantidad de energía entre partículas idénticas pero distinguibles.

Gráficas



Reporte de algoritmo metaheurístico: Recocido Simulado.



Conclusiones

Este algoritmo es eficiente, rápido y con un consumo computacional bajo en comparación a otros algoritmos. No importa las condiciones iniciales que tenga, siempre determinar la mejor solución. Esto debido a los modelos matemáticos utilizados para resolver el problema. Además, el tiempo de desarrollo de igual forma es corto debido a que no utiliza demasiadas, Sin embargo, un punto débil en cuanto a la generación de soluciones. Porque como vimos, solo genera una solución por iteración en comparación de otras soluciones que generan 100 o 50 o 1000 por iteraciones, depende del desarrollador. Este punto pudiera ser una amenaza para la exploración del espacio de búsqueda de las soluciones. Sin embargo, en una sección de clase

Código fuente

Main.m

```
clc;
clear;
close all;
MaxIt = 5000;       % Maximum Number of Iterations
T0 = 10000;        % Initial Temp.
alpha = 0.1;      % Temp. Reduction Rate
rbd = []
% %% Initialization
% % Create and Evaluate Initial Solution
%función que me regrese una structura inicial con dos valores x2 y x2
con
%su función de costo
```

```
sol = Init();
% % % Initialize Best Solution Ever Found
BestSol = sol.Cost;
BestCost = zeros(MaxIt, 1); %Array con las mejores soluciones
% % % % Intialize Temp.
T = T0;
% % Storage
x1 = zeros(MaxIt, 1); %Array con las mejores soluciones
x2= zeros(MaxIt, 1);
rest_1= zeros(MaxIt, 1);
rest_2= zeros(MaxIt, 1);
genera= zeros(MaxIt, 1);
%%
for it = 1:MaxIt %%cambiar paro por un if
% % %new Neighbor
newsol = CreateNeighbor(sol);
if newsol.Cost <= sol.Cost % If NEWSOL is better than SOL
    sol = newsol;
% % % % %
else % If NEWSOL is NOT better than SOL
% % % %
 DELTA = (newsol.Cost-sol.Cost)/sol.Cost;
% % % %
 P = exp(-DELTA/T); %%%->probabilidad
% %
  if rand <= P
      sol = newsol;
       end
% % % %
  end
     if newsol.Cost <= sol.Cost
               BestSol = newsol;
     end
% %    %     % Store Best Cost Ever Found
 BestCost(it) = BestSol.Cost;
 x1(it) = BestSol.X1;
x2(it) = BestSol.X2;
rest_1(it) = BestSol.R1;
rest_2(it) = BestSol.R2;
%
%
% % %
% % Update Temp.
T = alpha*T;
X = ['Iteración Número ',num2str(it),' con una temperatura de:
',num2str(T), ' °C'];
%disp(X)
%  % Plot Best Solution
end
% %% Results

subplot(3,1,1)
plot(BestCost, 'LineWidth', 2);
title('Evolución de función de costo')
ylabel('Best Cost');
```

```
subplot(3,1,2)
plot(x1, 'LineWidth', 2);
title('Evolución de X1')
ylabel('X1');
subplot(3,1,3)
plot(x2, 'LineWidth', 2);
title('Evolución de X2')
xlabel('Iteration');
ylabel('Evolución de X2');

figure ();
subplot(2,1,1);
plot(rest_1,'LineWidth', 2)
title('Restricciones R1')
subplot(2,1,2);
plot(rest_2,'LineWidth', 2)
title('Restricciones R2')
xlabel('Iteraciones');
```

Init.m

```
function model = SolInit()
X1 = 0:0.1:100; % LIMITES
X1_v =X1(randi([1,numel(X1)]));
X2 = 13:0.1:100; % LIMITES
X2_v = X2(randi([1,numel(X2)]));


penal_x1 = randn();

Cost = (X1_v-10)^3 + (X2_v-20)^3;
R1 = -(X1_v-5)^2 - (X2_v-5)^2 + 100;
R2 = (X1_v-6)^2 + (X2_v-5)^2 - 82.81;
if R1 > 0 || R2 > 0
   FCOST = Cost + penal_x1;
end
FCOST = Cost

     model.X1 = X1_v;
     model.X2 = X2_v;
     model.Cost = FCOST;
     model.R1 = R1;
     model.R2 = R2;
 end
```

CreateNeighbor.m

```
function vecino = CreateNeighbor(x);
valor_X1 = x.X1;
```

Reporte de algoritmo metaheurístico: Recocido Simulado.

```
valor_X2 = x.X2;
step_size = 0.1;
vecino_X1 = valor_X1+randn*step_size;
vecino_X2 = valor_X2+randn*step_size;

V = [vecino_X1 vecino_X2];

costo = CostFuntionVecino(V);
if costo == 1
    vecino_X1 = valor_X1+randn*step_size;
   vecino_X2 = valor_X2+randn*step_size;
   V = [vecino_X1 vecino_X2];
   costo = CostFuntionVecino(V);
else
vecino.X1 = vecino_X1;
vecino.X2 = vecino_X2;
vecino.Cost = costo(1);
vecino.R1 = costo(2);
vecino.R2 = costo(3);
%end
end
```

CostFuntion.m

```
%%costo y restricciones
function costos = CostFuntion(values)
X1_v = values(1);
X2_v = values(2);


Cost = (X1_v-10)^3 + (X2_v-20)^3;
R1 = -(X1_v-5)^2 - (X2_v-5)^2 + 100;
R2 = (X1_v-6)^2 + (X2_v-5)^2 - 82.81;
%penalizar por separado
%evolución pareja de las penalizaciones
if R1 > 0
  penal_x1 = randn();
  costos = Cost + penal_x1;
end
if R2 > 0
   penal_x1 = randn();
   costos = Cost + penal_x1;
end
costos = [Cost,R1,R2];
end
```

CostFuntionVecino

```
%%costo y restricciones
function costos = CostFuntionVecino(values)
```

Reporte de algoritmo metaheurístico: Recocido Simulado.

```
X1_v = values(1);
X2_v = values(2);


Cost = (X1_v-10)^3 + (X2_v-20)^3;
R1 = -(X1_v-5)^2 - (X2_v-5)^2 + 100;
R2 = (X1_v-6)^2 + (X2_v-5)^2 - 82.81;
%penalizar por separado
%evolución pareja de las penalizaciones
if R1 > 0 || R2 > 0
  penal_x1 = randn();
  costos = 1;
end
costos = [Cost,R1,R2];
end
```