ICT-UNPA-133-2015
ISSN: 1852-4516
Aprobado por Resolución N° 1123/15-R-UNPA

Metaheurísticas de trayectoria y poblacional aplicadas a problemas de optimización combinatoria

Alancay, *N*[1]. **Villagra**, *S*[2]. **Villagra**, *A*[2].
[1]**{***Becaria de Investigación***}**
[2]**{***Docentes Investigadores de la UNPA***}**
alancaynatalia@hotmail.com, {svillagra, avillagra}@uaco.unpa.edu.ar

UNPA- UACO
Universidad Nacional de la Patagonia Austral - Unidad Académica Caleta Olivia
Departamento de Ciencias Exactas y Naturales
LabTEm- Laboratorio de Tecnologías Emergentes
Caleta Olivia, 2014

Resumen: En el mundo existen una multitud de problemas cotidianos que, precisan de una solución que cumpla con un conjunto de requisitos de la manera más apropiada maximizando o minimizando un determinado valor. Sin embargo, para encontrar una solución óptima para ciertos problemas de optimización puede ser una tarea increíblemente difícil o imposible. Esto es, porque cuando un problema se vuelve lo suficientemente grande, tenemos que buscar a través de un enorme número de posibles soluciones, la solución más eficiente, es decir, la que tiene costo menor. Las metaheurísticas son una de las estrategias que ha obtenido una gran aceptación y que han ido consiguiendo un importante cuerpo formal; ya que estás establecen estrategias para recorrer y explorar el espacio de soluciones del problema normalmente generadas en forma aleatoria y de manera iterativa. La principal ventaja de estas técnicas es su flexibilidad y robustez, lo que permite aplicarlas a un amplio conjunto de problemas. En este trabajo proponemos concentrarnos en una metaheurística basada en trayectoria *Simulated Annealing* y una basada en población Algoritmo Genético Celular con el objetivo de realizar un estudio y comparación de los resultados obtenidos en su aplicación para la resolución de un conjunto de problemas académicos de optimización combinatoria.

Palabras claves: Metaheurísticas; Algoritmo Genético Celular; *Simulated Annealing* y Problemas de Optimización Combinatoria.

1. Introducción

Existen áreas del conocimiento y del quehacer humano donde surgen problemas relacionados con la mejora de determinados procesos o la obtención de mejores soluciones (generales o particulares). Todos ellos son problemas que presentan una característica en común, donde tratan de optimizar (maximizar o minimizar) un determinado valor. Por esto, se llaman problemas de optimización. Los problemas de optimización pueden clasificarse en función de diferentes factores como son su complejidad, la existencia o no de restricciones, su carácter estático o dinámico, lineal o no lineal, mono-objetivo o multi-objetivo, etc. En cuanto a las técnicas de búsquedas, estas se pueden clasificar en función de si aseguran obtener el resultado óptimo (técnicas exactas) o si por el contrario permiten obtener soluciones cercanas al optimo (técnicas aproximadas).

Tomando en cuenta que la optimización combinatoria consiste en encontrar la mejor solución (óptima) de entre un conjunto finito de soluciones alternativas. La calidad u optimalidad de las soluciones vendrá definida por la capacidad de dichas soluciones para minimizar o maximizar una determinada función, denominada función objetivo, compuesta por un conjunto determinado de variables definidas sobre un conjunto discreto. Esta disciplina tiene numerosas aplicaciones en ámbito como las ciencias, la ingeniería, la industria, la logística, la administración de organizaciones, etc. Como ejemplo podemos mencionar, entre otros, el diseño de redes de telecomunicación, la asignación de tareas a procesadores, el enrutamiento y carga de vehículos en redes de distribución, la planificación de la producción, la selección de carteras financieras, la planificación de la generación de electricidad, el diseño de proteínas, la distribución de ambulancias en una región para asegurar un cierto nivel de servicio a su población, etc. La gran variedad de aplicaciones requieren la ayuda de optimizadores que han provocado un gran interés en las últimas décadas en el desarrollo de nuevos métodos.

En esta dirección, si bien las técnicas exactas garantizan la obtención de la solución para cualquier tipo de problema, requieren un costo computacional elevado, es decir, que para obtener la mejor solución, el tiempo necesario crece exponencialmente con el tamaño del problema y en algunos casos es imposible encontrarla por el tiempo que demanda. Esto ha volcado el interés hacia las técnicas aproximadas que permiten obtener una solución “casi” óptima en un tiempo razonable; en particular al uso de las metaheurísticas que se pueden definir como (Glover y Kochenberger 2003) estrategias inteligentes generales para diseñar o mejorar procedimientos heurísticos para la resolución de problemas con un alto rendimiento. Dentro de los algoritmos no exactos podemos encontrar tres tipos: los heurísticos constructivos (también llamado voraces), los métodos de búsqueda local (o métodos de seguimientos del gradiente) y las metaheurísticas. Existen diferentes formas de clasificar y describir las técnicas metaheurísticas (Crainic y Toulouse 2003), dependiendo de las características que se seleccionen se pueden obtener: basadas en la naturaleza o no basadas en la naturaleza, basadas en memoria o sin memoria, con función objetivo estática o dinámica, etc. En este caso vamos a centrarnos en las metaheurísticas que están dadas por el número de soluciones que procesan en un determinado momento durante el proceso de búsqueda, es decir, aquellas “basadas en trayectoria” y las “basadas en población” (o poblacionales).

Las metaheurísticas basadas en trayectoria se caracterizan porque parten de un punto para mejora continua de la solución actual mediante la inspección de un vecindario donde la búsqueda finaliza cuando se alcanza un número máximo de iteraciones y se encuentra una solución con una calidad aceptable, o se detecta un estancamiento del proceso, mientras que

las metaheurísticas basadas en población trabajan con un conjunto de soluciones que iterativamente son mejoradas a través de un proceso inteligente de exploración del espacio de búsqueda.

En este trabajo nos hemos centrado en realizar una comparación entre una metaheurística poblacional denominada Algoritmo Genético Celular (Alba y Dorronsoro 2008) (*cellular Genetic Algorithms* - cGA) y una metaheurística de trayectoria denominada *Simulated Annealing* (Kirkpatrick et al. 1983) (SA) aplicadas a la resolución de problemas de optimización combinatoria para determinar cuál de ellas es la de mejor desempeño en la resolución de una batería de problemas de optimización combinatoria.

En las siguientes secciones se llevara a cabo una descripción más detallada de los conceptos a tener en cuenta en la investigación.

2. Conceptos generales de metaheurísticas

En las últimas décadas se ha desarrollado una nueva clase de algoritmos de aproximación que básicamente tratan de combinar métodos heurísticos básicos en un marco de alto nivel orientado a buscar la eficiencia en el proceso de búsqueda. Durante los años 80 y principios de los 90 estas técnicas se conocieron como heurísticas modernas (Glover et al. 1993). Dichas técnicas consisten en procedimientos sistemáticos de prueba que ofrecen soluciones aceptables, no necesariamente óptimos absolutos, para problemas donde el espacio de soluciones es indeterminado o lo suficientemente amplio como para que no pueda ser evaluado en un tiempo aceptable. En muchos casos, las técnicas heurísticas se diseñan en función de las características particulares del problema a resolver.

Además del concepto de heurística, es habitual hablar de metaheurística. Las metaheurísticas son procedimientos generales que permiten generar soluciones aproximadas a problemas. Aunque se suele considerar como metaheurística a toda aquella generalización de una determinada heurística para cualquier tipo de problema, en la práctica es necesario analizar detalladamente el problema a resolver para determinar cuál de ellas es la que se presupone, y puede tener un mayor éxito en la búsqueda de soluciones, si bien no se pueda determinar con total seguridad si alcanzaremos dicho éxito.

A continuación describimos las características básicas de las metaheurísticas, según la descripción ofrecida por Blum et al. (2003):

- son estrategias que guían el proceso de búsqueda;
- exploran eficientemente el espacio de búsqueda con el objetivo de encontrar soluciones próximas al óptimo global;
- las técnicas que constituyen las metaheurísticas varían entre métodos de búsqueda local simple a complejos métodos de aprendizaje;
- pueden incorporar mecanismos para evitar quedar atrapado en óptimos locales del espacio de búsqueda;
- permiten un nivel de descripción abstracto, no específico del problema;
- pueden hacer uso de conocimiento del dominio específico de forma que las heurísticas son controladas por una estrategia de nivel superior;
- las metaheurísticas avanzadas utilizan la experiencia de búsqueda (haciendo uso de alguna forma de memoria) para guiar la búsqueda.

En función de las características propias de cada metaheurística se pueden establecer diferentes clasificaciones. A continuación ofrecemos una de las clasificaciones más aceptadas en la actualidad (Blum y Roli):

- **Técnicas basadas en trayectorias vs. poblacionales**. Un aspecto diferenciador entre metaheurísticas, radica en el número de soluciones que se utilizan en el proceso de optimización. Metaheurísticas como *Simulated Annealing* (enfriamiento simulado) (Kirkpatrick et al. 1983), o Búsqueda Tabú (Glover et al. 1993) utilizan una sola solución durante el proceso de búsqueda, por lo que se suelen denominar métodos de trayectoria ya que la solución describe una trayectoria desde la solución de partida hasta encontrar la solución final. Por otro lado, técnicas como los algoritmos genéticos (Holland 1975) hacen uso de un conjunto de soluciones (población) que son optimizadas de forma simultánea durante la búsqueda. Normalmente los métodos poblacionales suelen conseguir soluciones de más calidad debido a la ventaja que supone explorar simultáneamente diferentes áreas del espacio de búsqueda. En la Figura 1. (a) muestra una posible trayectoria de una solución al aplicar SA y en (b) podemos ver la creación y mejora de una población de soluciones a las que se aplican operadores de mutación y cruce.

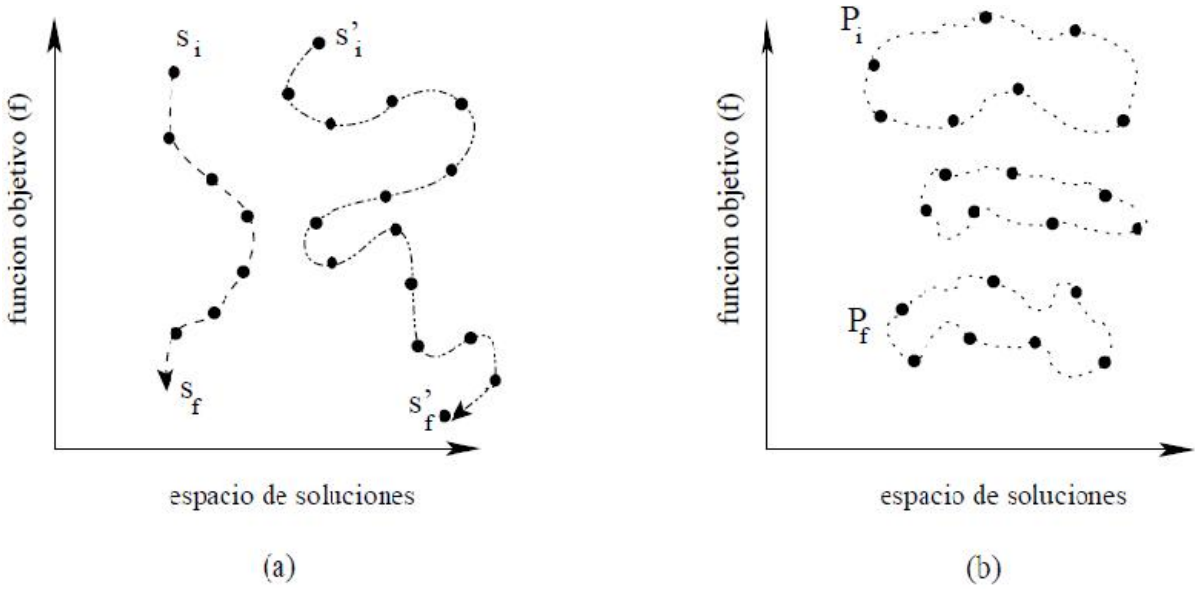

Figura 1:(a) Metaheuristicas basadas en trayectorias, (b) poblacionales

En este trabajo hemos abordado la metaheurística basada en trayectoria (SA) y la metaheurística poblacional (cGA), las cuales son unas de las metaheurísticas más clásicas. Su simplicidad y buenos resultados en numerosos problemas la han convertido en una de las herramientas más populares, con cientos de aplicaciones en los más variados campos, como lo es para resolver una amplia gama de problemas de optimización combinatoria.

3. Simulated Annealing

Esta metaheurística aparece a principios de los años 80 (Kirkpatrick et al. 1983). Se trata de un modelo de resolución para la optimización de problemas de tipo combinatorio con mínimos locales. Consiste en generar aleatoriamente una solución cercana a la solución actual (o en el entorno de la solución) y la acepta como buena si consigue reducir una determinada función de costo, o con una determinada probabilidad de aceptación. Esta probabilidad de aceptación se irá reduciendo con el número de iteraciones y está relacionada también con el

grado de empeoramiento del costo, es decir, en el algoritmo *Simulated Annealing* se pueden aceptar soluciones que empeoran la solución actual, solo que esta aceptación dependerá de una determinada probabilidad que depende de un parámetro, denominado temperatura, del estado del sistema.

El nombre de SA viene de la idea en que está basado en un algoritmo diseñado en los años 50 para simular el enfriamiento de material (un proceso denominado "recocido"). El proceso de SA es el siguiente: parte de una solución inicial que de modo paulatino es transformada en otras que a su vez son mejoradas al introducirles pequeñas perturbaciones o cambios (tales como cambiar el valor de una variable o intercambiar los valores que tienen dos variables). Si este cambio da lugar a una solución "mejor" que la actual, se sustituye ésta por la nueva, continuando el proceso hasta que no es posible ninguna nueva mejora. Esto significa que la búsqueda finaliza en un óptimo local, que no tiene por qué ser forzosamente el global.

Un modo de evitar este problema es permitir que algunos movimientos sean hacia soluciones peores. Pero por si la búsqueda está realmente yendo hacia una buena solución, estos movimientos "de escape" deben realizarse de un modo controlado. En SA esto se realiza controlando la frecuencia de los movimientos de escape mediante una función de probabilidad que hará disminuir la probabilidad de esos movimientos hacia soluciones peores conforme avanza la búsqueda (y por tanto estamos previsiblemente más cerca del óptimo global).

La fundamentación de este control se basa en el trabajo de Metropolis (1953) en el campo de la termodinámica estadística. Básicamente, Metropolis modeló el proceso de recocido mencionado anteriormente simulando los cambios energéticos en un sistema de partículas conforme decrece la temperatura, hasta que converge a un estado estable (congelado). La ley de la termodinámica dicen que a una temperatura t la probabilidad de un incremento energético de magnitud δE se puede aproximar por $P[\delta E] = exp\ (-\delta E\ /kt)$ siendo k una constante física denominada de Boltzmann. En el algoritmo de Metropolis se genera una perturbación aleatoria en el sistema y se calculan los cambios de energía resultantes: si hay una caída energética, el cambio se acepta automáticamente; por contra, si se produce un incremento energético, el cambio será aceptado con una probabilidad dada por la formula anterior. El proceso se repite durante un número predefinido de iteraciones en series decrecientes de temperaturas, hasta que el sistema esté "frío".

Los algoritmos de enfriamiento lento simulado tienen algunas ventajas con respecto a otras técnicas de optimización global. De acuerdo con (Elmohamed 1998) entre las ventajas de estos algoritmos se pueden citar:

- Lo relativamente sencillo que resulta implementar este tipo de problemas.
- Su aplicabilidad a la mayoría de los problemas de optimización con una estructura combinatoria.
- Su capacidad para ofrecer soluciones razonablemente buenas a la mayoría de los problemas, aunque hay una cierta dependencia de la planificación del enfriamiento y los movimientos que se realicen.

Por otra parte, también se pueden citar algunos aspectos que pueden limitar su utilización:

- Se necesita elegir con mucho cuidado los movimientos que se realizan, así como los parámetros que se van a utilizar para tratarlo, como por ejemplo la tasa de enfriamiento.
- Una ejecución del problema puede requerir mucho tiempo de cálculo.

- Puede que sea necesario realizar muchas ejecuciones para encontrar una solución satisfactoria.
- Dependiendo de los parámetros elegidos, las soluciones que se van encontrando pueden ser poco estables, "saltando" mucho de unas a otras sin encontrar una solución buena con la rapidez suficiente lo que obliga a retocar los parámetros con las distintas ejecuciones.

3.1 Estructura del algoritmo

En la Figura 2 se muestra la estructura básica de un algoritmo de enfriamiento lento simulado para el caso de problemas de maximización.

```
solución inicial ← S0
Seleccionar temp. inicial t0 > 0
Velocidad de enfriamiento ← v
Repetir
    Repetir
        Solución Vecina S ← S0
        si(S >= S0 )
            S0 ← S
        else{
            d ← f(s) - f( S0 )
            si( Math.random() < exp (-d/t) )
                S0 ← S
        }
    Hasta un número de repeticiones
    t ← t*v
Hasta condición de parada
Solución: la mejor de todas S0 encontradas
```

Figura 2: Pseudocódigo de un Simulated Annealing

En este seudocódigo se pueden diferenciar dos tipos de elementos que permiten afinar el algoritmo para que la ejecución del mismo sea la más eficaz de acuerdo con el problema que se desea resolver:

1. **Elementos Genéricos:** son elementos que no tienen una dependencia directa del problema, aunque hay que afinarlos para el problema concreto que se desea resolver.
    - Temperatura inicial: t0
    - Proceso de enfriamiento: v
    - Números de repeticiones.
    - Condición de parada.

2. **Elementos dependientes del problema.** Son los elementos que definen de forma directa el problema que se está resolviendo. Definen un modelo del problema y la estructura del espacio de soluciones para el mismo.
    - Espacio de soluciones: S
    - Solución inicial: s_0
    - Estructura de vecindad: s_i
    - Función de costo: f(s)

A continuación describiremos con más detalle los elementos mencionados.

a. Temperatura inicial

La temperatura inicial es la que se adopta para comenzar el proceso de enfriamiento y se irá bajando en tanto avance la evolución. Está debe ser lo suficientemente alta para permitir el libre cambio de soluciones vecinas y, sobre todo, que la solución final sea independiente de la solución inicial. Una regla práctica que se le suele poner a la temperatura inicial es que debe tener un valor tal que la proporción inicial de vecinos aceptados (tanto de mejora como de no mejora) tenga un alto valor, es decir, inicialmente todos los vecinos se aceptan con una probabilidad cercana a 1.

Lógicamente hay que elegir esta temperatura con cuidado porque si se elige una temperatura inicial demasiado alta se perderá tiempo realizando muchos movimientos que se aceptan siempre, lo que no resulta de mucho interés ya que no se está persiguiendo ningún valor concreto. Por otra lado, si se elige una temperatura inicial demasiado baja el proceso se mueve poco desde la solución inicial por lo que se puede quedar atrapado desde un inicio en una región desde la que sólo se pueda llegar a un óptimo local, por lo que se perderá la posibilidad de explorar otras regiones donde se podría encontrar alguno de los óptimos globales del sistema.

b. Proceso de enfriamiento

El proceso de enfriamiento es el mecanismo por el cual la temperatura va tendiendo a 0. Es decir, la probabilidad de aceptación de soluciones peores tiende a 0. El proceso de enfriamiento, por tanto, determina cómo se modifica la temperatura durante la ejecución del algoritmo.

Hay dos modelos básicos que se pueden utilizar para el proceso de enfriamiento. En el primero podríamos ir enfriando muy lentamente con cada iteración de la cadena de Markov de manera que en un número de pasos extremadamente alto la temperatura tienda a 0, ó utilizar un proceso de enfriamiento más rápido, pero antes de aplicar un cambio en la temperatura hay que dejar que el sistema llegue a un estado en el que se estabilice la solución.

Hay que intentar durante el proceso de enfriamiento que haya una alta aceptación al principio (de forma que se pueda realizar un proceso de exploración inicial en el espacio de búsqueda) e ir reduciendo está aceptación hacia el final.

c. Número de repeticiones

El número de repeticiones del bucle interno nos va a dar el número de vecinos que se visitan antes de reducir la temperatura del sistema. El proceso debe ser dinámico, de forma que se

aumente el número de repeticiones según se reduce la temperatura. De esta forma se conseguirá explotar la vecindad hacia el final del proceso.

Un mecanismo habitual para conseguirlo es repetir el bucle hasta que se cumpla una de las siguientes condiciones:

- Se aceptan un cierto número de evaluaciones. De esta forma el bucle termina en caso de que se hayan analizado un determinado número de evaluaciones del algoritmo.
- Se hayan obtenido la solución óptima. De esta forma el bucle termina si se da el caso que se ha conseguido la mejor solución.

Para que se pueda explorar con cierta completitud todos los vecinos.

d. Condición de parada

Esta condición nos indica cuando damos por terminado el algoritmo de cálculo. Hay que tener en cuenta que al reducirse la temperatura, la probabilidad de aceptar soluciones peores tiende a 0. Así mismo, la probabilidad de encontrar soluciones mejores también tiende a 0. En este sentido las condiciones para terminar el algoritmo deberían tener en cuenta que ya se han realizado un determinado número de iteraciones sin mejorar la solución. La idea es evitar seguir insistiendo reiteradamente en la vecindad de una solución para la que ya se lleva tiempo intentando explorar.

e. Espacio de soluciones

La definición del modelo del problema determinará la estructura del espacio de soluciones y, por tanto, la forma en que se va a poder recorrer, los algoritmos que se pueden utilizar para crear soluciones, etc.
Lógicamente para cada problema hay que estudiar de forma diferenciada cómo generar este espacio de soluciones, en relación con la facilidad para generar una vecindad y el cálculo de la función de costo asociada a una determinada solución.

f. Solución inicial

La solución inicial, en general es cualquier solución válida del problema. Al ser un problema de optimización combinatoria, el modelo permite expresar una solución como una combinación de partes del mismo. De esta forma, la solución inicial más sencilla resulta de elegir una combinación aleatoria de elementos del problema que conformen una solución con la estructura del modelo elegido.

g. Estructura de vecindad

Dada una solución, hay que poder conseguir una nueva solución mediante un pequeño cambio en la solución original. Además la forma de generar una vecindad nos debe poder asegurar que a partir de una solución cualquiera se debe poder llegar a cualquier otra ya sea directa o indirectamente. Esta característica es vital para dar validez al proceso de convergencia del algoritmo. Asimismo, la solución vecina debería ser suficientemente pequeña como para poder realizar una búsqueda en pocas iteraciones, como suficientemente grande para generar mejoras sustanciales en pocos movimientos.
Desde el punto de vista computacional la generación de un vecino debería ser rápida, de forma que se puedan calcular un mayor número de vecinos por unidad de tiempo. Así mismo,

sería muy apropiado que el vecino generado siempre sea una solución factible, lo que puede resultar complicado.

h. Función de costo

La función de costo es una medida asociada a todos los elementos de una determinada solución. Uno de los objetivos que deben guiar el desarrollo de la función de costo es que esta sea rápida de calcular al generar una solución vecina, creando un cálculo incremental que tenga en cuenta sólo los cambios que generan la vecindad. De esta forma se podrían calcular más soluciones por unidad de tiempo. De hecho, el cálculo de la función de costo suele ser uno de los aspectos más costosos en tiempo de cómputo del algoritmo.

4. Algoritmo Genético Celular

Dentro de los algoritmos evolutivos (*Evolutionary Algorithms* - EAs), que forman parte de metaheurísticas basadas en población, se encuentra una de las ramas que son de particular interés, las cuales son conocidas como algoritmos genéticos (*Genetic Algorithms* - GAs).

Los GAs, desarrollados principalmente por Holland durante los años 60, son algoritmos de búsqueda aleatoria basados en el modelo de la evolución biológica ([Goldberg 1989], [Holland 1975]). Según la definición dada, fácilmente podremos deducir que forman parte del campo de la computación evolutiva. Estos algoritmos tienen en cuenta el proceso de aprendizaje colectivo en una población de individuos, cada uno de los cuales representa un punto de búsqueda en el espacio de soluciones potenciales de un problema de optimización dado. La población evoluciona hacia regiones del espacio de búsqueda cada vez mejores por medio de procesos aleatorios como son la selección, la mutación y la recombinación. El mecanismo de selección favorece que, cuando se vaya a crear la nueva población, los individuos con mejor valor de adecuación (los más próximos a la solución) se reproduzcan con más frecuencia que los que cuentan con menores valores de fitness. La recombinación permite la mezcla de la información de los padres cuando se le pasa a sus descendientes, mientras que la mutación introduce un cierto factor de innovación en la información de los individuos de la población. Normalmente, la población inicial se inicializa con valores aleatorios, además, el proceso de evolución se detiene tras haber realizado un número predefinido de iteraciones.

Una de las herramientas más conocidas de los GAs estructurados, son los cGA(Alba y Dorronsoro 2008), que tienen una única instancia del GA opera sobre la población en forma de vecindario, es decir, para cada individuo, realiza las operaciones correspondientes entre él y sus vecinos.

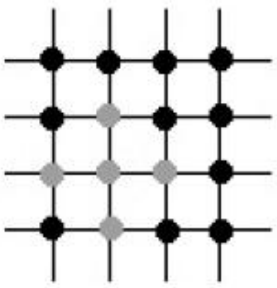

Figura 3 : Disposición de la población en un cGA

En la Figura 3 muestra la ventaja principal del cGA, donde el algoritmo celular no está centralizado, de manera que el ciclo reproductivo se realiza sobre cada individuo y sus vecinos como, a su vez, dicho individuo pertenece a varios vecindarios, esto permite que se produzca poco a poco una difusión de las mejores soluciones a través de la población. Por tanto, la población se estructura en una rejilla (normalmente en bidimensional, aunque el número de dimensiones puede ser extendido fácilmente a tres o más) con forma toroidal, y se les permite recombinarse con individuos cercanos.

El siguiente es un pseudocódigo de un cGA canónico (Figura 4). Se puede ver que tras la generación y evaluación (líneas 2 y 3 respectivamente) de la población inicial, los operadores genéticos tales como selección de los padres, recombinación, mutación y el reemplazo del individuo actual por un descendiente (de líneas 8 a 12), son aplicados a cada uno de los individuos dentro del entorno de sus vecindarios iterativamente hasta alcanzar una condición de finalización (en este caso MAX_PASOS).

Los individuos que formarán parte de la población de la siguiente generación (los nuevo descendientes generados o bien los individuos de la población actual, dependiendo del criterio de reemplazo declarado en la línea 12) van almacenándose en una población auxiliar que tras cada generación reemplaza a la población actual. Por tanto, en este modelo todos los individuos son actualizados simultáneamente en la población (Alba y Dorronsoro 2009).

```
1.  proc Evoluciona(cga)        // Parámetros del algoritmo en 'cga'
2.  GeneraPoblaciónInicial(cga.pobl);
3.  Evaluación(cga.pobl);
4.  para s ← 1 hasta MAX_PASOS hacer
5.     para x ← 1 hasta cga.ANCHO hacer
6.        para y ← 1 hasta cga.ALTO hacer
7.           vecinos ← CalculaVecindario(cga,posición(x,y));
8.           padres ← Selección(vecinos);
9.           descendiente ← Recombinación(cga.Pc,padres);
10.          descendiente ← Mutación(cga.Pm,descendiente);
11.          Evaluación(descendiente);
12.          Remplazo(posición(x,y),pobl_auxiliar,descendiente);
13.       fin para
14.    fin para
15.    cga.pobl ← pobl_auxiliar;
16. fin para
17. fin proc Evoluciona
```

Figura 4: Pseudocódigo de un cGA canónico.

En la siguiente sección se presenta una descripción de los problemas utilizados en este trabajo.

5. Problemas de Optimización Combinatoria

Cuando uno trata con la resolución de problemas rápidamente se da cuenta de que no todos ellos presentan el mismo grado de dificultad. Un problema de la *clase P*, es un problema que se puede resolver siempre utilizando un algoritmo que conlleva un número de pasos que es

función polimodal del tamaño de entrada del problema. Se han utilizado *métodos aproximados* mediante heurísticas que permiten aproximarse a una solución óptima, generando soluciones factibles al problema que resulten de utilidad práctica.

En las siguientes líneas se ofrece una descripción detallada de los distintos problemas de optimización combinatoria que hemos utilizado para el análisis:

- **Problema COUNTSAT** (Droste et al. 2000). Es una instancia de MAXSAT. En COUNTSAT el valor de la solución se corresponde con el número de cláusulas (de entre todas las posibles cláusulas de *Horn* de tres variables) satisfechas por una entrada formada por *n* variables booleanas. Es fácil comprobar que el óptimo se consigue cuando todas las variables tienen el valor 1. En este estudio consideramos una instancia de *n = 20* variables, y el valor de la solución óptima al problema es 6860.

  La función COUNTSAT esta extraída de MAXSAT con la intención de ser muy difícil de resolver con Gas (Droste et al. 2000). Las entradas aleatorias generadas uniformemente tendrán *n = 2* unos en término medio. Por tanto, los cambios locales decrementando el número de unos nos conducirán hacia entradas mejores, mientras que si se incrementa el número de unos se decrementará el valor de adecuación. En consecuencia, cabe esperar que un GA encuentre rápidamente la cadena formada por todos sus componentes a cero y tenga dificultades para encontrar la cadena compuesta por unos en todas sus posiciones.

- **Problema del Diseño de Códigos Correctores de Errores – ECC** (MacWilliams y Sloane 1977). Consideremos una tupla (*n; M; d*) donde *n* es la longitud (número de bits) de cada palabra de código, *M* es el número de palabras de código, y *d* es la mínima distancia de Hamming entre dos palabras de código cualesquiera. El objetivo será obtener un código con el mayor valor posible de *d* (reflejando una mayor tolerancia al ruido y a los errores), y con valores de *n y M* fijados previamente.

  En este estudio consideramos una instancia donde *C* está formado por *M*=24 palabras de longitud *n*=12 bits. Lo que forma un espacio de búsqueda de tamaño $\binom{4096}{24}$, que es 10^{87} aproximadamente. La solución óptima para la instancia con *M* =24 y *n* = 12 tendría el valor de adecuación 0.0674.

- **Problema de la Modulación en Frecuencia de Sonidos – FMS** (Tsutsui et al. 1997). Consiste en determinar los 6 parámetros reales $\vec{x} = (a_1; w_1; a_2; w_2; a_3; w_3)$ del modelo de sonido en frecuencia, donde = 2π/100. Este problema puede ser definido como un problema de optimización discreta o continua. En el caso que discreto, los parámetros están codificados en el rango [-6.4, +6.35] con cadenas de 32 bits.

  El objetivo de este problema consiste en minimizar la suma del error cuadrático medio entre los datos de muestre y los reales. Este problema es una función multimodal altamente compleja. Debido a la extrema dificultad para resolver este problema con alta precisión sin utilizar operadores específicos de optimización continua, el algoritmo se detiene cuando el error cae por debajo de 10^{-2}. La función de adecuación que tendremos que maximizar se corresponde su máximo valor cuando $E_{FMS} = 0.0$.

- **Problema del Máximo Corte de un Grafo – MAXCUT**. Consiste en dividir un grafo ponderado $G = (V, E)$ en dos subgrafos disjuntos $G_0 = (V_0, E_0)$ y $G1 = (V_1, E1)$ de manera que la suma de los pesos de los ejes que posean un extremo en V_0 y el otro en V_1 sea máxima. Para codificar una partición de los vértices utilizaremos una cadena binaria $(x_1, x_2,\ldots, x_n)$ donde cada dígito se corresponde con un vértice. De manera que si un dígito tiene valor 1, su vértice correspondiente estaría en el conjunto V_1, y si tuviera valor 0 el vértice respectivo pertenecería a V_0. Aunque es posible generar instancias de grafos diferentes de forma aleatoria, hemos utilizado tres instancias distintas del problema. Dos de ellas se corresponden con grafos generados aleatoriamente: uno disperso "MAXCUT20_01" y otro denso "MAXCUT20_09", ambos formados por 20 vértices. La otra instancia es un grafo escalable de 100 vértices. Las soluciones óptimas para estas instancias son 10,11 para el caso de "MAXCUT20_01", 56,74 para "MAXCUT20_09" y 1077 para "MAXCUT100".

- **Problema Masivamente Multimodal – MMDP.** Esta específicamente diseñado para ser difícil de afrontar para los EAs. Está compuesto de k sub-problemas engañosos (s_i) de 6 bits cada uno. El valor de cada uno de estos sub-problemas (*fitness* s_i) depende del número de unos que tiene la cadena binaria que lo compone. Es fácil comprender por qué se considera a esta función engañosa, puesto que estas sub-funciones tienen dos máximos globales y un atractor engañoso en el punto medio.

  En MMDP, cada sub-problema s_i contribuye al valor de fitness total en función del número de unos que tiene. El óptimo global del problema tiene el valor k, y se alcanza cuando todos los sub-problemas están compuestos de cero o seis unos. El número de óptimos locales es muy grande (22^k), mientras que sólo hay 2^k soluciones globales. Por tanto, el grado de multimodalidad viene dado por el parámetro k. En nuestros estudios, utilizaremos (mientras no se especifique lo contrario) una instancia considerablemente grande de $k = 40$ sub-problemas. La función que trataremos de maximizar corresponde con un valor máximo de 40.

- **Problema de las Tareas de Mínima Espera – MTTP (**Centre y Stinson 1985). Es un problema de planificación de tareas en el que cada tarea i del conjunto de tareas $T = \{1, 2,\ldots, n\}$ tiene una longitud l_i - el tiempo que tarda en ejecutarse- un límite de tiempo (o deadline) di -antes del cual debe ser planificada- y un peso w_i. El peso es una penalización que debe ser añadida a la función objetivo en el caso de que la tarea permanezca sin estar planificada. Las longitudes, pesos, y límites de tiempo de las tareas son todos números enteros positivos. Planificar la tarea de un sub-conjunto S de T consiste en encontrar el tiempo de comienzo de cada tarea en S, de forma que como mucho se realiza una tarea cada vez y que cada tarea termina de realizarse antes de su límite de tiempo.

  El objetivo del problema es minimizar la suma de los pesos de las tareas no planificadas. Por tanto, la planificación óptima minimiza la función.

  Hemos utilizado para nuestros estudios tres diferentes instancias (Khuri et al. 1994): "MTTP", "MTTP100" y "MTTP200", de tamaños 20, 100 y 200, y valores óptimos de 0.02439, 0.005 y 0.0025, respectivamente.

- **Problema OneMax** (Schaffer y Eshelman 1991). Es un problema muy simple que consiste en maximizar el número de unos que contiene una cadena de bits. Formalmente, este problema se puede describir como encontrar una cadena $\vec{x} = \{x1, x2,\ldots, x_n\}$ con $x_i \in \{0, 1\}$, que maximice la función. La función f_{OneMax} tiene $(n+1)$ posibles valores diferentes, que están distribuidos multimodalmente. Para definir una instancia de este problema, sólo necesitamos definir la longitud de la cadena de *bits (n)*. Para un *n* dado, la solución óptima al problema es una cadena de *n* unos, es decir, se les fija el valor uno a todos los *bits* de la cadena.

- **Problema P-PEAK** (Kenneth et al. 1997). Es un generador de problemas multimodales. Un generador de problemas es una tarea fácilmente parametrizable, con un grado de dificultad que se puede afinar, de manera que nos permita generar instancias con una complejidad tan grande como queramos. Adicionalmente, el uso de un generador de problemas elimina la posibilidad de afinar a mano los algoritmos para un problema particular, permitiendo de esta forma una mayor justicia al comparar los algoritmos. Utilizando un generador de problemas, evaluamos nuestros algoritmos sobre un elevado número de instancias de problema aleatorias, ya que cada vez que se ejecuta el algoritmo resolvemos una instancia distinta. Por tanto, se incrementa el poder de predicción de los resultados para la clase de problemas como tal, no para instancias concretas.

La idea de *P-PEAK* consiste en generar *P* cadenas aleatorias de *N bits* que representan la localización de *P* cimas en el espacio de búsqueda. Usando un mayor (o menor) número de cimas obtenemos problemas más (o menos) epistáticos. En nuestro caso hemos utilizado una instancia de $P = 100$ cimas de longitud $N = 100$ *bits* cada una, representando un nivel de epistasis medio/alto (Alba y Troya 2000). El máximo valor de *fitness* que podremos conseguir es 1.0.

Luego de haber descripto las instancias de los problemas a utilizar en este trabajo, en la siguiente sección presentamos el diseño de procedimientos y los resultados obtenidos.

6. Diseño de experimentos y resultados

Con el objetivo de estudiar el comportamiento de la metaheurística de trayectoria SA y metaheurística poblacional cGA para resolver un conjunto de problemas de optimización combinatoria se realizaron 30 corridas independientes. Los algoritmos se corrieron en una PC con sistema operativo Windows 7 Starter, procesador de 2,10 GHz, memoria RAM de 2 GB y el software utilizado para realizar dicha acción fue Eclipse.

La parametrización de cGA se realizo teniendo en cuenta lo propuesto en (Dorronsoro 2007). Para SA se realizaron una serie de pruebas para encontrar los parámetros adecuados. A continuación en la Tabla 1 se resume los valores utilizados tanto para SA como para cGA:

Parámetro	cGA	SA
Tamaño de población	400 individuos (20x20)	1 individuo
N° de evaluaciones	8000000	8000000
Longitud de individuo	L (longitud de individuo)	L (longitud de individuo)
Temperatura inicial	-	1000
Velocidad de Enfriamiento	-	0,003

Tabla 1: Parametrización utilizada en cGA y SA

Para el cGA se utilizó un tamaño de población de 400 individuos y en el SA solo se genero un individuo de acuerdo al análisis del problema. Cada individuo se genera de manera aleatoria con valores binarios. Por otro lado el número máximo de evaluaciones utilizadas en los dos algoritmos (SA y cGA) fue de 8000000. La longitud del individuo es una variable que se establece automáticamente de acuerdo al problema a ejecutar. En SA el parámetro de la temperatura inicial se ha establecido con el valor de 1000 buscando que a altas temperaturas en el enfriamiento simulado se diversifique aceptando las soluciones malas con altas probabilidades; y que a bajas temperaturas explorara esas soluciones encontradas a altas temperaturas, aceptando con una baja probabilidad pocas soluciones malas.

A continuación analizamos los resultados obtenidos por los algoritmos aplicados al conjunto de problemas de optimización, con el fin de determinar cuál es el de mejor comportamiento y determinar si las diferencias son estadísticamente significativas. En este trabajo los mejores resultados obtenidos están resaltados con **negrita**. En la Tabla 2 se muestran los porcentajes de éxito obtenidos de 30 corridas independientes en ambos algoritmos para cada una de las instancias del problema abordado. Donde podremos observar que tanto en cGA como SA obtienen el 100% de éxito en 8 de las 12 instancias. No obstante cGA obtiene en tres instancias un mayor porcentaje de éxito que SA (instancias FMS, MAXCUT100 y MMDP). Al final de la tabla hemos calculado el promedio de éxito general para cada uno de los algoritmos en la cual podemos deducir que cGA cuenta con el mayor promedio de éxito (82%).

INSTANCIA	Óptimo	cGA	SA
COUNTSAT	6860,00	0%	**100%**
ECC	0,06	**100%**	**100%**
FMS	0,01	47%	0%
MAXCUT100	1077,00	67%	**100%**
MAXCUT20_01	10,11	**100%**	**100%**
MAXCUT20_09	56,74	**100%**	**100%**
MMDP	40,00	70%	0%
MTTP	0,02	**100%**	**100%**
MTTP100	0,00	**100%**	0%
MTTP200	0,00	**100%**	0%
OneMax	500,00	**100%**	**100%**
P-PEAK	1,00	**100%**	**100%**
Promedio	-	**82%**	67%

Tabla 2: Porcentaje de éxito en los resultados obtenidos para cGA y SA

En la Tabla 3 se resume el valor estadístico más representativo para cada una de las instancias en los algoritmos mencionados anteriormente. La primera columna pertenece a la instancia utilizada (problema). En las siguientes columnas se presenta el valor de la mediana en relación a la cantidad de evaluaciones y el tiempo (en milisegundos) que demanda. Los mejores valores para la mediana se colocan en negrita.

INSTANCIA	cGA		SA	
	Evaluación	Tiempo	Evaluación	Tiempo
ECC	154400	10734	**36923**	**3266**
MAXCUT20_01	4400	125	**700**	**109**
MAXCUT20_09	7000	**187**	**2237**	188
MTTP	5000	140	**914**	**133**
OneMax	**129400**	14141	393660	**12078**
P-PEAK	46400	10290	**1560**	**390**

Tabla 3: Resultados obtenidos por los algoritmos para las instancias abordadas en este estudio.

De acuerdo a estos valores podemos notar que en SA se consiguieron en la mayoría de los casos analizados el menor valor de la mediana con respecto a la evaluación y tiempo. A excepción del problema MAXCUT20_09 (tiene el mejor tiempo de la mediana) y OneMax (tiene la mejor evaluación de la mediana) en cGA, como muestran en Tabla 3.

En cuanto a los test estadísticos podemos decir que existen dos pruebas estadísticas que nos permite determinar si existen diferencias significativas entre los algoritmos: las *paramétricas* y las *no paramétricas*.

Para elegir un ***test estadístico paramétrico*** es necesario que la variable sobre la cual se va a ejecutar el procedimiento estadístico deba cumplir con las siguientes restricciones relacionas con la existencia de Independencia, Normalidad y Homocedasticidad. En el caso contrario el ***test no paramétrico*** se aplica cuando no cumplen con *algunas* de las condiciones mencionadas anteriormente. Como estamos analizando dos algoritmos aplicaremos el T-test como test paramétrico y el test de *Wilcoxon Signed-Rank* (*o U de Mann-Whitney*) como test no paramétrico.

En este trabajo la independencia de resultados está demostrada por tratarse de ejecuciones independientes de los algoritmos. Por otro lado, mediante los test de Kolmogarov-Smirnov nos permitirá examinar la Normalidad y para realizar un análisis de Homocedasticidad utilizaremos el test de Levene. En este estudio siempre trabajamos para todos los test con un nivel de significancia $\alpha=0,05$ y un p-valor asociado que representa la disimilitud de los resultados con respecto a la forma normal, de acuerdo a esto podemos decir que, si un p-valor es menor del nivel de significancia entonces señala una distribución no normal, en caso contrario que el p-valor sea mayor a α indica que la condición de normalidad se cumple. Para realizar estos test utilizaremos la herramienta SPSS 15.0 para Windows.

En la Tabla 4 podremos observar la prueba estadística del test de Kolmogarov Smirnov aplicados en los algoritmos de cGA y SA, en la cual el símbolo "*" significa que los resultados obtenidos por los algoritmos no cumplen con la condición de normalidad, en base a esto cabe mencionar que cuando los resultados de al menos un algoritmo no cumplen las condiciones de normalidad debemos entonces aplicar test no paramétricos.

	cGA		SA	
Instancia	**Evaluación**	**Tiempo**	**Evaluación**	**Tiempo**
ECC	0,89	0,90	0,10	0,10
MAXCUT100	(*)0,01	(*)0,00	0,17	0,09
MAXCUT20_01	0,92	0,84	0,16	0,24
MAXCUT20_09	0,77	0,36	0,19	0,73
MTTP	0,63	0,38	0,15	0,47
OneMax	0,53	0,55	0,56	0,51
P-PEAK	0,72	0,88	(*)0,01	(*)0,00

Tabla 4: Test de Kolmogorov-Smirnov aplicado a variables de performance 'Evaluaciones' y 'Tiempo' de un cGA y SA.

Los valores representados en la Tabla 4 ponen de manifiesto que no se cumple la condición de normalidad dado que para la instancia MAXCUT100 (en cGA) y P-PEAK (en SA) presentan valores menores al nivel de significancia.

Ahora si analizamos la Tabla 5 podremos notar que no se cumple la Homocedasticidad al realizar el test de Levene. Hemos utilizado el símbolo "*" para indicar que el resultado obtenido para el conjunto de datos no cumple con dicha condición.

Instancia	Estadistico de Levene	
	Evaluación	Tiempo
ECC	0,07	(*)0,03
MAXCUT100	(*)0,00	0,11
MAXCUT20_01	(*)0,00	0,21
MAXCUT20_09	0,10	(*)0,00
MTTP	0,89	(*)0,00
OneMax	(*)0,00	(*)0,00
P-PEAK	(*)0,00	(*)0,00

Tabla 5: Test de Levene aplicado a variables de 'Evaluación' y 'Tiempo' de los algoritmos de cGA y SA.

Analizando los valores obtenidos en las tablas de pruebas estadísticas de Levene y Kolmogorov-Smirnov para los resultados de ambos algoritmos en la variable de performance evaluación de las instancias ECC, MAXCUT20_09 y MTTP hemos observado que tienen una distribución normal y poseen homocedasticidad, al igual como sucede en la variable de performance tiempo para la instancia MAXCUT20_01 por tanto podemos aplicar test paramétricos en este caso T-test. Sin embargo para los restantes resultados al no cumplirse algunas de las condiciones se deben aplicar test no paramétricos (Wilcoxon) para las variables en estudio y determinar la existencia de diferencias estadísticas significativas entre los algoritmos. En la representación de la Tabla 6 se utiliza el signo (+) para señalar la existencia de diferencias estadísticamente significativas entre los resultados obtenidos por los algoritmos y (-) en caso opuesto, según el test que corresponda.

INSTANCIA	Evaluación		T-test / Wilcoxon	Tiempo		T-test / Wilcoxon
	cGA	SA		cGA	SA	
ECC	154400	**36923**	**(+)**	10734	**3266**	**(+)**
MAXCUT20_01	4400	**700**	**(+)**	125	**109**	**(+)**
MAXCUT20_09	7000	**2237**	**(+)**	**187**	188	**(-)**
MTTP	5000	**914**	**(+)**	140	**133**	**(+)**
OneMax	**129400**	393660	**(+)**	14141	**12078**	**(+)**
P-PEAK	46400	**1560**	**(+)**	10290	**390**	**(+)**

Tabla 6: T-test y Wilcoxon aplicado a las variables de performance 'Evaluación' y 'Tiempo' en los algoritmos de cGA y SA.

En la Tabla 6 podemos observar que SA para las instancias más pequeñas (ECC, MAXCUT20_01, MTTP y P-PEAK) tienen diferencias estadísticamente significativas con respecto a cGA en cuanto a las variables de performance evaluación y tiempo. Además SA presenta diferencias estadísticamente significativas en evaluaciones para MAXCUT20_09 en relación a cGA y diferencias estadísticamente significativas en cuanto a tiempo con respecto a cGA para OneMax. En cuanto a cGA presenta diferencias estadísticamente significativas (en evaluaciones) para OneMax con respecto a SA.

Seguidamente para mostrar gráficamente estas diferencias se han seleccionado algunos casos que manifiestan este comportamiento en gráfica de Box-Plot, la cual muestran como se distribuyen los resultados con respecto a la mediana. En las siguientes figuras se muestran únicamente dos resultados representativos del conjunto analizado. La Figura 5, 6, 7 y 8

muestran en boxplot los resultados obtenidos de la cantidad de evaluaciones y el tiempo en ejecución correspondiente a las instancias especificadas en las figuras para los diferentes algoritmos estudiados (cGA y SA). Podemos observar en la Figura 5 que el algoritmo cGA es más robusto que SA para la instancia OneMax. Los valores obtenidos están todos alrededor de la mediana y son menores a los obtenidos por SA. En la Figura 6 para la variable Tiempo, SA obtiene menores valores. No obstante cGA tiene resultados compactos. En las Figuras 7 y 8 vemos una marcada diferencia entre SA y cGA para el problema P-PEAK en ambas variables. Recordemos además que las diferencias son estadísticamente significativas y lo refleja claramente la gráfica.

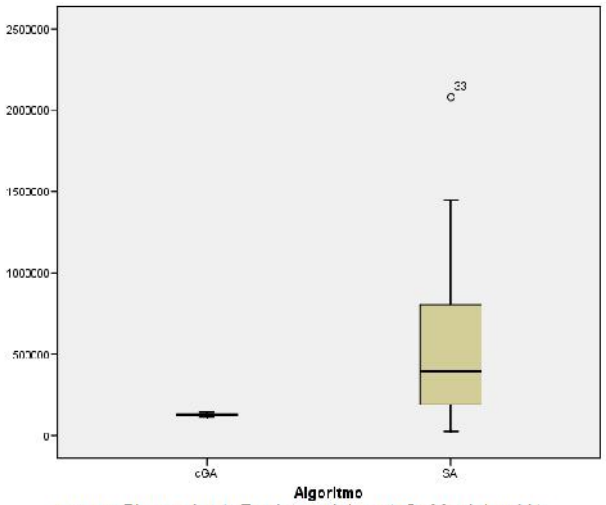

Figura 5: Diagrama de caja (Boxplot) para la instancia OneMax de la variable Evaluación en cGA y SA.

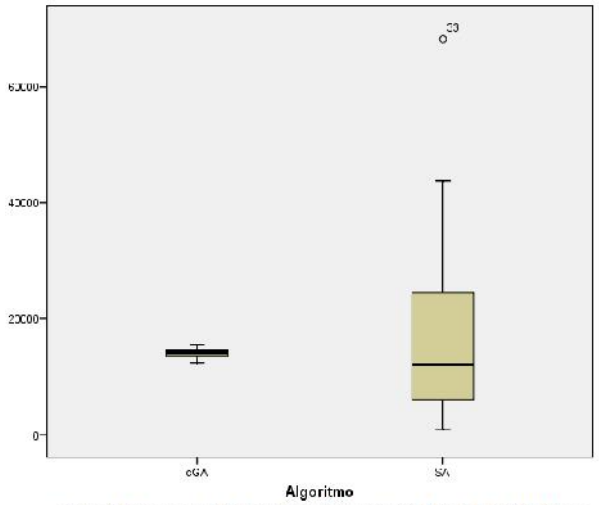

Figura 6: Diagrama de caja (boxplot) para la instancia OneMax de la variable Tiempo en los algoritmos cGA y SA.

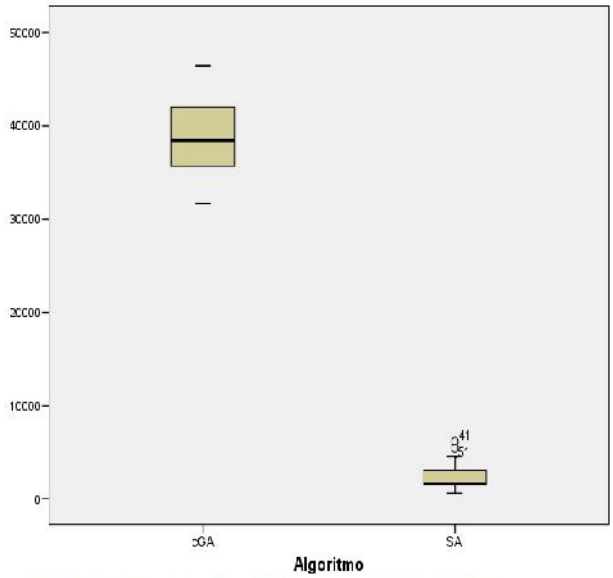

Figura 7: Diagrama de caja (Boxplot) para la instancia P-PEAK de la variable Evaluación en cGA y SA.

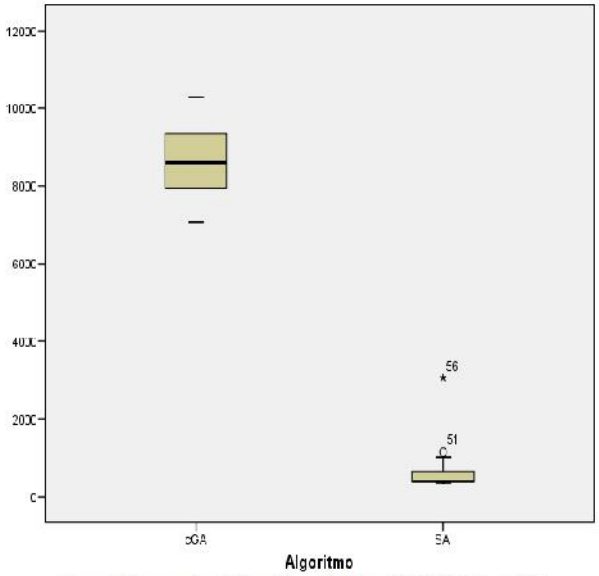

Figura 8: Diagrama de caja (Boxplot) para la instancia P- PEAK de la variable Tiempo en cGA y SA.

7. Conclusiones

En este trabajo de investigación hemos llevado a cabo el estudio de la metaheurística *Simulated Annealing* aplicado a diversos problemas de optimización combinatoria, comparando los resultados obtenidos contra los de una metaheurística poblacional Algoritmo Genético Celular. Se realizo un estudio estadístico detallado sobre las variables de número de evaluaciones y tiempo. Aplicamos test paramétricos y no paramétricos según el caso, para determinar la existencia de diferencias estadísticamente significativas entre los resultados obtenidos por los algoritmos en cada uno de los problemas tratados.

Finalmente para observar estas diferencias se han seleccionado algunos que se muestran, en grafica de Boxplot. Luego de realizar los análisis podemos concluir con un 95% de confianza que el algoritmo *Simulated Anneling* tiene un mejor desempeño que el Algoritmo Genético Celular cuando se lo aplica a las instancias pequeñas en este caso ECC, MAXCUT20_01, MTTP y P-PEAK. En todos ellas la diferencia fue estadísticamente significativa en cuanto a evaluaciones y tiempo. No obstante, el poblacional Algoritmo Genético Celular ha tenido un mejor desempeño en cuanto al porcentaje de éxito para instancias complejas como FMS, MMDP, MTTP100 y MTTP200 donde SA no logró encontrar el óptimo. Además en cuanto al promedio de éxitos el Algoritmo Genético Celular logró un 82% contra un 67% obtenido por el Simulated Annealing. Hemos podido comprobar con el conjunto de instancias seleccionadas el buen desempeño de la metaheurística de trayectoria *Simulated Annealing* en problemas de tamaño pequeño y el buen desempeño de la metaheurística poblacional cGA en instancias complejas de gran tamaño.

Como trabajo futuro se realizará el estudio de otros problemas de optimización así como también la incorporación de otras metaheurísticas tanto de trayectoria como poblacional.

8. Agradecimientos

Se agradece la cooperación del equipo de proyecto del LabTEm y la Universidad Nacional de la Patagonia Austral, por el continuo apoyo y los valiosos aportes para el desarrollo de este trabajo.

9. Referencias

Alba E. y Dorronsoro B, 2008. *Cellular Genetic Algorithms*. Springer.

Alba E. y Dorronsoro B, 2009. Cellular Genetic Algorithms, volume 42. Springer.

Alba E. y Troya J. M, 2000. Cellular evolutionary algorithms: Evaluating the influence of ratio. In Parallel Problem Solving from Nature PPSN VI, pages 29–38. Springer.

Blum C. y Roli, A., 2003. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison, ACM Computing Surveys 35(3), pp. 268-308.

Centre Ch. B. R. y Stinson D. R, 1985. An introduction to the Design and Analysis of Algorithms. Charles Babbage Research Centre.

Crainic T. y Toulouse M, 2003. Handbook of Metaheuristics, chapter Pa-rallel Strategies, pages 475–513. Kluwer Academic Publishers.

Dorronsoro B, 2007. Diseño e implementación de algoritmos genéticos celulares para problemas complejos. PhD thesis, Universidad de Málaga.

Droste S., Jansen T. y Wegener I, 2000. A natural and simple function which is hard for all evolutionary algorithms, volume 4. IEEE.

Elmohamed S., Fox G. y Coddington P, 1998. ”A comparision of Annealing Techniques for Academic Course Scheduling,” in Practice an theory of Automated Timetabling (PATAT) II, vol. 1408, Lecture Notes in Computer Science, E. Burke and M. Carter, Eds. Berlin: Springer-Verlag.

Glover F. y Kochenberger G, 2003. Handbook of Metaheuristics , Kluwer Academic Publishers.

Glover F., Laguna M. y Dowsland K. A, 1993. Modern Heuristic Techniques for Combinatorial Problems. C.R. Reeves (ed.), Blackwell, London, 1993.

Goldberg D. E, 1989. Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, New York.

Holland J, 1975. Adaptation in Natural and Arti_cial Systems, MIT Press.

Kenneth A De J., Mitchell A P. y Spears W. M, 1997. Using problem generators to explore the effects of epistasis. In ICGA, pages 338–345. Citeseer.

Khuri S., Bäck T. y Heitkötter J, 1994. An evolutionary approach to combinatorial optimization problems. In ACM Conference on Computer Science, pages 66–73.Citeseer.

Kirkpatrick S., Gelatt C.D. y Vecchi M.P, 1983. Optimization by Simulated Annealing, Science 220, pp. 671.

MacWilliams F. J. y Sloane NJ N. J. A, 1977. The Theory of Error-correcting Codes: Part 2, volume 16. Elsevier.

Schaffer D. J. and Eshelman L. J, 1991. On crossover as an evolutionarily viable strategy. In ICGA, volume 91, pages 61–68.

Tsutsui S., Ghosh A., Corne D. y Fujimoto Y, 1997. A real coded genetic algorithm with an explorer and an exploiter populations. In ICGA, pages 238–245, 1997.