

Incorporación de un Parámetro Fractal al Algoritmo Genético en Problemas NP

Mayra Alejandra Céspedes Fernández

Universidad del Valle

Noviembre 2008

1 Introducción

2 Generadores de Problemas

- Generalidades
- TSP
- K-SAT
- Knapsack
- Automatización

3 Algoritmos Genéticos

- Generalidades
- Funciones de Aptitud
- AGM

4 AG vs. AGM

- Pruebas

Motivación

Problemas a tratar

En este proyecto de grado, se estudiarán los problemas TSP, K-SAT y Knapsack, pertenecientes a la clase NP-Completo.

Motivación

Problemas a tratar

En este proyecto de grado, se estudiarán los problemas TSP, K-SAT y Knapsack, pertenecientes a la clase NP-Completo.

Parámetro a utilizar

En el campo de la Matemática y la Física existe un fenómeno denominado *Percolación* el cual describe, para un sistema, la transición de un estado en otro.

Motivación

Problemas a tratar

En este proyecto de grado, se estudiarán los problemas TSP, K-SAT y Knapsack, pertenecientes a la clase NP-Completo.

Parámetro a utilizar

En el campo de la Matemática y la Física existe un fenómeno denominado *Percolación* el cual describe, para un sistema, la transición de un estado en otro.

Caso de Estudio

Se buscará si hay fenómenos similares a la percolación en los problemas intratables, cuando se intentan resolver con algoritmos genéticos, para entender cuán difícil es el problema concreto.

Motivación

Base

La idea surge al leer el libro *Investigations* del Biólogo Stuart Kauffman. Se plantea una forma de navegar por el espacio de soluciones de un problema complejo -Job Shop Scheduling-, basado en transiciones de fase.

Motivación

Base

La idea surge al leer el libro *Investigations* del Biólogo Stuart Kauffman. Se plantea una forma de navegar por el espacio de soluciones de un problema complejo -Job Shop Scheduling-, basado en transiciones de fase.

Objetivo General

El propósito de este proyecto es caracterizar la aplicación de la técnica de Percolación a problemas de tipo NP.

Objetivos

Objetivos Específicos

- 1 Aplicar la técnica de Percolación.
- 2 Determinar la dimensión fractal de los problemas Traveling Salesman Problem, K-SAT y Knapsack.
- 3 Determinar qué correlación existe entre la dificultad de un problema y el grado de percolación del mismo, aplicándolo a los problemas: Traveling Salesman Problem, K-SAT y Knapsack.
- 4 Desarrollar un algoritmo genético que permita aplicar de forma sencilla la técnica de percolación para controlar la presión selectiva.
- 5 Comparar los resultados obtenidos aplicando la técnica de percolación y cuando no se aplica.

Contenido

- 1 **Introducción**
- 2 **Generadores de Problemas**
 - Generalidades
 - TSP
 - K-SAT
 - Knapsack
 - Automatización
- 3 **Algoritmos Genéticos**
 - Generalidades
 - Funciones de Aptitud
 - AGM
- 4 **AG vs. AGM**
 - Pruebas

Estructura

Los generadores de instancias de cada uno de los problemas a tratar en este proyecto se programaron en el lenguaje **Python**.

► Sintaxis

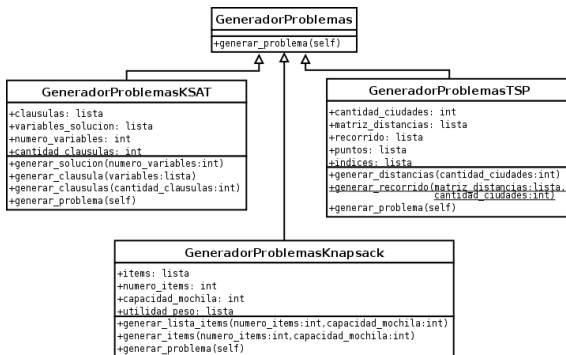


Figura: Diagrama de Clases

Contenido

1 Introducción

2 Generadores de Problemas

- Generalidades
- TSP
- K-SAT
- Knapsack
- Automatización

3 Algoritmos Genéticos

- Generalidades
- Funciones de Aptitud
- AGM

4 AG vs. AGM

- Pruebas

Definición

El TSP es uno de los problemas más famosos en el campo de la optimización combinatoria computacional.

Planteamiento del Problema

Sean n ciudades de un territorio. Se debe encontrar una ruta de manera que:

- empezando y terminando en una ciudad concreta,
- pase una sola vez por cada ciudad,
- y minimice la distancia recorrida por el viajante.

Configuración

Para cada ciudad se generan puntos (x, y) en el plano, ubicados de tal manera que formen figuras geométricas como se muestra a continuación:

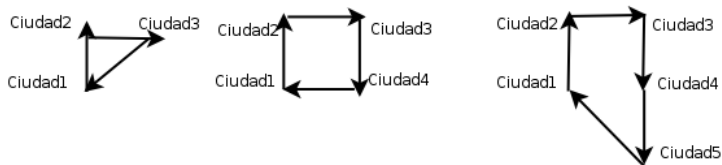


Figura: Formas Geométricas

Configuración

Problema

Se calculan las distancias entre cada par de ciudades, formando con ello una matriz diagonal:

	A	B	C	D	E
D :	0	2	2.82	2	2.82
	2	0	2	2.82	4.47
	2.82	2	0	2	4
	2	2.82	2	0	2
	2.82	4.47	4	2	0

Configuración

Problema

Se calculan las distancias entre cada par de ciudades, formando con ello una matriz diagonal:

	A	B	C	D	E
$D:$	0	2	2.82	2	2.82
	2	0	2	2.82	4.47
	2.82	2	0	2	4
	2	2.82	2	0	2
	2.82	4.47	4	2	0

Solución

En este caso, la lista [1, 2, 3, 4, 5] es la solución óptima y la suma del recorrido es: 10.83.

Contenido

- 1 **Introducción**
- 2 **Generadores de Problemas**
 - Generalidades
 - TSP
 - **K-SAT**
 - Knapsack
 - Automatización
- 3 **Algoritmos Genéticos**
 - Generalidades
 - Funciones de Aptitud
 - AGM
- 4 **AG vs. AGM**
 - Pruebas

K-SAT

Una variable booleana es una variable que solo puede tomar los valores *Verdadero* o *Falso*.

Una fórmula booleana es una expresión formada con variables booleanas y los operadores $\vee$ (disyunción), $\wedge$ (conjunción) y $\neg$ (negación). Por ejemplo, la fórmula $(x_1 \vee x_2) \wedge (x_2 \vee \neg x_3)$ es una fórmula booleana en FNC.

K-SAT

Una variable booleana es una variable que solo puede tomar los valores *Verdadero* o *Falso*.

Una fórmula booleana es una expresión formada con variables booleanas y los operadores $\vee$ (disyunción), $\wedge$ (conjunción) y $\neg$ (negación). Por ejemplo, la fórmula $(x_1 \vee x_2) \wedge (x_2 \vee \neg x_3)$ es una fórmula booleana en FNC.

Planteamiento del Problema

- INSTANCIA: una colección $C_1, C_2, \dots, C_m$ de cláusulas de un conjunto finito de variables U .
- PREGUNTA: ¿Existe una asignación para U que satisfaga todas las cláusulas de C ?

Configuración

Solución

Se generan valores aleatorios (0, 1) y se insertan en una lista. Para un problema de 5 variables una posible solución es [1,0,1,1,1]. A partir de dicha lista, se generan las cláusulas.

Codificación

Las cláusulas del problema se codifican de la siguiente manera:

- **0:** la variable se encuentra negada
- **1:** la variable se encuentra normal
- **2:** la variable no se encuentra en la cláusula

Configuración

Generación de cláusulas

- Se generan 3 posiciones aleatorias distintas.
- Para cada posición se genera un valor aleatorio (0, 1).
- El resto de posiciones se llenan con el número 2.

Configuración

Generación de cláusulas

- Se generan 3 posiciones aleatorias distintas.
- Para cada posición se genera un valor aleatorio (0, 1).
- El resto de posiciones se llenan con el número 2.

Cláusula

$$C = [p_1, p_2, p_3, p_3, p_5]$$

Configuración

Generación de cláusulas

- Se generan 3 posiciones aleatorias distintas.
- Para cada posición se genera un valor aleatorio (0, 1).
- El resto de posiciones se llenan con el número 2.

Cláusula

$$C = [p_1, p_2, p_3, p_3, p_5]$$

Cláusula

$$C = [0, p_2, 1, 1, p_5]$$

Configuración

Generación de cláusulas

- Se generan 3 posiciones aleatorias distintas.
- Para cada posición se genera un valor aleatorio (0, 1).
- El resto de posiciones se llenan con el número 2.

Cláusula

$$C = [p_1, p_2, p_3, p_3, p_5]$$

Cláusula

$$C = [0, p_2, 1, 1, p_5]$$

Cláusula

$$C = [0, 2, 1, 1, 2] \rightarrow (\neg x_1 \vee x_3 \vee x_4)$$

Contenido

- 1 **Introducción**
- 2 **Generadores de Problemas**
 - Generalidades
 - TSP
 - K-SAT
 - **Knapsack**
 - Automatización
- 3 **Algoritmos Genéticos**
 - Generalidades
 - Funciones de Aptitud
 - AGM
- 4 **AG vs. AGM**
 - Pruebas

Knapsack

Supóngase que se tiene n objetos distintos y un recipiente. Cada uno de los objetos tiene asociado un peso positivo w_i y un valor positivo v_i para $i = 1, 2, \dots, n$. El recipiente no puede llevar un peso que sobrepase la cantidad W .

Knapsack

Supóngase que se tiene n objetos distintos y un recipiente. Cada uno de los objetos tiene asociado un peso positivo w_i y un valor positivo v_i para $i = 1, 2, \dots, n$. El recipiente no puede llevar un peso que sobrepase la cantidad W .

Planteamiento del Problema

Se debe llenar el recipiente de forma que:

- se maximice el valor de los objetos transportados en éste
- y la suma de los pesos de los objetos no sobrepase la capacidad W del recipiente.

Configuración

Codificación

Para cada ítem:

- Se genera aleatoriamente un peso
- Se le asigna un valor, potencia de 2, que representa la utilidad
- Cada tupla (utilidad, peso) se inserta en una lista ordenada por utilidad

Configuración

Codificación

Para cada ítem:

- Se genera aleatoriamente un peso
- Se le asigna un valor, potencia de 2, que representa la utilidad
- Cada tupla (utilidad, peso) se inserta en una lista ordenada por utilidad

Lista de ítems

$$lista = [(2^n, peso_n), (2^{n-1}, peso_{n-1}), \dots, (2^0, peso_0)]$$

Contenido

1

Introducción

2

Generadores de Problemas

- Generalidades
- TSP
- K-SAT
- Knapsack
- Automatización

3

Algoritmos Genéticos

- Generalidades
- Funciones de Aptitud
- AGM

4

AG vs. AGM

- Pruebas

Configuración

Interfaz

Se realizó una interfaz, usando **PyGTK**, que permite:

- ❶ Generar 1000 instancias de los problemas TSP, K-SAT, Knapsack
- ❷ Guarda en un archivo los problemas generados:
 - TSP: contiene la matriz de distancias
 - K-SAT: contiene las cláusulas del problema
 - Knapsack: contiene las utilidades y pesos de los ítems
- ❸ En otro guarda las soluciones generadas:
 - TSP: contiene el recorrido obtenido en el generador
 - K-SAT: contiene los valores (1, 0) de las variables del problema
 - Knapsack: contiene los valores (1, 0) de los ítems que se insertaron en el recipiente

Interfaz

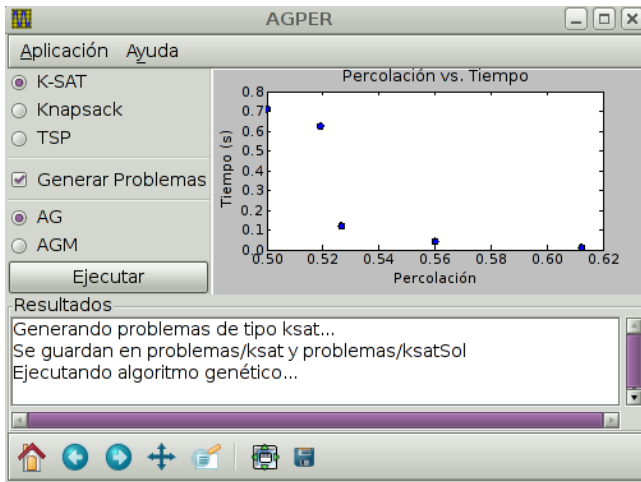


Figura: Generando problemas tipo K-SAT

Contenido

- 1 **Introducción**
- 2 **Generadores de Problemas**
 - Generalidades
 - TSP
 - K-SAT
 - Knapsack
 - Automatización
- 3 **Algoritmos Genéticos**
 - Generalidades
 - Funciones de Aptitud
 - AGM
- 4 **AG vs. AGM**
 - Pruebas

Funcionamiento

- 1 **Generar** una población de N cromosomas al azar.
- 2 **Evaluar** cada cromosoma utilizando la función de aptitud.
- 3 **Seleccionar** un subconjunto de cromosomas dependiendo de su aptitud.
- 4 **Reproducir** los cromosomas seleccionados utilizando los operadores de cruce y/o mutación.
- 5 **Reemplazar** los viejos cromosomas por los nuevos, manteniendo constante el tamaño de la población.
- 6 **Repetir** hasta encontrar una solución aceptable, hasta que haya convergido o hasta que haya alcanzado el número máximo de iteraciones.

Configuración

- 1 **Generar Población** Se define una población de 100 cromosomas.
- 2 **Cálculo de Percolación** Se hace el cálculo de percolación antes de iniciar el AG.
- 3 **Cálculo de Aptitudes**
- 4 **Selección** Se selecciona el 60 % de la población usando la técnica de la ruleta.
- 5 **Reproducción** Se aplican los operadores de cruce y mutación a parejas de cromosomas.
- 6 **Reemplazo** Se generan k posiciones aleatorias. El cromosoma que se encuentre en esa posición será reemplazado por uno nuevo.

Contenido

- 1 **Introducción**
- 2 **Generadores de Problemas**
 - Generalidades
 - TSP
 - K-SAT
 - Knapsack
 - Automatización
- 3 **Algoritmos Genéticos**
 - Generalidades
 - **Funciones de Aptitud**
 - AGM
- 4 **AG vs. AGM**
 - Pruebas

Cálculo de aptitudes

Para cada cromosoma de la población:

- 1 Se calculan las distancias del recorrido que representan
- 2 Se obtiene el menor valor (la menor distancia)
- 3 Se genera la calificación mediante la siguiente división:
$$\text{menorValor} / \text{distanciaRecorrido}[i]$$

Cálculo de aptitudes

Para cada cromosoma de la población:

- 1 Se calculan las distancias del recorrido que representan
- 2 Se obtiene el menor valor (la menor distancia)
- 3 Se genera la calificación mediante la siguiente división:
 $\text{menorValor} / \text{distanciaRecorrido}[i]$

Ejemplo

$\text{distRecorrido} = [20.34 \quad 15.85 \quad 10.83 \quad 30.43 \quad 18.57]$

$\text{menorValor} = 10.83$

$\text{aptitudes} = [0.53 \quad 0.68 \quad 1.0 \quad 0.35 \quad 0.58]$

Cálculo de aptitudes

Para cada cromosoma de la población:

- 1 Se evalúa cada cláusula mediante el operador **or**
- 2 Se obtiene el número de cláusulas satisfechas
- 3 Se genera la calificación mediante la siguiente división:
$$\text{clausulasSatisfechas} / \#totalClausulas$$

K-SAT

Cálculo de aptitudes

Para cada cromosoma de la población:

- 1 Se evalúa cada cláusula mediante el operador **or**
- 2 Se obtiene el número de cláusulas satisfechas
- 3 Se genera la calificación mediante la siguiente división:
$$\text{clausulasSatisfechas} / \#totalClausulas$$

Ejemplo

$\text{clausulasSatisfechas} = [10 \quad 8 \quad 1 \quad 3 \quad 9]$

$\text{totalClausulas} = 10$

$\text{aptitudes} = [1.0 \quad 0.8 \quad 0.1 \quad 0.3 \quad 0.9]$

Knapsack

Cálculo de aptitudes

Para cada cromosoma de la población:

- 1 Se calcula la utilidad total de cada cromosoma
- 2 Se obtiene el mayor valor (mayor utilidad)
- 3 Se genera la calificación mediante la siguiente división:
$$\text{utilidad}[i] / \text{mayorUtilidad}$$

Knapsack

Cálculo de aptitudes

Para cada cromosoma de la población:

- 1 Se calcula la utilidad total de cada cromosoma
- 2 Se obtiene el mayor valor (mayor utilidad)
- 3 Se genera la calificación mediante la siguiente división:
$$\text{utilidad}[i] / \text{mayorUtilidad}$$

Ejemplo

$\text{utilidades} = [250 \quad 800 \quad 150 \quad 300 \quad 450]$

$\text{mayorUtilidad} = 800$

$\text{aptitudes} = [0.3125 \quad 1.0 \quad 0.1875 \quad 0.375 \quad 0.5625]$

Contenido

- 1 **Introducción**
- 2 **Generadores de Problemas**
 - Generalidades
 - TSP
 - K-SAT
 - Knapsack
 - Automatización
- 3 **Algoritmos Genéticos**
 - Generalidades
 - Funciones de Aptitud
 - **AGM**
- 4 **AG vs. AGM**
 - Pruebas

Algoritmo Genético Modificado

Algoritmo

- 1 Se genera una población de 100 cromosomas
- 2 Se realiza el **cálculo de percolación**
- 3 Se calculan las aptitudes
- 4 Se realiza la **selección de cromosomas**
- 5 Se realiza la reproducción de los cromosomas seleccionados
- 6 Se hace el reemplazo de los cromosomas

Definición de Percolación

La Teoría de los procesos de percolación comenzó en 1957 cuando S. R. Broadbent y J. M. Hammersley introdujeron el proceso de *Percolación independiente*.

Plantemiento

Se planteó como un modelo probabilístico para el flujo de un fluido o un gas a través de un medio aleatorio.

- Fluido: líquido, vapor, flujo de calor, corriente eléctrica, infección, sistema solar o cualquier fluido que pueda moverse a través de un medio.
- Medio: puede ser el espacio poroso de una roca, un suelo, un arreglo de árboles o el mismo universo.

Cálculo de Percolación

Algoritmo

- 1 Se escogen k cromosomas de la población
- 2 Se generan pequeños cambios en cada una de sus N dimensiones, obteniendo con esto N nuevos cromosomas
- 3 Se obtienen los cromosomas factibles F
- 4 Se calculan las aptitudes los F cromosomas
- 5 Se comparan las F aptitudes con la aptitud del cromosoma para ver cuántos lo superaron (M)
- 6 Finalmente, la percolación será igual a: $P = M/N$

Algoritmo

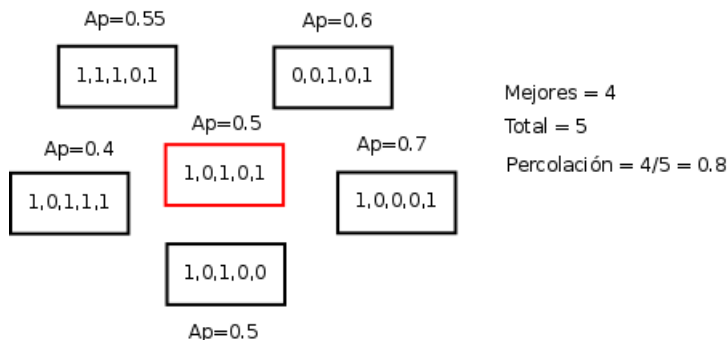


Figura: Cálculo de Percolación

Selección de Cromosomas

Algoritmo

- 1 Se calcula la percolación nuevamente
- 2 Se compara con la percolación calculada al inicio del algoritmo:
 - Menor: el porcentaje de selección se incrementa en 10 %(70 %)
 - Mayor: el porcentaje de selección se disminuye en 10 %(50 %)
 - Constante: el porcentaje de selección se mantiene en 60 %
- 3 Se realiza el tipo de selección ruleta

Contenido

- 1 **Introducción**
- 2 **Generadores de Problemas**
 - Generalidades
 - TSP
 - K-SAT
 - Knapsack
 - Automatización
- 3 **Algoritmos Genéticos**
 - Generalidades
 - Funciones de Aptitud
 - AGM
- 4 **AG vs. AGM**
 - Pruebas

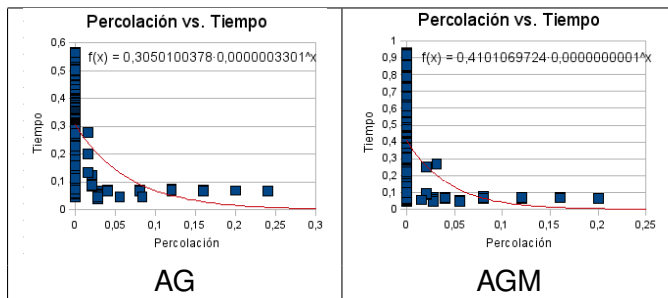


Figura: Percolación vs. Tiempo

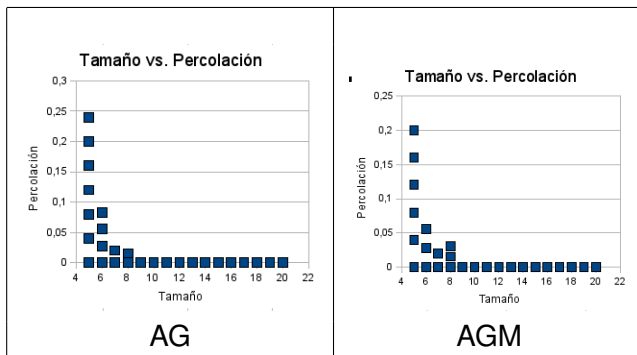


Figura: Tamaño vs. Percolación

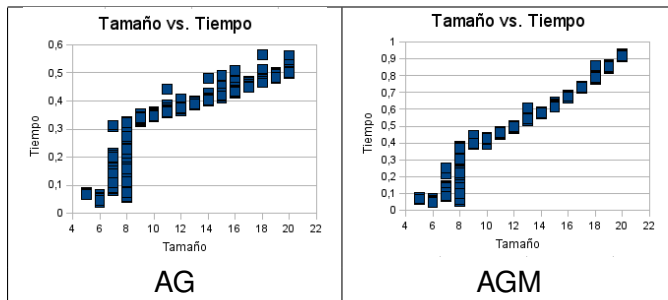


Figura: Tamaño vs. Tiempo

TSP Aleatorio

Se realizaron pruebas con una configuración aleatoria del problema TSP y se probaron en el AG.

TSP Aleatorio

Se realizaron pruebas con una configuración aleatoria del problema TSP y se probaron en el AG.

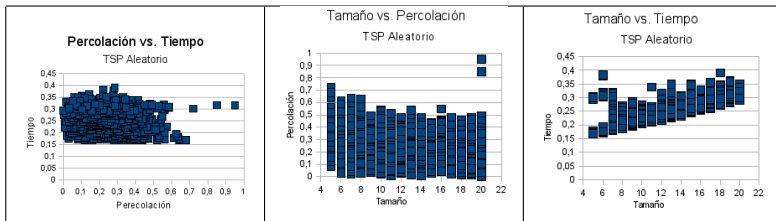


Figura: AG TSP Aleatorio

TSP Una Instancia

Adicionalmente se realizaron pruebas con una instancia particular del problema TSP y se probaron en el AGM.

TSP Una Instancia

Adicionalmente se realizaron pruebas con una instancia particular del problema TSP y se probaron en el AGM.

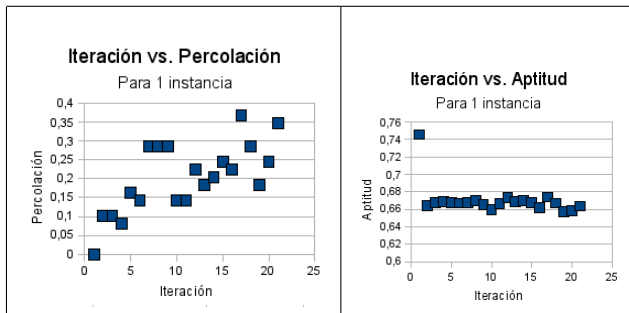


Figura: TSP Seguimiento 1 Instancia

Comparación AGM vs. AG

A pesar de que en AGM hubo un leve aumento en el tiempo de ejecución, se tiene lo siguiente:

- AGM encontró soluciones óptimas para todos los problemas de tamaños 5, 6 y 7. Algunas para tamaño 8 y muy pocas para 10.
- AG sólo pudo encontrar soluciones óptimas para todos los problemas de tamaños 5 y 6, muy pocas para 7.

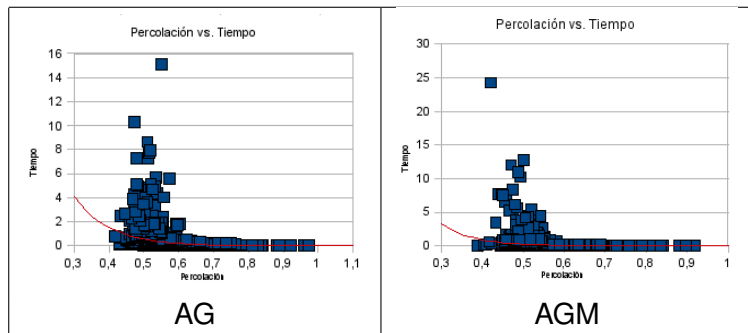


Figura: Percolación vs. Tiempo

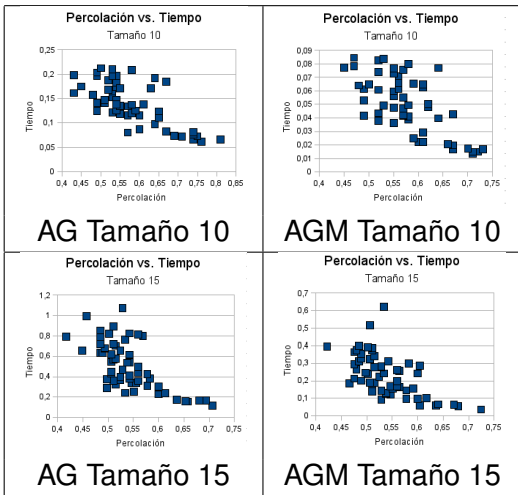


Figura: Percolación vs. Tiempo

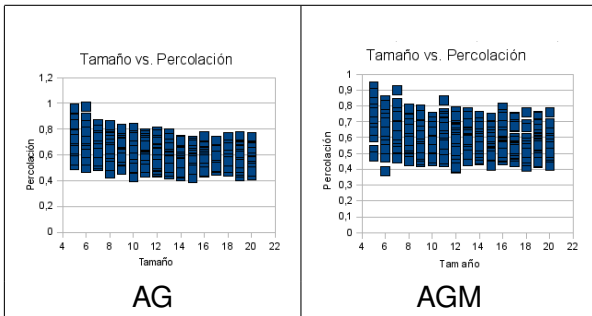


Figura: Tamaño vs. Percolación

K-SAT

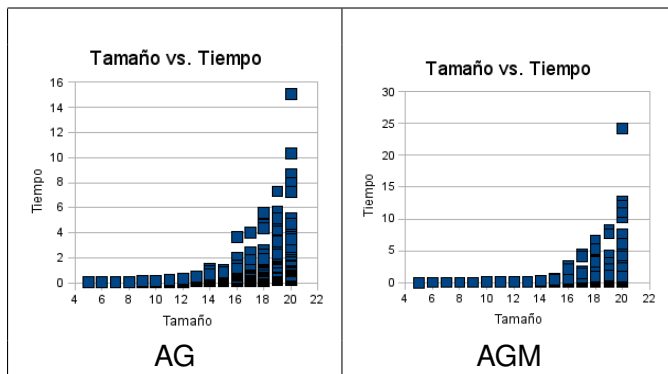


Figura: Tamaño vs. Tiempo

K-SAT

K-SAT Una Instancia

Adicionalmente se realizaron pruebas con una instancia particular del problema K-SAT y se probaron en el AGM.

K-SAT Una Instancia

Adicionalmente se realizaron pruebas con una instancia particular del problema K-SAT y se probaron en el AGM.

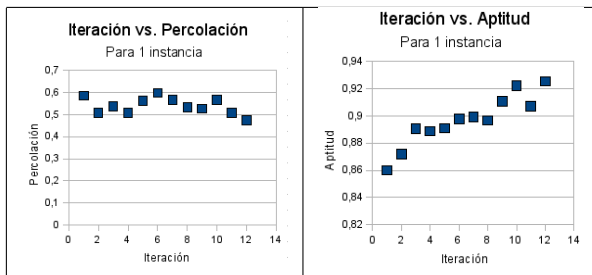


Figura: K-SAT Seguimiento 1 Instancia

Comparación AGM vs. AG

Al analizar las gráficas obtenidas se puede concluir que:

- El AGM supera al AG en cuanto a tiempos de ejecución
- Para algunos problemas de más de 15 variables el AGM puede tardar un poco más que el AG.

Knapsack

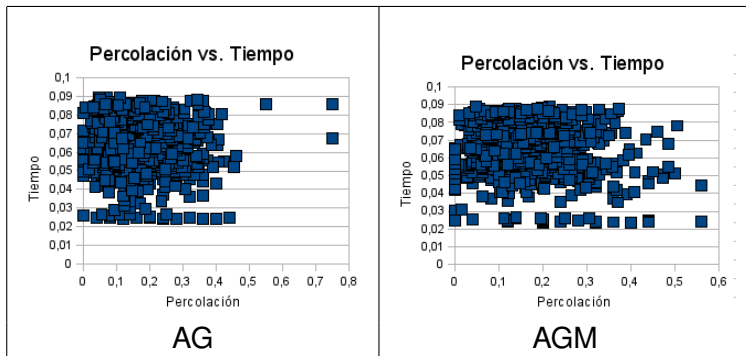


Figura: Percolación vs. Tiempo

Knapsack

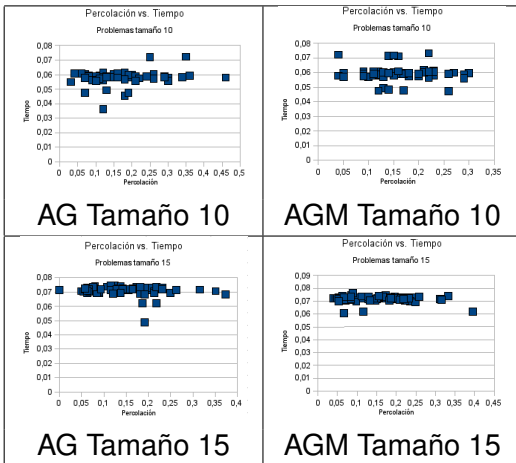


Figura: Percolación vs. Tiempo

Knapsack

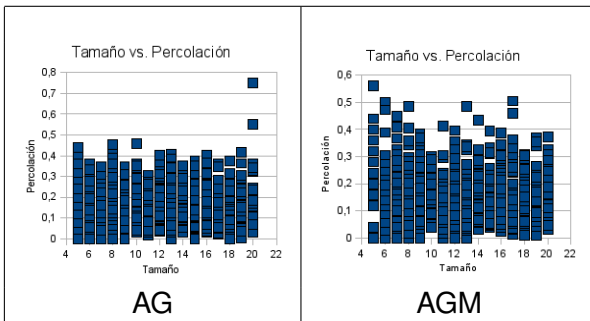


Figura: Tamaño vs. Percolación

Knapsack

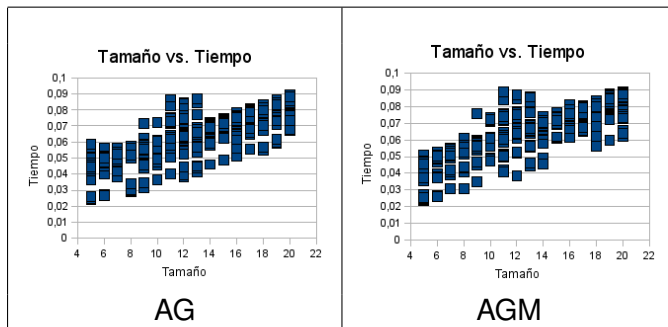


Figura: Tamaño vs. Tiempo

Knapsack

Knapsack Una Instancia

Adicionalmente se realizaron pruebas con una instancia particular del problema Knapsack y se probaron en el AGM.

Knapsack

Knapsack Una Instancia

Adicionalmente se realizaron pruebas con una instancia particular del problema Knapsack y se probaron en el AGM.

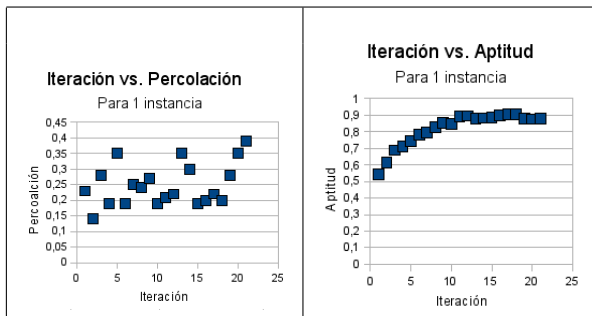


Figura: Knapsack Seguimiento 1 Instancia

Knapsack

Comparación AGM vs. AG

Tanto en el AG como en el AGM se observan tiempos de ejecución similares. En cuanto a la calidad de la solución encontrada se tiene lo siguiente:

- AG encuentra soluciones óptimas para problemas de tamaños 5 y 6, de ahí en adelante el algoritmo devuelve la solución aproximada.
- AGM puede encontrar soluciones óptimas para problemas de tamaños 5, 6, 7 y 8, superando al AG en ese sentido.

Resumen

- Se realizaron generadores de problemas para TSP, K-SAT y Knapsack, con el fin de automatizar el proceso de pruebas.

Resumen

- Se realizaron generadores de problemas para TSP, K-SAT y Knapsack, con el fin de automatizar el proceso de pruebas.
- Se realizaron los algoritmos genéticos para cada problema, enfocándose en el diseño de las funciones de aptitud.

Resumen

- Se realizaron generadores de problemas para TSP, K-SAT y Knapsack, con el fin de automatizar el proceso de pruebas.
- Se realizaron los algoritmos genéticos para cada problema, enfocándose en el diseño de las funciones de aptitud.
- Se hizo el cálculo del parámetro de percolación para cada problema.

Resumen

- Se realizaron generadores de problemas para TSP, K-SAT y Knapsack, con el fin de automatizar el proceso de pruebas.
- Se realizaron los algoritmos genéticos para cada problema, enfocándose en el diseño de las funciones de aptitud.
- Se hizo el cálculo del parámetro de percolación para cada problema.
- Se incluyó dentro del AG el cálculo de percolación y por medio de éste se controla la presión selectiva, obteniendo con esto un algoritmo genético modificado (AGM).

Resumen

- Se realizaron generadores de problemas para TSP, K-SAT y Knapsack, con el fin de automatizar el proceso de pruebas.
- Se realizaron los algoritmos genéticos para cada problema, enfocándose en el diseño de las funciones de aptitud.
- Se hizo el cálculo del parámetro de percolación para cada problema.
- Se incluyó dentro del AG el cálculo de percolación y por medio de éste se controla la presión selectiva, obteniendo con esto un algoritmo genético modificado (AGM).
- Se realizaron las pruebas con ambos algoritmos (AG y AGM).

Conclusiones y Recomendaciones

- Existe una relación entre Percolación y Dificultad del problema.
- Se puede implementar el AGM en la resolución de problemas NP y NP-Complejos.
- Se pueden investigar otras formas de cambiar el AG en función de la percolación, por ejemplo cambiando probabilidades de mutación o agregando otros operadores reproducción que produzcan cambios más bruscos.

Gracias

¡¡GRACIAS!!



PyRex

Los algoritmos genéticos se programaron en el lenguaje **PyRex**.

Ventajas

- Su sintaxis es similar a la de Python
- Su compilador genera código en lenguaje C
- Se contribuye a la comunidad de software libre

PyRex

Los algoritmos genéticos se programaron en el lenguaje **PyRex**.

Ventajas

- Su sintaxis es similar a la de Python
- Su compilador genera código en lenguaje C
- Se contribuye a la comunidad de software libre

Forma de Uso

- `pyrex [nombreArchivo].pyx`
- `gcc -I/usr/include/python2.5 -c [nombreArchivo].c`
- `gcc [nombreArchivo].o -fpic -shared -o [nombreArchivo].so`

Python

Los generadores de problemas se programaron en el lenguaje **Python**.

Ventajas

- Es un lenguaje interpretado
- Orientado a objetos
- Tipado dinámico

Python

Los generadores de problemas se programaron en el lenguaje **Python**.

Ventajas

- Es un lenguaje interpretado
- Orientado a objetos
- Tipado dinámico

Elementos Utilizados

- Listas -> *lista* = [x, y, z, ..]
- Tuplas -> *tupla* = (x, y)
- Listas de listas -> *listasL* = [[x, y, z], [a, b, c]]
- Listas de tuplas -> *listasT* = [(a, b), (c, d), (x, y)]