Informe final

Práctica investigativa 1: Simulación Optimización

Laura Cardona M

Grupo de Investigación de Simulación y Modelación Matemática

Profesora Paula Escudero

Departamento de Ciencias Básicas
Escuela de Ciencias y Humanidades
Universidad EAFIT
2007

Introducción

Durante esta práctica investigativa se estudiaron aspectos fundamentales de la simulación optimización. En el siguiente informe se recopila información y resultados sobre este estudio. Inicialmente se presenta una breve descripción de la optimización y la heurística, elementos fundamentales en la simulación optimización. Seguidamente se define la simulación optimización, se presentan sus objetivos y su justificación como una herramienta importante en el mundo actual. Luego, se introduce el software OptQuest, herramienta para realizar simulación optimización sobre modelos de simulación, se define su estructura básica y se definen sus principales componentes: la búsqueda dispersa y la búsqueda tabú. Posteriormente, se explora la simulación optimización aplicada a sistemas de inventarios y se muestra un caso de estudio. Además, se muestran dos ejemplos realizados para implementar lo anterior, ambos aplicados a diferentes sistemas de inventarios y resueltos con diferentes formas de simulación optimización. Finalmente se presentan los resultados de estos ejemplos, conclusiones y las referencias bibliográficas.

Optimización y heurística

En la optimización matemática existen un tipo especial de problemas denominados optimización combinatoria. En los que las variables de decisión son enteras y, por lo general, el espacio solución está formado por ordenaciones o subconjuntos de números naturales. En ocasiones, este tipo de problemas se formulan mediante Programación Lineal, pero hay algunos en los que su complejidad computacional lleva a buscar otros métodos de solución [8].

Dada la dificultada práctica para resolver algunos de estos problemas, empezaron a aparecer algoritmos que encontraban soluciones factibles que, aunque no optimicen el problema, por lo menos son cercanas al valor óptimo y se encuentran en un tiempo computacional razonable. De esto surgió la heurística, una rama de la optimización que toma las matemáticas tradicionales y les introduce inteligencia artificial y métodos basados en analogías físicas, biológicas o de procesos evolutivos [7]. Se considera que los heurísticos son "*procedimientos simples, a menudo basados en el sentido común, que se supone ofrecerán una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido*" [8].

En cualquier método heurístico se debe tener lo siguiente: una manera de representar las soluciones, una forma de evaluar su bondad y un mecanismo de movimiento hacia otras soluciones. En este trabajo se utilizaron o estudiaron los heurísticos: búsqueda local, búsqueda aleatoria y búsqueda dispersa.

La búsqueda aleatoria es un heurístico que, utilizando un procedimiento simple, genera muchas soluciones aleatoriamente y elige la mejor de las generadas. Es flexible y fácil de implementar y en general obtiene soluciones buenas [8].
La búsqueda local es un heurístico que parte de una solución inicial y la mejora sucesivamente utilizando operaciones simples hasta que no sea posible mejorarla o se cumpla un criterio de parada. Los movimientos simples se encargan de encontrar soluciones parecidas a la inicial y de esta maneja, crea un vecindario en el que intenta encontrar la mejor solución [8].

Simulación optimización

En los últimos años se han realizado acercamientos a la simulación optimización con diferentes enfoques por ejemplo, han existido aproximaciones estocásticas en las que el objetivo principal involucra análisis del comportamiento de convergencia en un marco de tiempo infinito; estos enfoques no son realmente útiles desde el punto de vista práctico. Por otro lado, surgen, en un marco de tiempo finito, los algoritmos de enumeración, aunque este proceso garantiza soluciones óptimas, tiene aplicaciones limitadas. Los enfoques más recientes han llevado a la construcción de procedimientos de búsqueda inteligentes, capaces de encontrar un óptimo o una solución cercana al óptimo en problemas complejos con espacios de búsqueda grandes, sólo con explorar una pequeña área de las alternativas posibles, estas herramientas se conocen como heurísticos y son la base de la simulación optimización actual.[1]

Una de las mejores herramientas de optimización es la programación lineal, que asume que se tiene una función objetivo y restricciones que pueden ser expresadas por funciones matemáticas lineales y se hacen también suposiciones para simplificar el sistema. La programación lineal encuentra soluciones óptimas sin tener que evaluar todas las posibles alternativas, en un tiempo de cómputo razonable. Sin embargo, no todos los sistemas pueden ser representados matemáticamente por funciones lineales o no lineales, especialmente los problemas de la vida real en empresas e industrias (por ser sistemas complejos). Es así como muchos problemas no pueden ser optimizados por medio de la programación lineal, bien porque no pueden ser representados matemáticamente o porque los supuesto de la programación lineal intentan quitar su incertidumbre, elemento fundamental en el sistema real.

En la programación lineal, una vez se ha formulado el problema es generalmente fácil de resolver pero existen algunos problemas cuyo proceso de solución es prácticamente imposible, por su carácter combinatorio, y requerirían tiempos de cómputo impensables.
La heurística ha sido utilizada por muchos años para dar soluciones aproximadas a problemas complejos. En ocasiones puede dar soluciones cercanas a las óptimas pero en otros puede tener soluciones malas. Así las cosas, se debía decidir entre encontrar soluciones óptimas a problemas pequeños o soluciones no óptimas o malas a problemas más complejos. De esto, surgió la

metaheurística como una manera de tener procedimientos inteligentes de búsqueda e implementaciones computacionales rápidas, pero con el problema de que se debía ajustar cada estratégica para cada problema específico.

La simulación optimización es una herramienta que combina la simulación y la optimización para intentar encontrar mejores configuraciones de un sistema, permitiendo simular un sistema complejo y optimizarlo. De esta manera, utiliza las bondades de dos herramientas poderosas para lograr un nuevo tipo de resultados. En este caso, se utiliza la simulación discreta para modelar y simular un sistema complejo en el que influyen componentes aleatorias, que permite capturar la operatividad del sistema pero encuentra configuraciones óptimas; y la optimización por medio de heurísticos que permite asignar recursos limitados y que encuentra soluciones para optimizar alguna medida de desempeño del sistema en cuestión. También se puede usar con simulación continua, que permite capturar interrelaciones y realimentaciones, evaluar medidas a mediano y largo plazo, pero no permite encontrar óptimos.

Esta alianza surgió, de parte de la simulación, porque se hace necesario explorar varios o muchos escenarios en simulación para encontrar el mejor y en la optimización, para abarcar problemas que presentan incertidumbre y tienen interacciones complejas [1].

El objetivo de *la simulación optimización* es optimizar sistemas complejos, que son aquellos que no pueden ser fácilmente formulados como modelos matemáticos y resueltos con herramientas clásicas de optimización [2].

La mayoría de los problemas del mundo real se caracterizan por tener múltiples no linealidades, relaciones combinatorias e incertidumbre, lo que hace que no puedan ser representados por formulaciones matemáticas manejables [2].

De esta forma, se presenta el siguiente tipo de problema:

$$
\begin{aligned}
&\max (\min) \quad F(x) \\
&\text{sujeto a} \quad Ax \le b \qquad (\text{restricciones}) \\
&\qquad g_l \le G(X) \le g_u \qquad (\text{requerimientos}) \\
&\qquad l \le x \le u \qquad (\text{cotas})
\end{aligned}
$$

Donde x puede ser una variable continua o discreta (con un tamaño de paso arbitrario), $F(x)$ puede ser cualquier función de un conjunto de valores x en los reales, las restricciones deben ser lineales y la matriz A y los valores b deben ser conocidos. Los requerimientos son simples cotas superiores e inferiores (g_l y g_u deben ser constantes conocidas sobre una función lineal o no lineal y todas las variables deben ser acotadas.

Se asume la variable de desempeño $F(x)$ sólo se puede estimar con la simulación estocástica de eventos discretos, al observar la variable aleatoria de salida de la simulación, $Y(x)$, donde $F(x) = E[Y(x)]$ y se conoce poco o nada sobre la superficie de respuesta de $F(x)$ [2].

Al combinar elementos heurísticos, se debe tener en cuenta el costo y timepo de cómputo al resolver y un problema de simulación optimización. Así, un elemento clave en la simulación optimización es el balance del intercambio entre el esfuerzo computacional usado para la estimación de $F(x)$ y el usado para la exploración del espacio de solución en búsqueda de mejores soluciones [2].

OptQuest®

OptQuest es una herramienta computacional que, al asociarse con un software de simulación, sirve para optimizar modelos de simulación. Aunque es un paquete independiente al de simulación, tiene mecanismos de comunicación que permiten obtener resultados del modelo de simulación y modificar valores de parámetros en la simulación. Así, se definen dos elementos básicos, el problema (representado por el modelo de simulación) y el optimizador (representado por OptQuest) [9].

En el caso de OptQuest se logró separar el problema del método de solución, mediante un enfoque de "caja negra". Así, el procedimiento de optimización es genérico y está separado (no conoce nada) del tipo de problema, por otro lado, se puede utilizar ese optimizador genérico para muchos problemas o sistemas diferentes.
Se define el problema de optimización fuera del sistema (modelo de simulación) permitiendo que el modelo pueda ser cambiado o se le agreguen aspectos adicionales, mientras las rutinas de optimización permanecen iguales. [1]

De esta manera, se ve en la Figura 1 que el sistema (modelo de simulación) se considera como una caja negra, en que son sólo relevantes la entrada y la salida, sin importar lo que sucede adentro. Bajo este enfoque, el sistema y el optimizador se coordinan mediante la producción de salidas y entradas (Figura 2), es decir, el modelo de simulación produce una salida determinada que es la entrada para el procedimiento de optimización y una vez éste finaliza, genera una entrada para el modelo de simulación. Una de las desventajas de este enfoque de caja negra es que el procedimiento de optimización no se puede valer de ningún tipo de información de cada problema específico, algo que ayudaría en la estrategia de solución [7]. Pero este mismo enfoque permite que el procedimiento sea utilizado en muchos problemas diferentes.

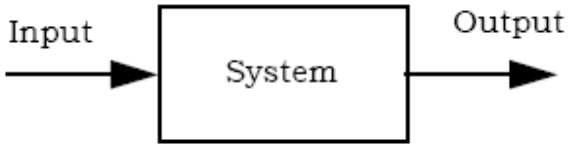

Figura 1: El sistema como una caja negra.

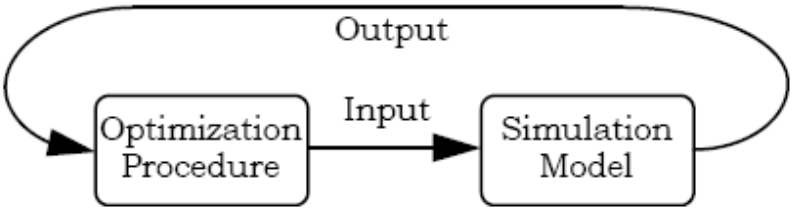

Figura 2: Coordinación entre optimización y simulación

El procedimiento de optimización usa los productos (salidas) del modelo de simulación para evaluar los resultados de los parámetros de entrada (entradas). Según esa evaluación y las evaluaciones pasadas, que son integradas y analizadas con las salidas de la simulación actual, el procedimiento de optimización decide sobre un nuevo conjunto de valores de entrada. Ese procedimiento de optimización está diseñado para llevar a cabo un procedimiento de "búsqueda no monótona", donde las entradas producidas sucesivamente dan lugar a evaluaciones variadas, no siempre mejorando pero que a largo plazo garantizan una trayectoria eficiente hacia la mejor solución. El procedimiento continúa hasta que se satisface algún criterio de parada [7].
En cuanto a los procesos que llevan al optimizador a dar soluciones, se implementan metaheurísticos que encuentren soluciones aproximadas que puedan evaluarse. OptQuest utiliza una combinación de tres métodos metaheurísticos: redes neuronales, búsqueda dispersa y búsqueda tabú, pero el principal es la búsqueda dispersa.

La búsqueda dispersa (BD) tiene algunos aspectos en común con los algoritmos genéticos, pero otros muy diferentes. La BD está diseñada para trabajar con un conjunto de puntos llamados *puntos de referencia* que constituyen buenas soluciones obtenidas en búsquedas anteriores. Luego, se generan sistemáticamente combinaciones lineales de los puntos de referencia para crear nuevos puntos, cada uno de los cuales es asociado con un punto factible. Seguidamente se aplica la búsqueda Tabú (BT) para controlar la composición de los puntos de referencia en cada etapa del proceso [1].

La BT tiene sus raíces en la inteligencia artificial y en la optimización y se caracteriza por tener memoria adaptativa, que le permite hacer uso de la historia de la búsqueda para guiar el proceso de solución. La más simple de las maneras de memoria adaptativa consiste en prohibir la

búsqueda de soluciones que ya han sido visitadas. Sin embargo, el uso de la memoria en OptQuest es más complejo, llamando funciones de memoria que incentiven la diversificación e intensificación. Estos elementos de memoria permiten escapar de óptimos locales y, en varios casos, encontrar el óptimo global.

La búsqueda dispersa es un método evolutivo efectivo para resolver problemas de optimización. Dadas dos soluciones se pueden combinar para mejorar las soluciones que la originaron, "*se basa en el principio de que la información sobre la calidad o el atractivo de un conjunto de reglas, restricciones o soluciones puede ser utilizado mediante la combinación de éstas"* [6]. La BD es un enfoque dirigido por la información, explotando así el conocimiento derivado del espacio de búsqueda, soluciones buenas encontradas en ese espacio y trayectorias a través del espacio en el tiempo. La combinación de estos factores crea un proceso de solución altamente efectivo. La incorporación de este diseño es la responsable de dar al sistema OPTQUEST la habilidad e resolver problemas complejos basados en simulación con eficiencia sin precedentes [1].

Según [6], los principales aspectos de la BD son:

- El método se centra en combinar dos o más soluciones del conjunto de referencia. La combinación de más de dos soluciones tiene como objetivo generar centroides.
- Generar soluciones en la línea que unen dos dadas se considera una forma reducida del método
- Al combinar se deben seleccionar pesos apropiados y no tomar valores al azar
- Se deben realizar combinaciones convexas y no convexas de las soluciones.
- La distribución de los puntos se considera importante y deben tomarse dispersos.

Según [1], se tiene el siguiente algoritmo:
x: solución al problema de optimización. Vector de n dimensiones. Cada xi puede ser una variable acotada real o entera.
F(**x**): función objetivo. Se puede obtener al correr el modelo de simulación que usa **x** como valor de sus parámetros de entrada.
Restricciones: restricciones de igual o desigualdad que se imponen sobre **x.**
El algoritmo empieza por generar una población inicial de puntos de referencia. La población inicial puede incluir puntos sugeridos por el usuario y siempre incluye el siguiente punto medio:

$$x_i = l_i + \frac{u_i - l_i}{2}$$

Donde u_i y l_i son las cotas superior e inferior de x_i, respectivamente. Se crean puntos adicionales con el objetivo de crear una población diversa. Una población se considera diversa si sus elementos son significativamente diferentes entre sí (determinar criterio de diferencia significativa). Se usa la medida de distancia para determinar cuán cerca está un potencial punto

nuevo de los puntos existentes en la población, para decidir si el punto es rechazado o incluido. Cada punto de referencia **x** está sujeto a una prueba de factibilidad antes de ser evaluado (antes de correr el modelo de simulación para determinar el valor de F(**x**)). La prueba de factibilidad consiste en probar que se cumplan una por una las restricciones establecidas. Un punto **x** no factible se hace factible al formular y resolver un problema de programación lineal. El problema de programación lineal consiste en encontrar un punto factible **x*** que minimice la desviación absoluta entre **x** y **x***.

El tamaño de la población se ajusta automáticamente al sistema considerando el tiempo que se requiere completar una evaluación de F(**x**) y el tiempo límite que el usuario ha establecido para la búsqueda. Una vez se ha generado la población, e procedimiento itera en la búsqueda de resultados mejorados. En cada iteración, se seleccionan dos puntos de referencia para crear cuatro 'hijos'. Sean los puntos **x1** y **x2** los puntos de referencia padres y **x3** hasta **x6** los puntos 'hijos' que se hallan como sigue:

$$\mathbf{x_3} = \mathbf{x_1} + d$$
$$\mathbf{x_4} = \mathbf{x_1} - d$$
$$\mathbf{x_5} = \mathbf{x_2} + d$$
$$\mathbf{x_6} = \mathbf{x_2} - d$$

Donde $d = (\mathbf{x_1} - \mathbf{x_2})/3$

La selección de $\mathbf{x_1}$ y $\mathbf{x_2}$ influida por los valores de F($\mathbf{x_1}$) y F($\mathbf{x_2}$) y por las funciones de memoria de la BT. Una iteración termina al reemplazar el peor padre por le mejor hijo y dar al padre sobreviviente una estatus de tabú por un número dado de iteraciones. En iteraciones siguientes el uso de dos padres tabú está prohibido.

En la búsqueda de un óptimo global, la población puede contener puntos de referencia con características similares. Esto es, en el proceso de generación de hijos de la unión de puntos de referencia de alta calidad y puntos de referencia ordinarios de la población, la diversidad de la población puede tender a decrecer. Una estrategia para resolver esta situación es la creación de una nueva población. Así, la implementación de un mecanismo de reinicio tiene el objetivo de crear una población que es una mezcla de puntos de alta calidad (puntos élite) encontrados en exploraciones previas y puntos generados de la misma manera como la fase inicial. El procedimiento de reinicio inyecta diversidad a través de nuevos puntos generados y preserva la calidad a través de la inclusión de puntos élite.

Algunos de los puntos de la población inicial pueden tener valores pobres de la función objetivo, por esto, puede que nunca se elijan para ser padres y estarán en la población hasta que se reinicie. Para dar más diversidad a la búsqueda se incremente la 'atracción' de estos puntos poco usados en el tiempo; esto se hace usando una forma de memoria a largo plazo que es diferente a la implementación convencional basada en la frecuencia. En particular, se introduce la noción de 'edad' que define una medida de 'atracción' basada en la edad y el valor de la función objetivo de un punto en particular. La idea es usar la historia de la búsqueda para hacer que los puntos de

referencia no usados como padres sean atractivos al modificar sus valores de la función objetivo de acuerdo con su edad.
Al comienzo del proceso de búsqueda todos los puntos de referencia **x** en la población de tamaño *p* tiene edad cero. Al final de la primera iteración habrá *p-1* puntos de referencia de la población original y un nuevo hijo, y las edades de los puntos de referencia se cambian por uno; y la del hijo, por cero. El proceso se repite para las iteraciones siguientes.

Simulación optimización de problemas de inventarios

Uno de los problemas tratados con la simulación son los problemas de inventarios, en donde se quiere observar el comportamiento del inventario pero en especial de los costos asociados a mantener, almacenar y utilizar este inventario. La simulación de inventarios es importante el área de administración, economía e industria pues el control de los sistemas de inventarios es fundamental para garantizar un buen nivel de servicio y unos costo de inventario razonables.
Los costos del inventario se pueden representar por funciones matemáticas y el objetivo al simular y optimizar este tipo de sistemas es encontrar una cantidad óptima de pedido y un punto de reorden óptimo (según el problema), factores que influyen fuertemente en los costos totales del inventario y que en ocasiones es difícil hallar sus valores óptimos por la complejidad del sistema de inventarios. Dada esta complejidad presenta en algunos sistemas, la simulación optimización se levanta con una opción importante para resolver este tipo de problemas.

Caso de estudio

En el trabajo realizado por Kleijnen y Wan [3] se aplica simulación optimización a un problema de inventarios. En esta situación se tiene un inventario con una demanda aleatoria, este inventario se abastece de manera que cada vez que el nivel de inventario llega a un punto establecido, el punto de reorden, se hace un pedido de una cantidad determinada Q, que lleva a que se tenga un nivel máximo de inventario. Este pedido tiene un tiempo de entrega que puede ser constante o una variable aleatoria, y el nivel del inventario se conoce en todo momento pues se tiene revisión continua. Además se considera que si por alguna causa no se puede satisfacer la demanda, se tienen faltantes que serán cubiertos inmediatamente se tiene el pedido. Sin embargo, como estos faltantes son difíciles de determinar, se establece un nivel de servicio en la que se indica la cantidad de la demanda que se espera cumplir (por ejemplo, 90%). Dadas estas condiciones se tiene unos costos que pueden ser por mantener el inventario (costo fijo), costo de colocar una orden o pedido, costos por faltantes, entre otros. Estos costos pueden ser expresados por funciones matemáticas simples.
Según lo anterior, se plantea optimizar un sistema de inventarios con demanda aleatoria, tiempos de entrega aleatorios y una restricción de nivel de servicio. En esta optimización la función

objetivo es minimizar los costos totales de inventario, con el punto de reorden y el nivel máximo de inventario como variables de decisión, sujeto a la restricción de nivel de servicio.

La aleatoriedad de los tiempos de entrega puede implicar que en ocasiones se tengan pedidos cruzados, es decir, las órdenes no necesariamente se reciben en el orden en el que fueron colocadas. Esto hace que el análisis matemático sea más complicado y lleva a usar la simulación. Además, se entiende que el uso de métodos de fuerza bruta no es viable para problemas reales (por sus posibles dimensiones) y se encuentra que en estudios anteriores se reportan diferentes óptimos encontrados por este tipo de métodos, para el mismo caso de estudio.

En este trabajo se utiliza OptQuest para estimar los óptimos de las variables de decisión, se comparan los resultados con otro trabajos similares, se verificaron los resultados según las condiciones Karush – Kuhn –Tucker (KKT) y finalmente se hacen experimentos comparativos con resultados de otros trabajos para intentar establecer la eficiencia y eficacia de OptQuest.
Se encontró que OptQuest encuentra valores más cercanos a los óptimos reales y que la eficiencia de OptQuest depende en el tamaño del área de búsqueda y en la solución inicial sugerida por el usuario. Además se encontró que no todas las soluciones pasaron el test KKT y que existen otros métodos que encuentran soluciones similares en menos tiempo pero que requieren más conocimiento matemático por parte del usuario.

Este trabajo sirvió como base para elegir el ejemplo base a tratar y para delimitar el alcance del mismo. De esto, se pensó realizar un ejemplo más simple para empezar a trabajar en las herramientas elegidas.

Ejemplo 1

Inicialmente, se eligió un problema de inventarios EOQ (Cantidad económica de pedido por sus siglas en inglés), un modelo determinístico de revisión continua tomado del capítulo 19.3 de [4]. En este caso se supone que la demanda es una tasa constante conocida y se supone que el inventario de reabastece con una pedido de tamaño fijo Q en donde el pedido llega instantáneamente. Se supone revisión continua, es decir, se sabe en cualquier momento el nivel de inventario y no se admiten faltantes. En este caso se tiene un costo total compuesto por tres costos: costo de ordenar el pedido, costo de comprar cada unidad y el costo de mantener el inventario. Para cada uno de estos costos se tiene una constante asociada, K, c, h respectivamente. El objetivo al tratar este tipo de problemas consiste en determinar con qué frecuencia y en qué cantidad reabastecer el inventario de manera que se minimice la suma de los costos.
Como la tasa de demanda es fija, se pueden evitar los faltantes reabasteciendo cada vez que el inventario llegue a cero y así se minimiza el costo de mantener el inventario.

Así las cosas, se definen las siguientes funciones de costos, en donde un ciclo se define como el tiempo entre reabastecimientos consecutivos del inventario:

$$\text{costo por ordenar por ciclo} = K + cQ$$

$$\text{costo de mantener inventario por ciclo} = \frac{hQ^2}{2a}$$

$$\text{costo total por ciclo} = K + cQ + \frac{hQ^2}{2a}$$

$$\text{costo total por unidad de tiempo} = \frac{aK}{Q} + ac + \frac{hQ}{2}$$

donde

K es el costo de ordenar

c es el costo de producir o comprar cada unidad

h es elcosto de mantener el inventario por unidad por unidad de tiempo

a es la tasa de demanda

De la expresión de costo total por unidad por tiempo se puede encontrar un valor Q* que minimiza el costo total. Para esto, basta establecer la primera derivada e igualarla a cero, observando que la segunda derivada sea positiva. Así se encuentra

$$Q^* = \sqrt{\frac{2aK}{h}}$$

La curva de costo es bastante plana cerca al valor óptimo, por lo que cantidades similares a al óptimo se puede confundir con óptimos [4].

Según este sistema de inventarios se pensó en estudiar el ejemplo de modelo EOQ básico del capítulo 19.3 de [4]. Aunque este tipo de sistema de inventario permite realizar una optimización por medios matemáticos tradicionales (derivar la función de costos), se pensó en utilizar algún tipo de simulación para intentar representar el sistema y hallar un óptimo cercano o igual al reportado en el ejemplo. Dado que este tipo de sistema de inventarios es determinístico (la demanda no es variable) se implementó un ejemplo puntual en el software *PowerSim* de simulación continua. Como se ve en la figura 3, el inventario está representado por un nivel alimentado por la entrada de pedido Q y que disminuye por la demanda (ver figura 4). El resto de las variables y parámetros son para calcular los costos respectivos.

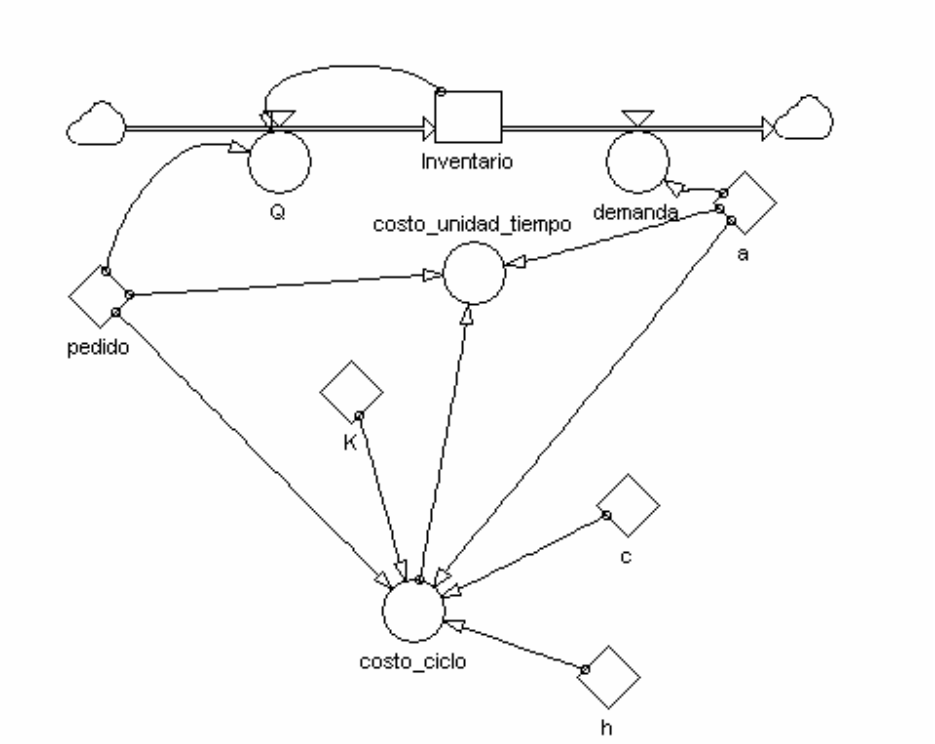

Figura 3: Diagrama de Forrester en PowerSim del ejemplo 1

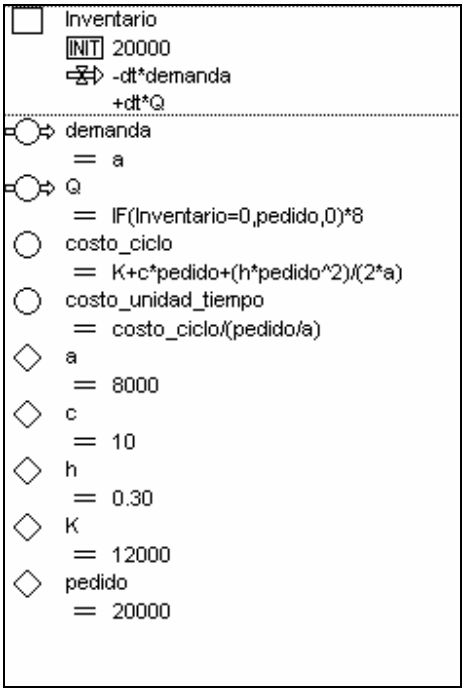

Figura 4: Ecuaciones del diagrama de Forrester del ejemplo1 en PowerSim

Al verificar el comportamiento del modelo, se observó que el nivel del inventario tenía el comportamiento adecuado (Figura 7), pues se ve un nivel de inventario que decrece en el tiempo, debido a la demanda constante, pero que en cuanto llega a cero, se reabastece instantáneamente con la cantidad de pedido establecida.

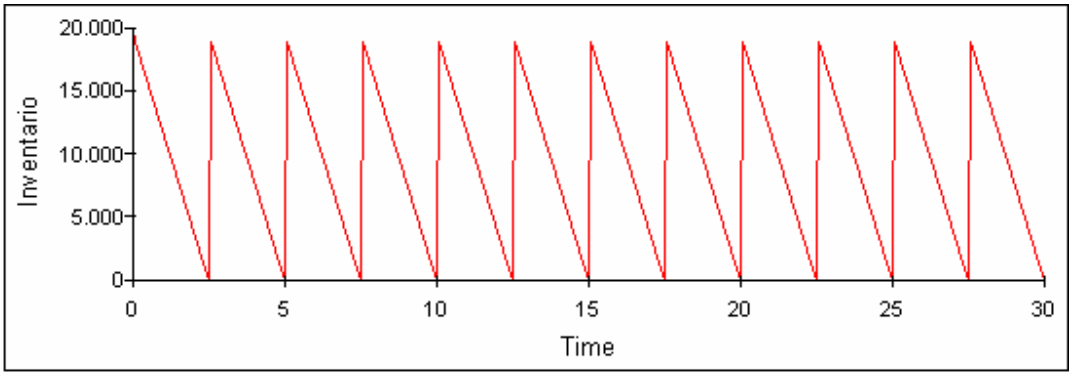

Figura 5: Inventario vs. Tiempo en la simulación en PowerSim del ejemplo 1

Sin embargo, en este ejemplo se consideraba fijo el tamaño de pedido Q y se utilizó la herramienta *Solver* de *PowerSim* para hallar el tamaño de pedido óptimo. El valor encontrado con esta herramienta fue igual que el valor óptimo reportado en [4].

A partir de esto, y con la idea de reproducir este resultado, se utilizó el software *MatLab* para programar la simulación realizada en *PowerSim* y se construyó un algoritmo heurístico para hallar el valor óptimo de Q. El código utilizado para esto se muestra a continuación.

```
function CT=simulacion(Q)
c=10;
K=12000;
h=0.30;
a=8000;
CT=((a*K)/Q)+(a*c)+((h*Q)/2);
NI=Q;
t=0;
dt=0.125;
i=2;
A(1,1)=NI;
while t<=30
   A(i,1)=A(i-1,1)-(a*dt);
   if A(i,1)<=0
      i=i+1;
      A(i,1)=NI;
   end
```

```
   t=t+dt;
   i=i+1;
end

function mejorsol=aleatoriaInv(valorMin,valorMax,tamano,iteraciones)

conjuntoelite=zeros(1,tamano);
costos=zeros(1,tamano);
%se llena el vector conjunto elite con tamano soluciones aleatorias
%(dentro del rango indicado) y se llena el vector costos con sus respectivos costos
for i=1:tamano
   conjuntoelite(i)=floor(valorMin+(valorMax-valorMin)*rand());
   costos(i)=simulacion(conjuntoelite(i));
end
% Durante un numero de iteraciones, se generan soluciones y si mejoran las
%existentes en el conjunto elite, se ingresan en el conjunto elite,
%asi se depura el conjunto, para que queden las mejores soluciones
for j=1:iteraciones
    nuevasol=floor(valorMin+(valorMax-valorMin)*rand());
    nuevocosto=simulacion(nuevasol);
   [costominimo,posicionMinimo] = min(costos);
  if nuevocosto<costominimo
      conjuntoelite(posicionMinimo)=nuevasol;
      costos(posicionMinimo)=nuevocosto;
   end
end
nuevomin=min(conjuntoelite);
nuevomax=max(conjuntoelite);
[mejorCosto,posicion]=min(costos);
mejorSolucion=conjuntoelite(posicion);
mejorsol=localInv(nuevomin,nuevomax,mejorSolucion,mejorCosto,iteraciones);

function
mejorSolucion=localInv(nuevomin,nuevomax,mejorSolucion,mejorCosto,iteraciones)
for j=1:iteraciones
   variacion=floor(1+(10-1)*rand());
   if rand()<0.5
      nuevasol=mejorSolucion-variacion;
   else
      nuevasol=mejorSolucion+variacion;
   end
   if nuevasol>=nuevomin && nuevasol<=nuevomax
      nuevocosto=simulacion(nuevasol);
      if nuevocosto<mejorCosto
         mejorSolucion=nuevasol;
         mejorCosto=nuevocosto;
```

```
        end
    end
end
```

El algoritmo que recrea la simulación recibe como parámetros una cantidad de pedido Q y, mediante la simulación, establece el costo asociado a esa cantidad de pedido. De esta forma, el algoritmo de simulación se asemeja a una función evaluadora de costos asociados con un pedido dado.

El heurísticos creado combina una búsqueda aleatoria con una búsqueda local.
Inicialmente, se realiza una búsqueda local en la que se tiene como parámetros de entada un rango posible para Q, valor mínimo y máximo (similar a OptQuest) y un número de iteraciones. Con este rango, se realiza una búsqueda aleatoria en la que se genera aleatoriamente un número determinado de soluciones y mediante la simulación se obtiene el costo asociado a ese tamaño de pedido. Así, se genera una muestra aleatoria del espacio de solución. Luego, durante el número de iteraciones dado como parámetro de entrada, se genera una nueva solución aleatoria y si resulta mejor que alguna de las que se tenían antes, se reemplaza. Así, se crea una muestra de soluciones, generadas aleatoriamente, pero que ya está más depurada y posiblemente ha estrechado el espacio de solución. Con este nuevo rango, se realiza una búsqueda local para explorar más detalladamente el nuevo espacio. En esta búsqueda local se encuentra una solución cercana la que hasta ese momento se considera mejor y si la solución cercana es mejor, se convierte en la mejor solución hasta ese momento. De esta manera se buscan soluciones cercanas a las mejores encontradas en el paso anterior, intentando encontrar soluciones aún mejores. Finalmente, luego de estos dos procesos, se tiene una cantidad de pedido Q que genera el menor costo, entre los explorados.

Como ya es claro, este algoritmo heurístico puede generar soluciones buenas o soluciones no tan buenas y ante parámetros de entrada iguales, puede generar soluciones diferentes. Teniendo en cuenta eso, se tuvo especial cuidado al establecer los parámetros de entrada, logrando unos que dieran una solución buena y que ésta se pudiera encontrar varias veces (no que fuera por cuestiones de “azar”). Haciendo esto, se pudo encontrar y replicar exactamente el mismo resultado encontrado teóricamente, que fue también encontrado con el *Solver* de PowerSim.

Ejemplo 2

Con la intensión de estudiar un sistema de inventarios más complejo, se pensó en revisar el modelo probabilístico de cantidad económica de pedido, es este caso, se toma del capítulo 16.1.2 de [5]. En este sistema de inventarios se tiene una demanda aleatoria y se quiere determinar una cantidad de pedido Q y el punto de reorden, nivel del inventario en el que se pedirá la cantidad Q. Además se considera que el tiempo entre la colocación del pedido y la recepción del mismo (tiempo de entrega) es constante. Dada la naturaleza aleatoria de la demanda, en este modelo pueden existir faltantes pues puede suceder que en el tiempo de entrega haya un crecimiento inesperado de la demanda y no se puedan satisfacer algunos pedidos. Si esto sucede, los faltantes serán satisfechos en el momento en el que llegue el pedido. El modelo tiene los siguientes supuestos, según [5]:

1. La demanda no satisfecha durante el tiempo de entrega se acumula,
2. No se permite más de un pedido vigente,
3. La distribución de la demanda durante el tiempo de entrega permanece estacionaria con el tiempo.

Para la función de costos se tienen los siguientes parámetros

$f(x)$: función de distribución de probabilidad de la demanda x durante el tiempo de entrega

D: demanda esperada por unidad de tiempo

h : costo de almacenamiento por unidad de inventario y por unidad de tiempo

p : costo faltante por unidad de inventario

K : costo de preparación por pedido

Q : cantidad de pedido

R : punto de reorden

Igualmente se definen costos de preparación, almacenamiento y faltantes. Los costos de preparación tienen en cuenta la cantidad aproximada de pedidos y el costo del pedido; los de almacenamiento tienen en cuenta el costo de almacenamiento por unidad por unidad de tiempo y el inventario promedio; y los de faltantes, la cantidad esperada de faltantes y el costo por faltante por unidad por unidad de tiempo.

Con los parámetros anteriores se definen estos costos como sigue:

$$\text{costo de preparación} = \frac{KD}{Q}$$

$$\text{costo de almacenamiento} = h\left(\frac{Q}{2} + R - E\{x\}\right)$$

$$\text{costo esperado por faltantes} = \frac{pD}{Q}\int_{R}^{\infty}(x-R)f(x)dx$$

$$\text{costo total por unidad de tiempo} = \frac{KD}{Q} + h\left(\frac{Q}{2} + R - E\{x\}\right) + \frac{pD}{Q}\int_{R}^{\infty}(x-R)f(x)dx$$

Las soluciones óptimas para Q* y R* no se pueden determinar analíticamente a partir de la función de costo total, por esto, en [5] se utiliza un algoritmo numérico para determinar las soluciones.
Dado este sistema de inventarios y el ejemplo 6.1-2 de [5], se quiso realizar un modelo de simulación discreta para representar este sistema con los parámetros establecidos en el ejemplo. Se usó simulación discreta por la aleatoriedad de la demanda y la necesidad de realizar múltiples corridas para analizar el sistema.
El modelo fue construido en SIMUL8, utilizando una cola, que representa el inventario y un centro de trabajo que representa la demanda y una salida que representa la demanda satisfecha (ver Figura 6).

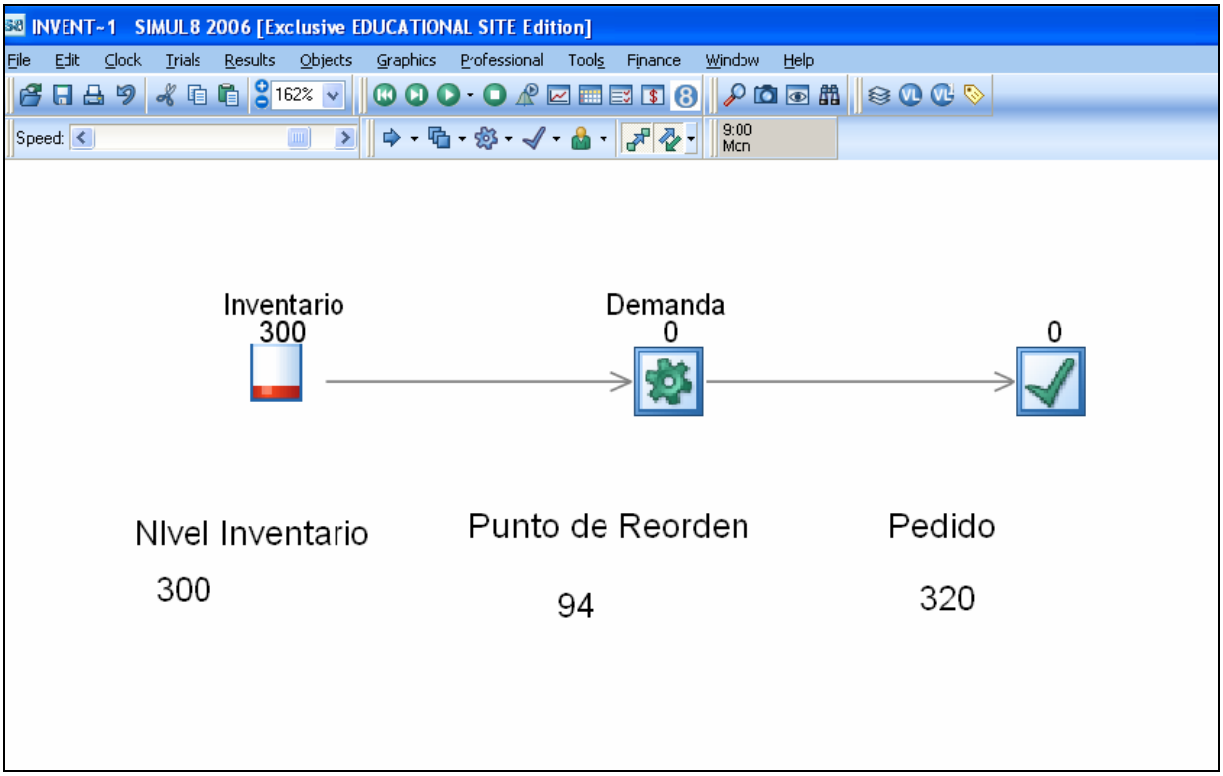

Figura 6: modelo del ejemplo2 en SIMUL8

Para el cálculo de los costos se tiene una redefinición de las fórmulas anteriores en el ambiente de la simulación. El costo de almacenamiento será calculado multiplicando *h* por el nivel de inventario en determinada momento, es decir el contenido de la cola que representa al inventario en la simulación. El costo por faltantes se calcula multiplicando pD/Q por la cantidad de faltantes que se tiene en determinado momento. Para calcular estas funciones dentro de la simulación y definir comportamiento específicos del modelo, se programó un código en Visual Logic de Simul8 (ver Figura 7).

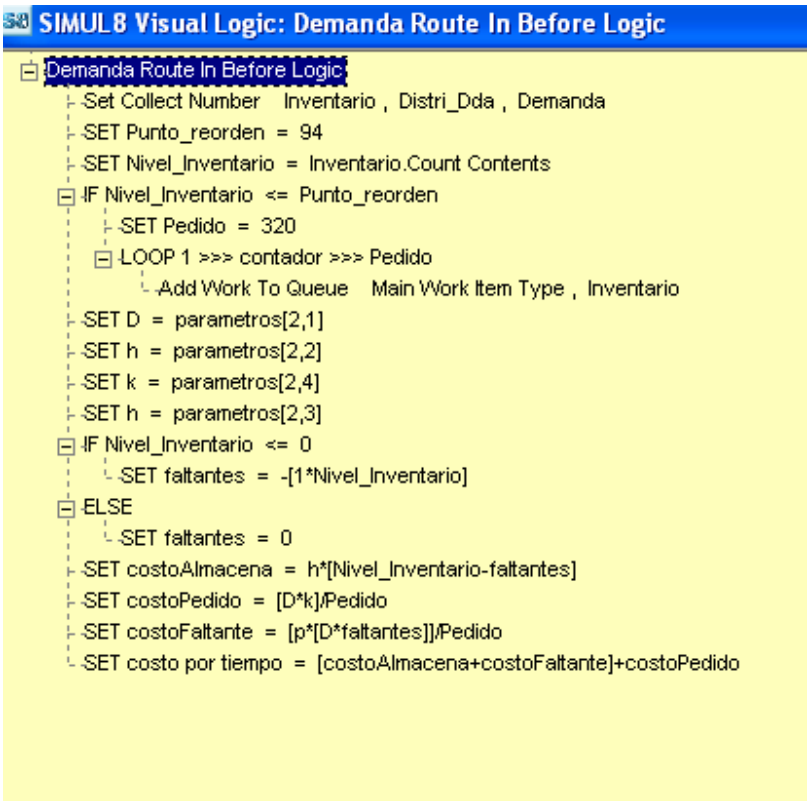

Figura 7: Código en Visual Logic de SIMUL8
para el modelo del ejemplo 2

Con el modelo de Simul8 se verificó el comportamiento del nivel de inventario en el tiempo (Figura 8), y se puede ver que cumple el comportamiento esperado, aunque difiere del comportamiento del ejemplo 1, esta diferencia se da por la variabilidad de la demanda en el segundo ejemplo.

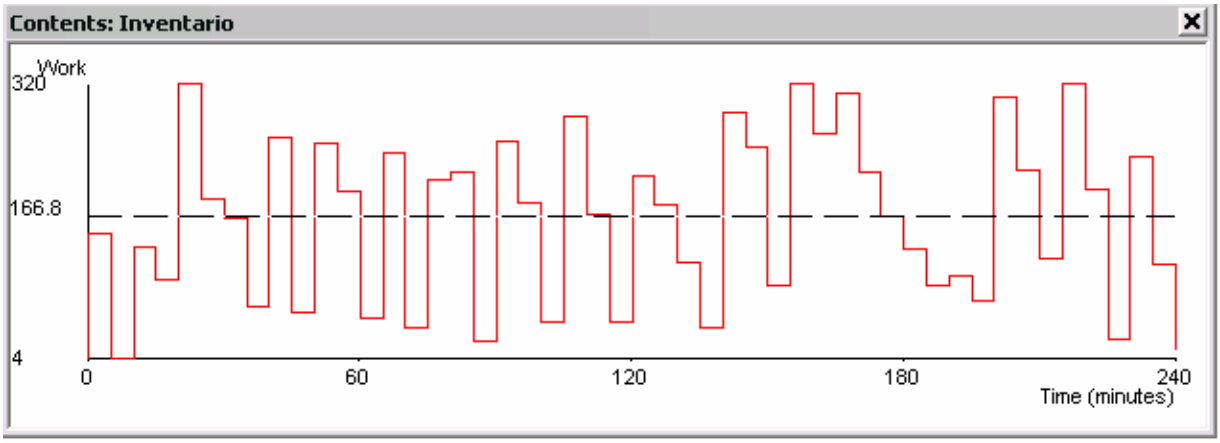

Figura 8: Gráfica del inventario en el tiempo en el modelo en SIMUL8 del ejemplo 2

Una vez se construyó el modelo en SIMUL8 se optimizó por medio de OptQuest tomando como variables de decisión el pedido Q y el punto de reorden R (ambas variables globales en el modelo de simulación), buscando minimizar el costo total (también una variable global). A construir el modelo en SIMUL8 hubo que relajar algunos supuestos del modelo y se tuvo inconvenientes con el manejo de los faltantes, lo que influyó en el cálculo del costo total de inventario. De la optimización en OptQuest se encontró soluciones aproximadas al óptimo que, por las razones antes mencionadas, no coincidieron con el óptimo reportado en [5]. Esto indica que le modelo de SIMUL8 debe ser depurado y ajustado al modelo teórico, para tener mejores resultados en la simulación optimización. Sin embargo, este ejercicio fue importante para aproximarse a las aplicaciones de la simulación optimización con SIMUL8 y OptQuest, y entender sus ventajas y dificultades.

Conclusiones

En este trabajo se hizo una revisión bibliográfica sobre simulación optimización y se encontró que existe mucha información pero que hay aspectos importantes sobre la que se omiten y se quiso reproducir ejemplos para revelar todo el proceso. Aunque se quería llegar a reproducir un optimizador genérico se encontró que hay poca información y dificultades técnicas para llevar a cabo el proceso, se piensa que por razones comerciales hay canales de comunicación entre software que están bloqueados a un usuario común y corriente. Es por esto que no se pudo establecer la comunicación adecuada ente SIMUL8 y Visual Basic para reproducir el optimizador. Sin embargo, queda como trabajo futuro intentar encontrar caminos diferentes para lograr este objetivo.

Dadas la dificultades anteriores se optó por reproducir ejemplos más sencillo y se logró, mediante heurísticos sencillos y simulación implementada en MatLab, encontrar óptimos a un problema de inventarios de tipo EOQ. Esto muestra que existen varias formas de aplicar simulación optimización a diferentes tipos de problemas, obteniendo muy buenos resultados.

Se implementó un ejemplo de inventarios en SIMUL8 y se optimizó con OptQuest, dando buenos resultados respecto a los reportados en la literatura. Este proceso sirvió para entender un poco más el comportamiento de ambas herramientas y compararlas con otro tipo de herramientas y resultados. Se encontró que a través de la simulación optimización se pueden optimizar problemas no aptos para optimización tradicional y simular problemas muy complejos para optimización.

Se considera que el estudio de la simulación optimización es importante ya que es una herramienta que está tomando auge a nivel mundial y que es poco conocida en nuestro país. Con

este estudio se está empezando el camino de apropiación de esta herramienta para su posible aplicación a casos reales y para difundir la utilización de esta importante herramienta.

Referencias

[1] Glover, F., Kelly, J., Laguna, M., *"The Optquest Approach to Crystal Ball Simulation Optimization"*.

[2] Pichitlamken, J., Nelson, B. *"A Combined Procedure for Optimization via Simulation"*. Proceeding of the 2002 Winter Simulation Conference.

[3] Kleijnen, J.P.C., Wan, J., "Optimization of smulated systems: OptQuest and alternatives". Simulation Modelling Practice and Theory 15 (2007) 354–362. Elsevier.

[4] Hillier, F., Lieberman, G. "Investigación de Operaciones". Ed. Mc Graw Hill. Séptima Edición, 2002.

[5] Taha, H. "Investigación de Operaciones". Ed. Pearson-Prentice Hall. Séptima Edición, 2004.

[6] Martí, R., Laguna, M. "Búsqueda Dispersa".

[7] Laguna, M., Martí, R. "The OptQuest Callable Library" to appear in *Optimization Software Class Libraries*, Stefan Voss y Davis Woodruff (eds.), Kluwer Academic Publishers, Boston, EEUU.

[8] Díaz, A., Glover, F., Ghaziri, H., González, J., Laguna, M., Moscazo, P., Tseng, F. "Optimización Heurística y Redes Neuronales". Ed. Paraninfo.

[9] www.opttek.com

ERROR: undefined
OFFENDING COMMAND:

STACK: