ICT-UNPA-161-2017
ISSN: 1852-4516
Aprobado por Resolución N° 0334/17-R-UNPA

Una propuesta para diversificar el campo de aplicación de un algoritmo Hill Climbing

A proposal to diversify the application field of an algorithm Hill Climbing

Christian Soria <lchristiansoria@gmail.com>
Hernán da Silva <hernanunpa@gmail.com>
Adriana Martín <adrianaelba.martin@gmail.com>

Unidad Académica Caleta Olivia (UACO), Universidad Nacional de la Patagonia Austral (UNPA), Ruta 3, Acceso Norte, Caleta Olivia, Santa Cruz, Argentina
Noviembre 2016

RESUMEN

Cuando se habla de optimización, se busca obtener la mejor solución posible a un problema determinado, de la mejor manera, utilizando todos los recursos disponibles. Los métodos para encontrar la mejor solución varían de acuerdo a la complejidad del problema enfrentado. A medida que la complejidad de un problema aumenta, crece la necesidad de implementar métodos heurísticos para resolverlo. Por ejemplo, las metaheurísticas basadas en trayectoria (denominadas como *Hill Climbing*), se aplican a la resolución de problemas de optimización combinatoria, donde el conjunto de posibles soluciones es discreto, o al menos, se puede reducir a un conjunto discreto. Nosotros proponemos modificar el algoritmo *Hill Climbing* a los efectos de incrementar la diversidad de problemas a los cuales pueda aplicarse este algoritmo. En primer lugar, se analizan y comparan de forma incremental 6 (seis) propuestas de mejora: 5 (cinco) basadas en el algoritmo *Hill Climbing* y, 1 (una) basada en el algoritmo genético. Luego, desarrollamos nuestra propuesta como resultado del análisis de estas 6 (seis) propuestas de mejora y considerando algunas áreas en las que no se encontraron registros anteriores. En este trabajo, se presenta y describe una propuesta a los efectos de contribuir con nuevas ideas al campo de la optimización combinatoria. Nuestro algoritmo está basado en la hibridación de metaheurísticas poblacionales y en trayectoria. Esta propuesta es el resultado de la investigación desarrollada durante 2016 en la asignatura "Técnicas para la Elaboración de Documentos Científicos-Técnicos", perteneciente a la carrera de grado "Ingeniería en Sistemas UNPA."

Palabras clave: Metaheurísticas; Hill Climbing; Optimización combinatoria.

ABSTRACT

When talking about optimization, we are seeking to obtain the best possible solution to a given problem, in the best way, using all available resources. The methods to find the best solution differ according to the complexity of the problem we are facing. As the complexity of a problem increases, the need for implementing heuristic methods to solve the problem grows. For example, path-based metaheuristics (called as *Hill Climbing*), are applied for solving combinatorial optimization problems, where the set of possible solutions is discrete, or at least, can be reduced to a discrete set. We propose to modify *Hill Climbing* algorithm in order to increase the diversity of problems to which this algorithm can be applied. In first place, we analyzed and compared incrementally 6 (six) improvement proposals: 5 (five) based

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

on the *Hill Climbing* algorithm and, 1 (one) based on the genetic algorithm. Then, we develop our proposal as a result of the analysis of these 6 (six) improvement proposals and taking into account some areas in which no previous records were found. In this work, we present and describe a proposal in order to contribute with new ideas to the field of combinatorial optimization. Our algorithm is based on the hybridization of population metaheuristics and on trajectory. This proposal is the result of the research developed during 2016 within the "Técnicas para la Elaboración de Documentos Científicos-Técnicos" course, belonging to the "Ingeniería en Sistemas UNPA" undergraduate career.

Key words: Metaheuristics; Hill Climbing; Combinatorial optimización.

1. INTRODUCCIÓN

La optimización en un contexto científico, es el proceso por el cual, se trata de encontrar la mejor solución posible para un problema determinado. Cuando hablamos de problemas de optimización, estamos haciendo referencia a problemáticas reales. Estas problemáticas, generalmente de difícil solución, se encuentran en distintas áreas de aplicación, presentando gran complejidad computacional para ser resueltas. Las mismas, se denominan no determinìstas en tiempo polinomial difíciles (NP-difíciles) (Garey y Johnson 1979). Un problema de optimización de este tipo es aquel para el que no podemos garantizar encontrar la mejor solución posible en un tiempo razonable. La resolución de los mismos se realiza a través de algoritmos de aproximación, los cuáles, proporcionan soluciones consideradas buenas para un determinado problema, en un tiempo moderado. Esto no necesariamente significa que sean soluciones óptimas. Estos métodos, en los que la rapidez del proceso es tan importante como la calidad de la solución obtenida, se denominan heurísticos o aproximados.
Con el propósito de obtener mejores resultados que los alcanzados por los heurísticos tradicionales surgen los denominados procedimientos metaheurísticos (Goldberg 1989). Estos procedimientos son una clase de métodos aproximados que están diseñados para resolver problemas de optimización complejos, en los que los heurísticos clásicos no son efectivos. Si bien existen distintos tipos de clasificaciones para métodos de solución de problemas, en (Morillo et al. 2014) los autores proponen dividirlos en dos ramas: Métodos Iterativos y métodos de Aproximación.

Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos, combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos. (Osman 1996, Martí 2003). La aplicación de técnicas metaheurísticas es especialmente interesante en caso de problemas de optimización combinatoria, que son problemas en las que las variables de decisión son enteras (o discretas, al menos) y que generalmente, tienen un espacio de soluciones formado por ordenaciones de valores de dichas variables. Sin embargo, las técnicas metaheurísticas se pueden aplicar también a problemas de otro tipo, como con variables continuas, por ejemplo.

Las técnicas metaheurísticas más extendidas son las siguientes: los algoritmos genéticos (GA o *genetic algorithm*), la búsqueda tabú *(TS o tabu search*), el recocido simulado (SA o *simulated annealing*), la búsqueda dispersa (SS o *scatter search*), los algoritmos voraces (GRASP o *greedy randomized adaptative search procedure*), y los algoritmos de colonia de hormigas (ACA o *ant colony algorithms)*, entre otras.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

En este trabajo se analizan un conjunto de algoritmos, entre ellos Hill Climbing y Genético, en los cuales se basa el algoritmo que se propone a continuación. El mismo tiene como objetivo incrementar el campo de problemas al que pueda aplicarse eficientemente.

El documento se organiza de la siguiente manera: En la Sección 2, se presentan antecedentes y trabajos relacionados, donde se desarrollarán conceptos teóricos sobre metaheurísticas en general y en particular se explicará el Algoritmo *Hill Climbing*. En la Sección 3, se describe nuestra propuesta de modificación al Algoritmo *Hill Climbing* propiamente dicho. Mientras que, en la Sección 4, se presenta el pseudocódigo del algoritmo propuesto, el cual se aplica a un caso de estudio para discutir sus ventajas y desventajas. Finalmente, en la Sección 5, se presentan las conclusiones y trabajo futuro.

2. ANTECEDENTES Y TRABAJOS RELACIONADOS

En esta Sección, se realiza un recorrido por los antecedentes y trabajos relacionados, a los efectos de proveer un marco conceptual adecuado para nuestra propuesta.

2.1 METAHEURÍSTICAS

Durante la década de los setenta se implementaron diversas técnicas con el objetivo de encontrar soluciones a problemas de optimización combinatoria. Las mismas se conocen como metodologías heurísticas. El término metaheurística fue introducido por primera vez por Glover (1986). El término metaheurística surge de anteponer el sufijo "meta" al término heurística, justamente para explicitar que estos algoritmos van "más allá" o trabajan "a un nivel superior" en la búsqueda de soluciones de buena calidad. Antes de que el término fuese aceptado por la comunidad científica estas técnicas eran conocidas como heurísticas modernas (Reeves 1993). Básicamente son algoritmos que permiten explorar de forma eficiente el vasto espacio de búsqueda de una solución. Gran parte de estos métodos parten de problemas de fácil representación, pero de muy difícil solución.
Si bien existen diferentes técnicas metaheurísticas todas tienen la siguiente serie de características en común (Sait y Youssef 1999):

- Son ciegas, no saben si llegan a la solución óptima. Por lo tanto, se les debe indicar cuándo deben detenerse.
- Son algoritmos aproximativos y, por lo tanto, no garantizan la obtención de la solución óptima.
- Aceptan ocasionalmente malos movimientos (es decir, se trata de procesos de búsqueda en los que cada nueva solución, no es necesariamente mejor que la anterior). Algunas veces aceptan, incluso, soluciones no factibles como paso intermedio para acceder a nuevas regiones dentro del espacio de búsqueda no explorado.
- Son relativamente sencillas; todo lo que se necesita es una representación adecuada del espacio de soluciones, es decir, una solución inicial (o un conjunto de ellas) y un mecanismo para explorar el campo de soluciones.
- Son generales. Prácticamente se pueden aplicar en la resolución de cualquier problema de optimización de carácter combinatorio. Vale destacar que, a mayor relación entre el problema considerado y la definición de la técnica, más eficientes y precisas resultan las soluciones.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

- La regla de selección, depende del instante del proceso y de la trayectoria recorrida hasta ese momento. Por ejemplo: si en dos iteraciones determinadas, la solución es la misma, la nueva solución de la siguiente iteración no tiene porqué ser necesariamente la misma.

Existen numerosas formas de clasificar metaheurísticas según sus características. Según el marco en el cual se inspiran (naturales o no), el uso que hacen de la memoria (con o sin uso), o el tipo de método que usen como función objetivo (estáticos y dinámicos), entre otros. Actualmente, la manera más popular de clasificarlas, se basa en analizar la cantidad de elementos que toman del espacio de búsqueda por paso que avanzan.

Por un lado, aquellas que utilizan un único elemento por paso que avanzan son denominadas metaheurísticas basadas en trayectoria. Mientras que aquellas trabajan con un conjunto de puntos o población, son llamadas metaheurísticas basadas en población. Esta taxonomía se muestra de forma gráfica en la Figura 1 la cual incluye las técnicas más representativas.

Las siglas de la Figura 1 se corresponden de la siguiente manera: *Simulated Annealing* (SA), *Tabu Search* (TS), *Greedy Randomized Adaptive Search Procedure* (GRASP), *Variable Neighborhood Search* (VNS), *Iterated Local Search* (ILS), *Evolutionary Algorithms* (EA), *Estimation Distribution Algorithm* (EDA), *Particle Swarm Optimization* (PSO), *Scatter Search* (SS), y *Ant Colony Optimization* (ACO).

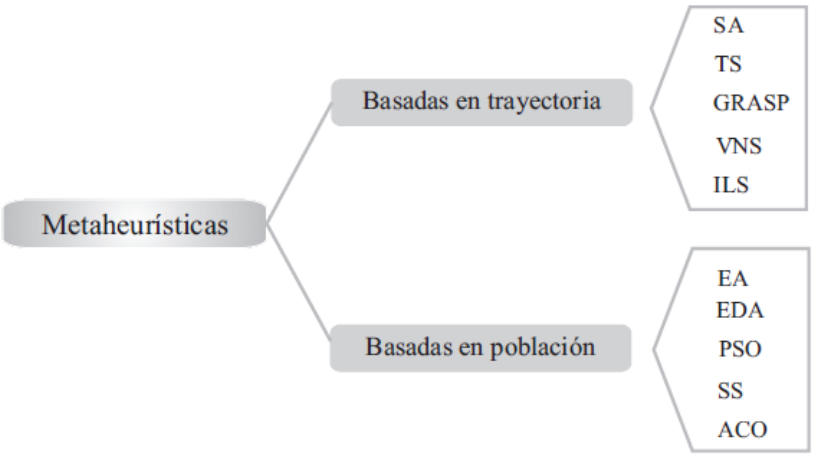

Figura 1: Clasificación de las Metaheurísticas (Salto 2009).

La principal característica de las metaheurísticas basadas en trayectoria, es que parten de una solución y, mediante la exploración del "vecindario", van actualizando la solución actual, formando una trayectoria. La mayoría de estos algoritmos surgen como extensiones de los métodos simples de Búsqueda Local, a los que se les añade algún mecanismo para escapar de los mínimos locales. Normalmente se termina la búsqueda cuando se alcanza un número máximo predefinido de iteraciones, se encuentra una solución con una calidad aceptable, o se detecta un estancamiento del proceso.
Mientras que las basadas en población se caracterizan por trabajar con un conjunto de soluciones, usualmente denominado población en cada iteración, los algoritmos basados en población proveen una forma natural e intrínseca de explorar el espacio de búsqueda, a diferencia de los métodos basados en trayectoria, que únicamente manipulan una solución del espacio de búsqueda por iteración.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

2.2 UN ALGORITMO BASADO EN TRAYECTORIA: HILL CLIMBING

En ciencia de la computación, *Hill Climbing* (Luke S. 2011), (también conocido como ascenso de colinas), es una técnica de optimización matemática que pertenece a la familia de los algoritmos de búsqueda local, y se puede clasificar como metaheurística basada en trayectoria. Es un algoritmo iterativo que comienza tomando una solución arbitraria a un problema para llegar a una solución óptima, o cercana a la óptima. *Hill Climbing* utiliza mecanismos operadores que en cada iteración obtendrán una nueva solución y mecanismos selectores que analizarán si la nueva solución es apta o descartada, según los criterios del algoritmo.
A modo de ejemplo, se puede definir *Hill Climbing* como el proceso deductivo o técnica que utiliza un agente situado en un paisaje para llegar al punto más alto del mismo. Como se explicó en la Sección 2.1, el grupo de metaheurísticas al que pertenece *Hill Climbing* son ciegas y aproximativas, es por eso que se originan los siguientes problemas:

Máximos locales: Picos del paisaje cuya altura es menor que el máximo global.
Llanuras: Áreas extensas del paisaje cuya variación de altura entre bloques es muy pequeña.
Riscos: Áreas del paisaje que cuentan con variaciones de altura negativas.

A continuación se presenta el algoritmo de la técnica *Hill Climbing* en pseudocódigo.

Algoritmo 1: *Hill Climbing* (Luke S. 2011)

```
1: S ← soluciòn candidata/inicial (Inicialiazaciòn)
2: repeat
3:        R ←Tweak[1](Copy(S)) (Operaciones de modificaciòn)
4:        if Calidad(R) > Calidad(S) then (Procedimientos de selección)
5:                S←R
6: until S = solución ideal o tiempo = límite de tiempo
7: return S
```

En las unidades posteriores se exponen modificaciones propuestas por diferentes autores, de modo que mejore performance y diversidad de campo de aplicación del algoritmo *Hill Climbing*.

En la Sección 2.3 se presentan 6 (seis) modificaciones propuestas por diferentes autores, para mejorar la performance y diversidad de campo de aplicación del algoritmo *Hill Climbing*. Estas, formarán parte del respaldo teórico a la propuesta del presente trabajo.

2.3 PROPUESTAS DE MODIFICACIÓN AL ALGORITMO DE HILL CLIMBING

Existen diferentes propuestas de mejora al algoritmo de Hill Climbing, ya sea desde la mejora relativa a eficiencia o diversidad de campo de aplicación. En esta Sección proponemos recorrer algunas de estas propuestas, explicando y señalando brevemente sus ventajas y desventajas.

[1] **El método Tweak:** Función estocástica que recibe como dato de entrada una solución (S) y devuelve como salida una leve modificación de esa entrada (S').

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

2.3.1 STEEPEST ASCENT HILL-CLIMBING

El siguiente algoritmo es una extensión del Algoritmo 1 presentado en la Sección 2.2 . Esta propuesta, en un número menor de iteraciones, explora una mayor cantidad de soluciones, lo cual le permite obtener una mejor solución en un menor tiempo.
A continuación se expone el Algoritmo 2 (Luke S. 2011).

Algoritmo 2: ***Steepest Ascent Hill Climbing***

```
1: n ← número de operaciones de modificación (Tweak)
2: S ← solución inicial/candidata
3: repeat
4:          R ← Tweak(Copy(S))
5:          for n − 1 times do
6:                    W ← Tweak(Copy(S))
7:                    if Calidad(W) > Calidad(R) then
8:                              R←W
9:          if Calidad(R) > Calidad(S) then
10:                   S←R
11: until S = solución ideal o tiempo = límite de tiempo
12: return S
```

Vale destacar que el algoritmo citado es más eficiente que el Algoritmo 1. Sin embargo el mismo es más lineal a la hora de explorar espacios de búsqueda, es decir, tiene el defecto de ser más susceptible a estancamientos en máximos o mínimos locales. Esto habla a las claras de que su campo de aplicación es reducido.

2.3.2 STEEPEST ASCENT HILL-CLIMBING WITH REPLACEMENT

El algoritmo expuesto a continuación, resuelve el problema que se genera al considerar siempre la solución más óptima como candidata a ser alterada en la próxima iteración. En el mismo Luke (2011), agrega una variable (*Best*), donde se almacenará la mejor solución.
Por cada iteración, se generan soluciones, las cuales pueden o no, ser de peor calidad que la solución almacenada en la variable *Best.* De este conjunto se selecciona la mejor (más óptima), para ser alterada en la próxima iteración, disminuyendo de esta manera la probabilidad que se produzcan estancamientos por óptimos locales. En caso de inclinarse por una trayectoria incorrecta, si se cumple la condición de parada por limite de tiempo, el algoritmo devuelve la mejor solución generada al momento (*Best*).

Algoritmo 3: ***Steepest Ascent Hill Climbing (with replacement)***

```
1: n ← número de operaciones de modificación deseadas
2: S ← solución inicial/candidata
3: Best ← S
4: repeat
5:          R ← Tweak(Copy(S))
6:          for n − 1 times do
7:                    W ← Tweak(Copy(S))
8:                    if Calidad(W) > Calidad(R) then
9:                              R←W
10:         S ←R
11:         if Calidad(S) > Calidad(Best) then
12:                   Best←S
13: until Best = solución ideal o tiempo = límite de tiempo
14: return Best
```


Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

En este caso, el autor propone un algoritmo que acrecienta la diversidad del campo de soluciones a explorar, lo cual, logra utilizando un componente de aleatoriedad, es decir, no necesariamente tomará las mejores soluciones siempre (no-elitismo). Cabe destacar que este algoritmo, al igual que el Algoritmo 2, sigue siendo lineal y con tendencia a estancarse.

2.3.3 HILL-CLIMBING WITH RANDOM SEARCH

La modificación propuesta en el Algoritmo 3 disminuye la probabilidad de estancamientos por óptimos locales. El algoritmo expuesto a continuación, disminuye este factor aún más que el propuesto en la Sección 2.3.2, aplicando técnicas estocásticas para generar nuevas soluciones iniciales. De esta manera se amplía la diversidad del espacio de búsqueda, siendo ésta su mayor ventaja respecto a las propuestas citadas con anterioridad.

Algoritmo 4: ***Hill Climbing with random restarts***

```
1: T ← semilla para generar intervalo de tiempo
2: S ← solución inicial/candidata aleatoria
3: Best ← S
4: repeat
5:          time ← rango de tiempo aleatorio generado a partir de T
6:          repeat
7:                      R ← Tweak(Copy(S))
8:                      if Calidad(R) > Calidad(S) then
9:                                  S←R
10:         until S = solución ideal o time = límite tiempo o Tiempo = Tiempo total
11:         if Calidad(S) > Calidad(Best) then
12:                     Best ← S
13:                     S ← solución calculada al azar
14: until Best = solución óptima o tiempo = Tiempo total
15: return Best
```

2.3.4 HYBRID EVOLUTIONARY AND HILL CLIMBING ALGORITHM

Existen muchas formas de mejorar la diversidad del campo de aplicación de los algoritmos basados en trayectoria, una de ellas es crear híbridos utilizando metaheurísticas que siguen la rama de las basadas en población. A continuación se expone el algoritmo 5 Luke (2011) el cual sigue esa línea.

Algoritmo 5: ***Hybrid evolutionary and Hill Climbing algorithm***

```
1: t ← número de iteraciones para Hill Climbing
2: P ← Inicializa población
3: Best ← null
4: repeat
5:          EvaluarAptitudIndividuos(P)
6:          for each individual Pi ∈ P do
7:                      Pi ← Hill-Climb(Pi ) for t iterations
8:                      if Best = null or Aptitud(Pi) > Aptitud(Best) then
9:                                  Best ← Pi
10:         P'← Breed(P)
11:         P ← Join(P, P')
12:         until Best = Solución óptima o Tiempo = Límite de tiempo
13:         return Best
```


Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

El algoritmo define una variable (t) la cual indica el número de iteraciones que se aplicará *Hill Climbing*. Luego genera una población (P) inicial, y crea una variable (Best) para almacenar la mejor solución. En el primer bucle el algoritmo evalúa la aptitud para cada individuo de la población, luego para cada uno de ellos, se aplica *Hill Climbing* t cantidad de veces y se guarda la mejor solución obtenida en la variable Best. Finalmente la función Breed, genera individuos de la población P utilizando mutaciones. La función Join combina mitad de individuos de la población P y mitad de individuos de la población obtenida de la función breed (P), almacenada en P' y, los almacena en la población (P). Se repiten los pasos anteriores hasta que se encuentre la solución ideal o se termine el tiempo límite de ejecución, devolviendo como resultado la variable best con la mejor solución encontrada.

2.3.5 HIBRIDACIÓN DE EVOLUCIÓN DIFERENCIAL UTILIZANDO HILL CLIMBING

En (Hernández S. et al. 2012), los autores exponen una hibridación entre una metaheurística de Evolución Diferencial basada en la población y una Hill Climbing basada en la trayectoria. En su producción se explican los dos tipos de metaheurísticas y se exponen los cambios realizados a un algoritmo Hill Climbing, como así también, tablas donde se muestran los resultados obtenidos en experimentos propuestos.

2.4 THE GENETIC ALGORITHM

Los algoritmos genéticos son metaheurísticas basadas en población. Parten del proceso genético que utilizan los seres vivos para sobrevivir y evolucionar. De forma análoga, las soluciones a un problema se conciben formando parte de un proceso similar al de la vida de los seres vivos en la cual rige el principio Darwiniano de selección natural: Se seleccionan dos padres, se reproducen, y finalmente se obtienen dos hijos que ocasionalmente mutan.
A continuación se expone el Algoritmo 6, algoritmo genético (Luke S. 2011).

Algoritmo 6: ***The Genetic Algorithm (GA)***

```
1: popsize ← Tamaño de la población
2: P ← {}
3: for popsize times do
4:         P ← P ∪{nuevo individuo aleatorio}
5: Best ← null
6: repeat
7:         for each individual Pi ∈ P do
8:                 EvaluarAptitudPoblación(Pi)
9:                 if Best = null or Aptitud(Pi) > Aptitud(Best) then
10:                        Best ← Pi
11:        Q ← {}
12:        for popsize/2 times do
13:                Parent Pa ← SeleccionConReemplazo(P)
14:                Parent Pb ← SeleccionConReemplazo(P)
15:                Children Ca, Cb ← Crossover(Copy(Pa), Copy(Pb))
16:                Q ← Q ∪{Mutate(Ca),Mutate(Cb)}
17:        P ← Q Se reemplaza P por Q
18: until Best = solución óptima o tiempo = límite de tiempo
19: return Best
```

Este algoritmo define una variable popsize la cual indica el tamaño de la población inicial. Inmediatamente después se genera esa población P, y se declara una variable Best para almacenar la mejor solución.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

Consiguientemente el algoritmo describe un bucle que contiene dos estructuras iterativas:

- En la primera se evalúa la aptitud para cada individuo de la población y se determina cual es el que tiene mayor fitness.
- En la segunda se crea una variable población (Q) que contendrá una nueva generación de individuos que se generará a partir de operaciones de cruza (Crossover) y mutación (*Mutate*) entre individuos de la población inicial (P).

El bucle continuará iterando hasta que la solución encontrada sea la óptima o hasta que se llegue al tiempo límite de ejecución. Al final se devuelve la variable Best con la mejor solución encontrada.

En la actualidad los Algoritmos Genéticos Celulares (cGAs) son una de las herramientas de optimización más populares. En (Soria et al. 2014) los autores proponen un estudio comparativo, y exponen los resultados en la performance de diferentes operadores de recombinación en un cGA aplicado a una serie de problemas académicos de optimización discretos.

3. NUESTRA PROPUESTA DE HIBRIDACIÓN: GENÉTICO-HILL CLIMBING

En la Sección 2.3, se ha incorporado un estado del arte, presentando 7 (siete) trabajos de diferentes autores basados en la técnica metaheurística *Hill Climbing*. Estos trabajos parten de un algoritmo básico para luego proponer modificaciones al mismo, de forma tal, que cada una de las propuestas descriptas anteriormente, presentan una mejora relativa a eficiencia o diversidad de campo de aplicación al algoritmo con respecto a la presentada en la Sección anterior.
Como se ha visto anteriormente, el Algoritmo 5 (*Hybrid evolutionary and Hill Climbing algorithm)* es un híbrido, es decir, combina metaheurísticas basadas en trayectoria y en población. Se ha destacado que la metaheurística basada en población utiliza la técnica básica de la computación evolutiva, de carácter elitista.

Lo que se propone en este trabajo es extender el Algoritmo 5 de modo que se implemente el concepto de computación genética (una rama de la computación evolutiva) utilizado en el Algoritmo 6 (*The genetic algorithm*), y de esta forma diversificar su campo de aplicación.

La Tabla 1 señala tres diferencias sustanciales entre el Algoritmo 5 y el algoritmo propuesto. Dicho análisis radica en la forma de determinar cuáles serán los nuevos miembros de la nueva generación de individuos o hijos que se obtenga en cada iteración:

A primera vista los operadores utilizados para realizar ese proceso en el cual se obtiene una nueva población son diferentes. Por un lado, los algoritmos evolutivos realizan operaciones de selección, mutación y reemplazo, mientras que los genéticos añaden recombinación entre medio de selección y mutación.

La elección de los métodos de selección y reemplazo, para ambos algoritmos tendrá implicancia en la caracterización de los mismos como elitistas o no.

Para el algoritmo propuesto:

- El método de selección será de tipo natural (no elitista), esto significa que, se tomarán todos los individuos como padres de a pares sin considerar su aptitud.
- El método de recombinación será el *Crossover*, esto significa, que se realizará una cruza entre los genotipos de los padres seleccionados para obtener un nuevo genotipo.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

- El método de mutación será *mutation*, que acorde al valor de una variable aleatoria que será diferente en cada iteración modificará el genotipo obtenido anteriormente.
- El método de reemplazo será de tipo no elitista también, ya que, se tomarán todos los individuos obtenidos de la recombinación sin considerar su aptitud para reemplazar a los padres.

Para el caso del Algoritmo 5:

- El método de selección es de tipo elitista, esto significa, que se tomará un pequeño grupo de individuos en base a su aptitud.
- No se realiza recombinación.
- El método de mutación será el *mutation*, contenido en la función *tweak*, cuya utilidad será devolver un nuevo individuo mutado casualmente.
- El método de reemplazo es de tipo elitista también, ya que, se reemplazará toda la población inicial o actual, de modo que los individuos con menor aptitud tendrán mayor probabilidad de ser descartados.

Cabe destacar que el hecho de utilizar métodos de reemplazo no elitistas aumenta drásticamente la probabilidad de que se generen poblaciones más variadas con respecto a la aptitud de los individuos. Esto se logra gracias al hecho de que no se tienen en cuenta sólo los individuos más aptos, lo cual amplía notablemente el espacio de búsqueda.

En la Tabla 1 se muestran las principales diferencias entre el híbrido presentado en la Sección 2.6 y el propuesto para la realización de este trabajo.

Tabla 1: Diferencias entre Evolutivo-*Hill Climbing* y Genético-*Hill Climbing*.

MÉTODO	**Evolutivo-*Hill Climbing***	**Genético-*Hill Climbing***
de Selección	Selección elitista de los individuos más aptos.	Selección natural de padres. No elitista.
de Generación de Poblaciones	1. Mutación	1. Recombinación 2. Mutación
de Reemplazo	Reemplazo de los individuos descartados por los generados.	Reemplazo de todos los padres por los hijos.

A continuación se listan las principales características del algoritmo propuesto:

- Su estructura general es similar a la del Algoritmo Genético (Algoritmo 6) con la diferencia de que en el bucle principal se implementa *Hill Climbing* a cada individuo de la población actual.
- La población inicial es generada aleatoriamente.
- Cuenta con una variable numérica la cual define la cantidad de iteraciones *Hill Climbing*.
- Garantiza el resguardo de la mejor solución generada hasta el momento.
- En la generación de hijos se asegura que ningún padre sea elegido dos veces.
- Utiliza técnicas de recombinación de padres y mutación de hijos para generar una nueva población actual.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

- Como toda técnica metaheurística, no asegura el hallar siempre la solución óptima a un problema.

En la siguiente Sección se desarrollará el pseudocódigo para el algoritmo propuesto y se describirá el mismo paso por paso.

4. DESCRIPCIÓN DEL ALGORITMO DESARROLLADO EN BASE A NUESTRA PROPUESTA

El algoritmo exhibido a continuación es producto de la combinación entre el Algoritmo 6 (Algoritmo Genético), y el Algoritmo 1 (Algoritmo *Hill Climbing*).

Algoritmo 7: ***Hybrid Genetic and Hill Climbing Algorithm***

```
1: t ← número de iteraciones para Hill Climbing
2: popsize ← Tamaño de la población
3: P ← {}
4: Best ← null
5: for popsize times do
6:         P ← P ∪{nuevo individuo aleatorio}
7: repeat
8:         for each individual Pi ∈ P do
9:                 repeat
10:                        R ←Tweak(Copy(Pi)) (Operaciones de modificaciòn)
11:                        if Calidad(R) > Calidad(Pi) then (Procedimientos de selección)
12:                        Pi←R
13:                until t
14:                if Best = null or Aptitud(Pi) > Aptitud(Best) then
15:                        Best ← Pi
16:        Q ← {}
17:        for popsize/2 times do
18:                Parent Pa ← SeleccionConReemplazo(P)
19:                Parent Pb ← SeleccionConReemplazo(P)
20:                Children Ca, Cb ← Crossover(Copy(Pa), Copy(Pb))
21:                Q ← Q ∪{Mutate(Ca),Mutate(Cb)}
22:        P ← Q Se reemplaza P por Q
23: until Best = solución óptima o tiempo = límite de tiempo
24: return Best
```

Para una mejor comprensión del algoritmo propuesto se procede a describir su funcionamiento paso a paso:

Inicialmente se inicializan las variables *t*, *popsize*, P y *Best*:

- La variable *t*, determina la cantidad de iteraciones *Hill Climbing* que se aplicarán para cada individuo de la población actual.
- La variable *Popsize* define la cantidad de individuos de la población (tamaño de la población).
- La variable P es un arreglo de datos, donde se almacenan los individuos.
- La variable *Best* tiene la función de almacenar la mejor solución encontrada hasta el momento. Hasta que el mismo no se encuentre, se le asigna un valor nulo.

El siguiente paso corresponde a generar la población inicial. Para cumplir esta tarea, se utiliza un bucle *for*, el cual por cada iteración agrega al arreglo P, un nuevo individuo generado aleatoriamente.

Una vez generada la población inicial, el programa ingresa al bucle principal del algoritmo. Dentro de éste, la primer acción a realizar es la de recorrer el arreglo P utilizando un bucle,

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

donde a cada uno de sus individuos se le aplica la técnica heurística *Hill Climbing t* cantidad de veces. El resultado de este proceso es la optimización (o no) del individuo Pi. Una vez terminado el paso anterior, se procede realizar una comparación entre la aptitud del individuo actual (Pi) y la aptitud del individuo almacenado en la variable *Best.* Si la calidad de Pi es superior a la de *Best* el valor de esta última es reemplazado.

El subsiguiente paso es la generación de una población para la próxima iteración. Por lo tanto, para llevar a cabo esta operación, se crea una nueva variable (Q) la cual se corresponde con un arreglo donde se almacenan los hijos generados para la nueva población. En este proceso se crea un bucle *for* el cual será iterado *popsize/2* veces, a fin de que por cada iteración se generen dos hijos nuevos, estos son el resultante de la combinación de dos individuos, elegidos de la población actual (P). La forma en que es realizada la selección asegura que un mismo individuo no sea seleccionado dos veces.

Una vez elegidos los padres Pa *y* Pb se utiliza una función para la combinación de los mismos, aplicando algún operador de recombinación, como por ejemplo cruce de un punto. Producto de dicha operación, se obtienen dos nuevos individuos, hijos de Pa y Pb (Ca y Cb). Una vez generados, por consiguiente, se procede a realizar sobre estos una operación de mutación, para luego ser almacenados en el arreglo Q.

Una vez terminado el bucle for para *popsize/2*, el tamaño de Q será igual a *popsize.* El contenido del arreglo Q reemplazará la población P, obteniéndose así una nueva generación o población actual para la siguiente iteración del bucle principal del algoritmo.

Finalmente, el bucle principal del algoritmo, iterará hasta que una condición de parada (solución óptima, limite de tiempo o evaluaciones, etc) se cumpla. Al finalizar su ejecución, el objetivo del algoritmo, de encontrar una solución a un problema determinado se habrá cumplido. Como en toda metaheurística, esta solución podrá ser (o no) la solución ideal.

5. CONCLUSIONES Y TRABAJO FUTURO

En el presente trabajo de investigación se ha llevado a cabo el estudio de una técnica de optimización llamada *Hill Climbing*, que se puede clasificar dentro de las metaheurísticas basadas en la trayectoria. Se presentaron distintos tipos de propuestas de algoritmos, desde un algoritmo básico *Hill Climbing* y sus extensiones, pasando por un híbrido Evolutivo-*Hill Climbing*, como así también un Algoritmo Genético.
Luego de analizar y estudiar estos distintos algoritmos existentes, se llegó a la conclusión de que es posible realizar la implementación de un algoritmo híbrido, entre una metaheurística basada en la población (Algoritmo Genético) y una basada en la trayectoria (*Hill Climbing*), para proveer una propuesta de solución a problemas de optimización combinatoria. El algoritmo resultante (Genético-*Hill Climbing*) contempla una mayor cantidad de soluciones a evaluar, debido a que utiliza selección natural (diversificadora), ampliando así el espacio de búsqueda. En este contexto, donde se analizaron antecedentes y trabajos relacionados, que constituyen el soporte del presente trabajo, se puede enunciar que, la aplicación práctica de la propuesta es posible, y que la misma se podrá implementar en una mayor cantidad de problemas, en comparación con los trabajos presentados y referenciados en la Sección 2.

Como trabajo futuro, se aplicará y analizará el algoritmo resultante del presente trabajo de investigación, para evaluar y comparar su performance con respecto a otras metaheurísticas, ya sean, basadas en población, como en trayectoria (*Hill Climbing*). Esto permitirá profundizar su validación y determinar si resulta una mejora efectiva a aplicar dentro del

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

campo práctico de la resolución de problemas. También, se analizará la alternativa de utilizar esta propuesta como base para la hibridación de distintas metaheurísticas.

REFERENCIAS

GAREY, M. R., y JOHNSON, D. S. 1979. Computers and Intractability. A Guide to the Theory of NP-Completeness. A Series of Books in the Mathematical Sciences, W. H. Freeman and Company, San Francisco.

GLOVER F. 1986. Future paths for integer programming and links to artificial intelligence. Computers Operations Research. https://doi.org/10.1016/0305-0548(86)90048-1

GOLDBERG D. E. 1989. Genetic algorithms in search, optimization, and machine learning, Addison Wesley.

HERNÁNDEZ S., LEGUIZAMÓN G., y MEZURA-MONTES E. 2012. Hibridación de Evolución Diferencial utilizando Hill Climbing para resolver problemas de optimización con restricciones. Universidad Nacional de San Luis.

LUKE S. 2011. Essentials of metaheuristics. Lulu, 2009. Available for free at http://cs. gmu. edu/_sean/book/metaheuristics.

MARTÍ R. 2003. Procedimientos metaheurısticos en optimización combinatoria. Matemátiques, Universidad de Valencia, 1(1):3–62.

MORILLO D., MORENO L., y DÍAZ J. 2014. Metodologías analíticas y heurísticas para la Solución del Problema de Programación de Tareas con Recursos Restringidos (RCPSP): una revisión. Parte 1. Ingeniería y Ciencia, 10(19), 247-271. https://doi.org/10.17230/ingciencia.10.19.12

OSMAN H. I. y Kelly J. P. 1996. Meta-heuristics: an overview. In Meta-Heuristics, pp 1–21. Springer. https://doi.org/10.1007/978-1-4613-1361-8_1

REEVES C. R. 1993. Modern heuristic techniques for combinatorial problems. John Wiley & Sons, Inc.

SAIT S. M. y YOUSSEF H. 1999. Iterative computer algorithms with applications in engineering: solving combinatorial optimization problems. IEEE Computer Society Press.

SALTO C. 2009. Metaheurísticas híbridas-paralelas para problemas industriales de corte, empaquetado y otros relacionados. UNSL

SEAN, L. 2009. Essentials of metaheuristics. Lulu: Department of Computer Science, George Mason University.

SORIA, C. L., Pandolfi, D. R., & Villagra, S. M. 2014. Algoritmos genéticos celulares con operadores de recombinación aplicados a problemas de optimización discretos. Informes Científicos-Técnicos UNPA, 6(3), 1-21.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.