Capacitated minimum spanning tree

Erick Salvador Alvarez Valencia, *CIMAT A.C.*

***Index Terms*—Árbol de expansión mínima con capacidades, metaheurísticas de trayectoria, operadores genéticos, multi dyn.**

I. Introducción

En el presente reporte se describirá la implementación de una metaheurística de trayectoria enfocada en resolver el problema del árbol de expansión mínima con capacidades. Se describirá el problema desde una versión más sencilla, posteriormente se hablará sobre lo desarrollado hasta la fecha para atacar dicho problema, así como el algoritmo implementado en este proyecto y finalmente se mostrarán los resultados obtenidos del algoritmo a un conjunto de datos con grafos de 40 y 80 nodos.

mds
Junio 08, 2018

II. Definición del problema

Para definir el problema que se trató en este proyecto primeramente se hablará de una versión más clásica del mismo, la cual es el árbol de expansión mínima.

II-A. Árbol de expansión mínima

Dado un grafo conexo, no dirigido y pesado G = (E, V). Un árbol de expansión es un árbol compuesto por todos los vértices y algunas (posiblemente todas) las aristas del grafo. Al ser creado un árbol no existirán ciclos, además debe existir una ruta entre cada par de vértices. Para que este árbol sea el de expansión mínima debe cumplir que la suma de todas sus aristas debe ser la mínima posible, es decir, dato $C = \{M_{i,j}\}$ el conjunto de todas las aristas del grafo, se quiere encontrar el árbol recubridor que cumpla

$$\min_{M} \sum_{i} \sum_{j} M_{i,j} \tag{1}$$

Para este problema no existe restricción sobre cuál debe ser la raíz del árbol resultante. En la actualidad existen algunos algoritmos que resuelven este problema en tiempo polinomial, tales como el algoritmo de Kruskal [6] o el algoritmo de Prim [7].

II-B. Árbol de expansión mínima con capacidades

El interés de este proyecto recae en una versión extendida del problema anterior, la cual consiste en dado un nodo central N_0, un conjunto de nodos $(N_1, ..., N_m)$, una matriz de pesos $M_{i,j}$, un conjunto de capacidades $(C_1, ..., C_m)$ para cada nodo y una constante K, se quiere encontrar un árbol de expansión mínima para el grafo G tal que:

$$\begin{aligned} &\min_{M} \sum_{i} \sum_{j} M_{i,j} \\ &s.\ a.\ C_j + \sum_{k \in Sub(j)} d_k \leq K \end{aligned} \tag{2}$$

En términos generales, lo anterior nos menciona que queremos encontrar el árbol de expansión mínima de un grafo y que además cumpla con tener como raíz al nodo N_0 y que la suma de las capacidades de cada sub-árbol conectado al nodo N_0 sea menor o igual a K.

Dada la restricción adicional de las capacidades en los nodos del árbol se demostró que este problema se convierte a uno NP-Completo (Papadimitriou, 1978). Y aunque en la actualidad existen algoritmos heurísticos que generan buenas soluciones ante extre problema, ninguno logra resolverlo de la manera más óptima posible, como ejemplo de un algoritmo que hace lo anterior tenemos el de Esau-Williams [8].

III. Trabajos previos

Previo a la implementación de un algoritmo para atacar este problema se hizo una pequeña revisión del estado del arte con respecto al área de metaheurísticas de trayectoria y metaheurísticas poblacionales que atacan al problema. A continuación se describe un resumen de dicho análisis.

Raidl et al. [2] proponen un algoritmo genético tipo *Steady state* que use una representación vectorial de los árboles de expansión sin violar las restricciones de capacidad, en dicho artículo comentan que la representación usada genera mejores resultados que las existentes hasta ese momento, de igual manera hacen la propuesta de una heurística constructiva así como operadores de cruza y mutación que generan individuos factibles para el problema. Como nota importante, de este artículo se obtuvieron los operadores y la heurística implementados en el presente proyecto.

José Torres et al. [5] proponen un algoritmo de recocido simulado el cual trabaja las funciones de perturbación y energía de tal manera que genere soluciones factibles para el CMST. Se utiliza una representación que garantiza implícitamente que todas las perturbaciones producen un árbol, además de que ellos hablan sobre el hecho de que muchas heurísticas proponen representaciones que no violan la capacidad del árbol y por lo mismo siempre tienden a llegar a óptimos locales estos algoritmos, por lo que el algoritmo se mueva por todo el espacio de búsqueda pero penalizando las violaciones de las restricciones con la función de energía.

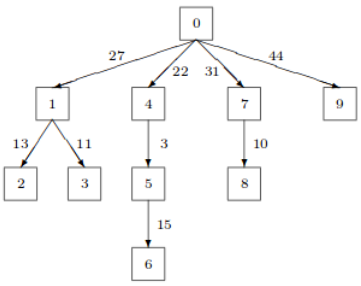
Estéfane y Manoel [3] proponen un algoritmo genético tipo *Steady state* el cual usa la misma representación, la misma función de heurística constructiva y el mismo operador de mutación que en [2] pero proponen un operador de cruza el cual transfiere nodos de sub-árboles a otro sub-árboles, así como un nuevo operador de mutación en donde se intercambian varios sub-árboles.

Efrain Ruiz et al. [4] proponen un algoritmo genético basado en un algoritmo generador de llaves aleatorias *(random-key genetic algorithms)* el cual trabaja las representaciones de los individuos como vectores que contienen valores reales, normalmente inicializados con una distribución uniforme y posteriormente existe un decoder que mapea dicho genotipo a un vector de permutación el cual para este caso debe ser un árbol factible y que además puede calcular su valor de aptitud. Esta representación es trabajada por un algoritmo genético que cuenta con operadores de cruza y mutación aunque en este artículo introducen el concepto de $mutantes$ los cuales son vectores de llaves aleatorias que generan ruido a los individuos para que no se queden en óptimos locales, es decir, el mismo concepto de mutación.

IV. Metaheurística poblacional para el CMST

En este proyecto se trabajó con un algoritmo de metaheurística de trayectoria con un enfoque de control de diversidad en la población a lo largo del tiempo propuesta por el Dr. Carlos Segura llamada: *Multi Dynamic* [1] además de utilizar los operadores cruza, mutación y la heurística constructiva propuestos en el artículo de Gunther R. Raidl y Christina Drexel mencionado en la sección anterior para encontrar soluciones factibles y de buen costo en el problema del CMST. Lo primero que se describirá son estos operadores y la heurística constructiva de forma conceptual y finalmente el *Multi Dynamic* previo a dar los resultados de la implementación.

Primeramente, la representación de un individuo (árbol) se hace mediante un arreglo del tamaño igual al número de nodos del grafo y en general, la idea es que para cada nodo i el arreglo en dicha posición apunte al padre de i (nodo previo). Por ejemplo, para el siguiente árbol.



(a) Figura 1. Ejemplo de CMST.

Su representación sería $p = (0, 0, 1, 1, 0, 4, 5, 0, 7, 0)$, de esta forma se vendría trabajando la población de individuos en el algoritmo.
Para la heurística constructiva de un individuo se tiene el siguiente algoritmo.

Algorithm 1 Heurística constructiva.

procedure GENERATEIND(n)
  p $\leftarrow$ {-1, -1, ..., -1}$_{i=1}^{n}$
  Sea S el conjunto de nodos actualmente conectados y sea U el conjunto de nodos sin conectar. Inicialmente S contiene el nodo raíz y U contiene a los demás nodos.
  while U no vacío **do**
    Selecciona un nodo j de S aleatoriamente.
    Intenta seleccionar un nodo i aleatorio de U el cual pueda ser conectado a j sin violar las restricciones de capacidad. Si tal nodo no existe, remueve j de S y vuelve a empezar el ciclo si S no está vacío.
    Si S se encuentra vacío, el algoritmo no fue capáz de generar un individuo factible.
    Conecta i con j ($p[i] = j$) y mueve i de U a S.
  return p.

El algoritmo anterior busca construir un individuo de manera aleatoria pero que sea factible en el sentido de cumplir las restricciones de capacidades que plantea el problema. De igual manera se introduce un enfoque greedy en el paso de la selección del nodo i, en donde se buscan todos los posibles nodos que se pueden conectar a j, se ordenan por el peso de la arista y de dicho conjunto se esconje un nodo aleatorio dentro de los primeros α nodos, esto anterior se hace con el fin de que la solución también tenga un buen valor de aptitud pero no pierda el toque de aleatoriedad.
Para la parte de verificar que la unión del nodo i con el j no viole las restricciones de capacidad se hace uso del algoritmo *Union-Find* [9] el cual implementa dos funciones para trabajar de manera eficiente con conjuntos conexos, y las cuales ayudan a saber si dos nodos se encuentran en el mismo conjunto conexo, así como el tamaño de dicho conjunto.

A continuación se describe el algoritmo de cruza de individuos.

Algorithm 2 Cruza.

procedure CROSSOVER(p, q, n)
  r $\leftarrow$ {-1, -1, ..., -1}$_{i=1}^{n}$
  Para todos los nodos tales que $p[i] = q[i]$ hacer $r[i] = p[q]$.
  Sea U el conjunto de nodos que aún no tienen predecesor, es decir, $U = \{i | r[i] = -1\}$
  while U no vacío **do**
    Selecciona un nodo aleatorio i de U.
    Selecciona $j \in \{p[i], q[i]\}$ aleatoriamente.
    Si los nodos i y j ya se encuentran conectados de alguna manera o se viola la restricción de capacidad por hacer la conexión, ir al siguiente paso, de lo contrario conectar i con j ($p[i] = j$), remover i de U y comenzar de nuevo el ciclo.
    Intenta hacer el paso anterior pero con el otro predecesor. Si no es posible esto, mover i de U a una cola Q de nodos que no se pudieron conectar.
  for i $\leftarrow$ 1 to $|Q|$ **do**
    Intenta encontrar un nodo j tal que se pueda realizar la conexión con i sin violar ninguna restricción de capacidad ni generar ciclos.
    Si se encontró un nodo j hacer la conexión con i ($p[i] = j$) y comenzar de nuevo el ciclo.
    No se pudo encontrar un individuo factible que fuera descendiente de p y q
  return r.

El algoritmo de cruza anterior busca generar un descendiente de p y q basado en el enfoque de heredar todas las aristas que sean iguales en los padres y a los nodos restantes buscar una conexión aleatoria que no viole las restricciones de capacidades.
Para hacer esta elección aleatoria se usa un enfoque heurístico donde se buscan todos los nodos con posible conexión a i y posteriormente se elige uno aleatoriamente que esté dentro de los α mejores, tal como se hizo con la heurística constructiva.

Posteriormente se muestra el algoritmo de mutación de individuos.

Algorithm 3 Mutación.

procedure MUTATION(p, n)
  Selecciona un nodo aleatorio i que no sea la raíz.
  Si se remueve la arista que conecta a i con su padre el árbol se convertirá en dos componentes conexas. Sea S el conjunto de todos los nodos que no sean parte del sub-árbol generado por i (incluyendo al nodo raíz).
  Remueve $p[i]$ de S.
  while S no vacío **do**
    Selecciona un nodo j de S aleatoriamente.
    Si la conexión de i con j viola alguna restricción de capacidad, remueve i de S y comienza de nuevo.
    Realiza la unión de i con j ($p[i] = j$) y retorna como solución a p
  No se encontró una solución factible como mutación de p.

El algoritmo anterior intenta mutar un árbol tomando de manera aleatoria un sub-árbol del mismo y uniéndolo en otro lugar que no sea donde se encontraba originalmente, esto sin violar ninguna restricción de capacidad.
Al igual que los dos algoritmos anteriores, se usó un enfoque greedy en el paso 7 donde encontramos un nodo j al cual conectar i, se almacena j en un conjunto de soluciones viables y de ellas se elige aleatoriamente una solución dentro de las α mejores, esto para fomentar la creación de buenas soluciones sin perder la parte aleatoria.

Como siguiente parte se hablará sobre la implementación del algoritmo genético con enfoque de control de diversidad *Multi-Dyn* el cual intenta mantener un grado de diversidad en la población que va disminuyendo a lo largo del tiempo, esto para que en etapas tempranas de la ejecución se haga una ámplia búsqueda en el dominio y en etapas tardías se haga más explotación de las soluciones encontradas, con ello se busca evitar la convergencia prematura a malas soluciones.
A continuación se mostrará el pseudocódigo del algoritmo *Multi-Dyn*.

Algorithm 4 Selección de individuos usando estrategia de diversidad.

procedure MULTIDYN($currentMembers$, n)
  newPop $\leftarrow$.
  sortByFitness(currentMembers).
  newPop $\leftarrow$ newPop $\bigcup$ currentMembers[0].
  currentMembers $\leftarrow$ currentMembers - currentMembers[0].
  while |newPop| $< n$ **do**
    DCN $\leftarrow$ getDCN(currentMembers, newPop).
    D $\leftarrow$ DI - DI * $\frac{currentTime}{maxTime}$.
    sortAsc(DCN).
    nonDominated $\leftarrow$ {}.
    fitnessMax $\leftarrow$ ∞.
    for i = 1 $\rightarrow$ DCN.size() **do**
      if DCN[i].diversity $\geq$ D **then**
        if currentMembers[i].fitness **then**
          nonDominated $\leftarrow$ nonDominated $\bigcup$ DCN[i].id.
          fitnessMax $\leftarrow$ currentMembers[DCN[i].id].fitness.
    idx $\leftarrow$ randomVal(nonDominated).
    newPop $\leftarrow$ newPop $\bigcup$ currentMembers[idx].
    currentMembers $\leftarrow$ currentMembers - currentMembers[idx].
  return $newPop$.

El algoritmo anterior muestra el esquema de selección de individuos basados en diversidad, primeramente hay que comentar que $currentMembers$ es una población constituida por la población en un tiempo t unida a otra población de descendencia construida en base a la población en el tiempo t. Lo primero que se hace es preservar al individuo que da el mejor fitness y quitarlo de $currentMembers$. Posteriormente para añadir un miembro a la nueva población se obtiene el

DCN de la población de $currentMembers$, dicho DCN no es más que el valor de mínima distancia que genera un individuo de $currentMembers$ con respecto a los individuos de la nueva población, dicha distancia puede manejarse de varias formas, para este trabajo se usó la distancia de Hamming. Estos valores nos miden la diversidad que generan los individuos con respecto a los ya considerados en la nueva población, esto es muy importante ya que este será un factor a tener en cuenta para escoger nuevos individuos. El siguiente paso será la parte multi-objetivo ya que se buscan individuos que tengan un buen fitness y a la vez una buena diversidad, y para ello se buscan en la población $currentMembers$ los individuos no dominados con respecto a estas dos características, es decir, los que se encuentren en el frente de pareto del conjunto. Una vez que se tiene calculado dicho frente, simplemente se elige un individuo al azar, se añade a la nueva población y se elimina del conjunto $currentMembers$.
Otra cosa a tener en cuenta es el hecho de que además de lo anterior comentado, se está imponiendo una penalización, la cual es medida por la variable D. Dicha penalización ignora a los individuos que tengan una diversidad menor a dicho D, pero también esta variable se va reduciendo linealmente con el tiempo transcurrido de la ejecución, esto es muy importante ya que en generaciones tempranas este valor tiende a ser alto y por lo cual ignora muchos individuos que tengan diversidad muy pequeña, pero al transcurrir el tiempo, esto se va reduciendo y deja pasar individuos más pobres en diversidad. Esto anterior sirve para hacer una búsqueda más exploratoria en etapas tempranas de la ejecución y una búsqueda más intensiva en etapas tardías, de esta forma es más fácil encontrar el óptimo global de la función.
A continuación se mostrará la función principal del algoritmo genético, en donde se incluye la implementación anterior.

Algorithm 5 Algoritmo genético.

```
1: procedure GeneticAlgorithm(n)
2:     pop ← randomInitialization(n).
3:     while Not convergence do
4:         selected ← Evaluation(pop).
5:         offspring ← MatingSelection(selected). ▷ Generar una población de hijos mediante un torneo binario, cruza y mutación a la población actual.
6:         pop ← MultiDyn(pop ⋃ offspring).
7:     return pop.
```

Podemos ver que el algoritmo anterior parte del concepto de un algoritmo genético, ya que se se genera una población de descendencia usando los operadores de cruza y mutación, la parte que cambia es la generación de la nueva población, en donde se aplica el enfoque multi-dynamic descrito anteriormente.

V. Ejecuciones y resultados

Para probar la implementación del algoritmo anterior se utilizó el conjunto de datos contenidos en la siguiente dirección $http : //people.brunel.ac.uk/\ mastjjb/jeb/orlib/capmstinfo.html$ en donde se encuentran conjuntos para el problema CMST con capacidades fijas en los nodos, es decir, todos los nodos del grafo tienen la misma capacidad y conjuntos para la versión donde las capacidades son variables, ambos conjuntos contienen 5 grafos con 40 nodos y 5 con 80, donde en cada grafo se proporciona la matriz de adyacencia del mismo y se asume siempre que el nodo raíz es el último de la matriz de adyacencia además para la versión de capacidades fija las capacidades de los nodos siempre valen 1. A parte de lo anterior descrito sobre el conjunto de datos, los problemas se dividen en base al valor de la restricción K la cual tiene 3 valores $\{3, 5, 10\}$. Como nota cabe mencionar que en este proyecto sólo se trabajó la versión fija del CMST.
La configuración del algoritmo genético para las ejecuciones tanto de 40 como 80 nodos es la siguiente:

1. **Número de ejecuciones por instancia:** 30.
2. **Tiempo por ejecución:** 1 hora.
3. **Tamaño de la población:** 100 individuos y cada generación de hijos era de 100 individuos también.
4. **Probabilidad de cruza:** 80 %.
5. **Probabilidad de mutación:** 50 %.
6. **Valor de α usado en cruza y mutación:** 50 %.
7. **Valor de DI:** 10 para los conjuntos de 40 nodos, 20 para los de 80.
8. **Tipo de selección:** Torneo binario.

La probabilidad de mutación se elige alta porque en la implementación del genético en el artículo también se elige de esta manera, y a final de cuentas se quiere hacer una comparación de ambos métodos así como de las soluciones encontradas.

A continuación se muestra una tabla con los resultados obtenidos de las 30 ejecuciones por instancia para el conjunto de grafos de 40 nodos.

Cuadro I: Resultados del algoritmo genético para el conjunto de grafos de 40 nodos y valores de $K = \{3, 5, 10\}$.

Instancia	K	Mejor resultado	Peor resultado	Media	Desviación estándar
tc40-1	3	746	762	749	5.48
tc40-2	3	723	738	726	4.54
tc40-3	3	716	732	722	5.29
tc40-4	3	787	795	790	2.84
tc40-5	3	747	755	749	2.87
tc40-1	5	594	610	599	4.65
tc40-2	5	593	609	599	4.90
tc40-3	5	590	616	596	9
tc40-4	5	621	632	625	4.21
tc40-5	5	618	630	623	4.90
tc40-1	10	508	517	510	3.04
tc40-2	10	503	512	506	3.23
tc40-3	10	512	518	514	2.27
tc40-4	10	514	518	515	2.1
tc40-5	10	520	525	522	2.01

Ahora se muestran los resultados presentados por el artículo para el mismo conjunto de datos.

Cuadro II: Resultados del artículo para el conjunto de grafos de 40 nodos y valores de $K = \{3, 5, 10\}$.

Instancia	K	Mejor resultado	Peor resultado	Media	Desviación estándar
tc40-1	3	742	-	742	0
tc40-2	3	717	-	718	1.52
tc40-3	3	716	-	716	0.42
tc40-4	3	778	-	779	1.7
tc40-5	3	741	-	741	0.63
tc40-1	5	586	-	587	1.05
tc40-2	5	579	-	579	0.63
tc40-3	5	577	-	577	0
tc40-4	5	617	-	617	0.32
tc40-5	5	602	-	604	1.08
tc40-1	10	498	-	498	0
tc40-2	10	490	-	490	0.84
tc40-3	10	500	-	501	2.9
tc40-4	10	512	-	512	0.7
tc40-5	10	504	-	504	0

Se puede notar que en general se obtuvieron resultados comparables con los que presenta el artículo para este conjunto de datos, aunque en casi todas las instancias se logró un resultado ligeramente peor que el que ellos presentan, además la variabilidad de los resultados fue mayor, en algunos casos en el artículo se presenta una desviación estándar de 0, lo cual significa que para todas sus ejecuciones se obtuvo el mismo resultado, en el presente caso esto no fue así.

Ahora se muestran los resultados para el conjunto de grafos con 80 nodos.

Cuadro III: Resultados del algoritmo genético para el conjunto de grafos de 80 nodos y valores de $K = \{5, 10, 20\}$.

Instancia	K	Mejor resultado	Peor resultado	Media	Desviación estándar
tc80-1	5	1672	1679	1674	2.58
tc80-2	5	1660	1667	1663	2.6
tc80-3	5	1556	1563	1558	2.15
tc80-4	5	1663	1672	1667	3.13
tc80-5	5	1598	1607	1600	3.16
tc80-1	10	1279	1287	1282	3.15
tc80-2	10	1338	1345	1339	2.15
tc80-3	10	1285	1292	1286	2.13
tc80-4	10	1172	1187	1175	4.52
tc80-5	10	1156	1164	1158	2.91
tc80-1	20	1035	1043	1038	2.67
tc80-2	20	1087	1095	1090	2.8
tc80-3	20	1025	1034	1029	3.43
tc80-4	20	1023	1036	1026	4.18
tc80-5	20	1105	1113	1108	3.01

Cuadro IV: Resultados del artículo para el conjunto de grafos de 80 nodos y valores de $K = \{5, 10, 20\}$.

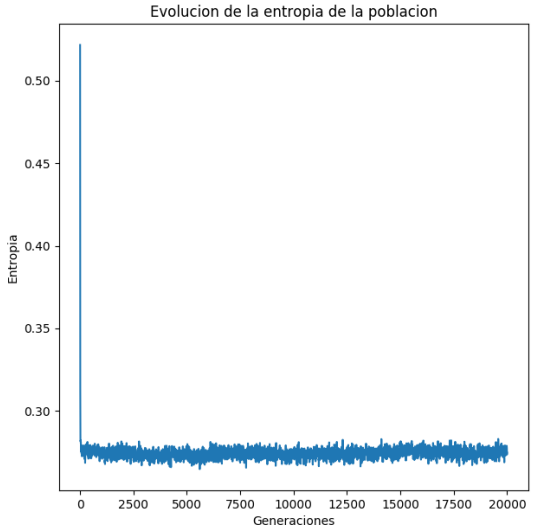
Instancia	K	Mejor resultado	Peor resultado	Media	Desviación estándar
tc80-1	5	1112	-	1124	8.97
tc80-2	5	1106	-	1111	2.54
tc80-3	5	1078	-	1089	5.05
tc80-4	5	1093	-	1100	5.8
tc80-5	5	1296	-	1306	6.68
tc80-1	10	896	-	1282	0.84
tc80-2	10	889	-	897	2.02
tc80-3	10	882	-	891	2.53
tc80-4	10	876	-	885	0
tc80-5	10	1029	-	876	2.54
tc80-1	20	838	-	1031	1.69
tc80-2	20	824	-	841	0.63
tc80-3	20	828	-	825	2.53
tc80-4	20	824	-	831	2.07
tc80-5	20	928	-	825	7.18

Para este conjunto de datos podemos notar que los resultados no son comparables ya que la diferencia de lo obtenido a lo reportado en el artículo es considerable, se cree que para este conjunto de datos se debió haber dado más tiempo al algoritmo para obtener mejores resultados, de la misma forma en la siguiente sección se hablará sobre un análisis de diversidad para verificar sobre los operadores implementados.

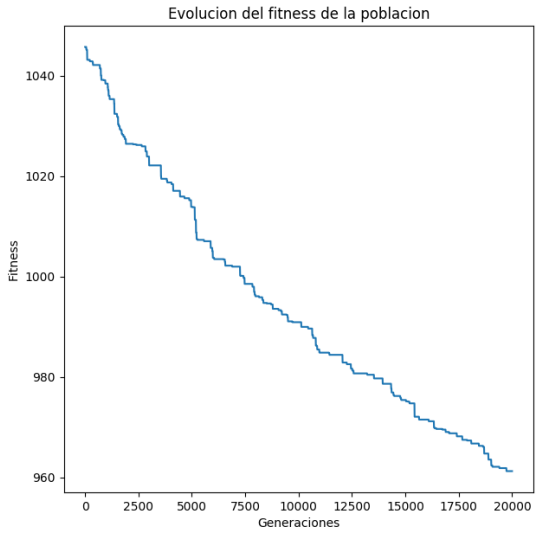
VI. Análisis de la diversidad

Debido a los resultados obtenidos anteriormente se propone un análisis de diversidad hacia la población, esto porque en el artículo se usa un algoritmo tipo *Steady state* el cual en la mayoría de los casos genera convergencia prematura, y se sospecha que esto es para contrarrestar efectos destructivos que tengan los operadores de cruza y mutación previamente implementados.
Lo que se hizo fue implementar la versión clásica del algoritmo genético y usar tanto los operadores de cruza y mutación como la heurística constructiva, posteriormente se hicieron 30 ejecucines de 20000 generaciones con un completo reemplazamiento en la población con la descendencia y una probabilidad de mutación de un 5 %. Cada 4 generaciones se guardaba la población en un archivo así como el mejor valor de aptitud de la misma, esto para poder generar una gráfica de evolución de la diversidad poblacional (entropía poblacional) con respecto a las generaciones. Este análisis no se realizó con el multi dyn ya que el algoritmo en si fomenta un manejo de diversidad alto y si en la gráfica aparecen niveles altos de diversidad no se sabrá con certeza si fue gracias a los operadores o al algoritmo.

A continuación se muestran las gráficas obtenidas con el algoritmo genético usando los operadores de cruza, mutación y heurística constructiva.

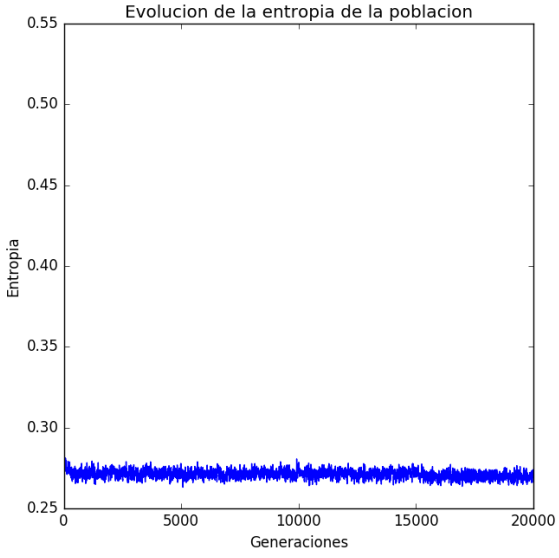


(b) Figura 2. Evolución de la diversidad poblacional del algoritmo genético.

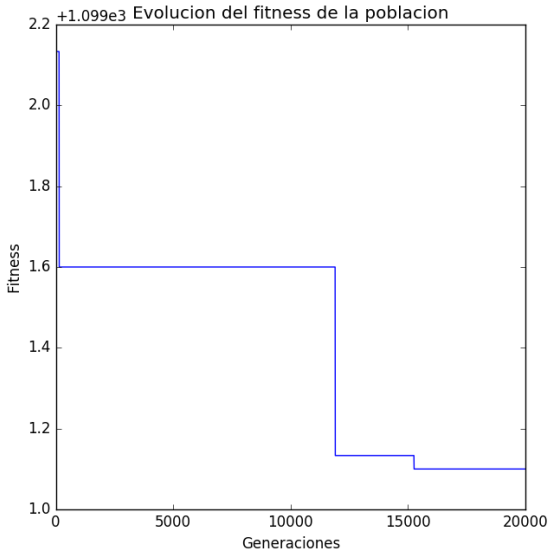


(c) Figura 3. Evolución del mejor valor de aptitud a través de las generaciones.

En la Figura 2 podemos ver que la evolución de la entropía promedio de las 30 ejecuciones indica que los operadores de cruza y mutación si son disruptivos a la población, esto porque desde generaciones tempranas empieza a oscilar y este comportamiento se mantiene constante durante las 20000 generaciones. De la misma forma se hizo otro experimento de 30 ejecuciones del algoritmo genético pero esta vez quitando la mutación, para probar que por sí sola, la cruza tiene este efecto disruptivo. A continuación se muestran los resultados de dicho experimento.



(d) Figura 4. Evolución de la diversidad poblacional del algoritmo genético.



(e) Figura 5. Evolución del mejor valor de aptitud a través de las generaciones.

En las Figuras anteriores se muestran los resultados de hacer el análisis pero sin el parámetro de mutación, podemos notar en la diversidad poblacional que esta se comporta de la misma manera que el caso anterior, aunque para la evolución de la aptitud se ve un cambio significativo en este experimento ya que por un gran número de generaciones este se mantuvo constante hasta que encierto punto pudo descender, esto anterior por efectos de la ausencia de la mutación.
Finalmente se puede decir que los operadores de cruza y mutación contienen efectos destructivos en la población.

VII. Conclusiones y trabajo futuro

En el presente reporte se mostró la implementación del algoritmo con estrategia de reemplazamiento $multi\ dyn$ así como el uso de los operadores de cruza, mutación y heurística constructiva propuestos en un artículo para poder tratar el problema del arbol de expansión mínima con capacidades.
Se pudo ver que en los experimentos para el conjunto de datos con grafos de 40 nodos se obtuvieron resultados muy buenos y parecidos a los del artículo, pero para conjuntos de grafos de 80 nodos no fue así ya que aunque las soluciones no eran tan malas si tenían diferencias significativas con respecto a las presentadas en el paper. Posteriormente se vió que los operadores de cruza y mutación contienen efectos disruptivos, esto aplicando un análisis de diversidad a la población y por lo mismo se concluye que la implementación con el multi dyn en conjunto con estos operadores no es factible ya que el algoritmo está pensado para operadores que aseguren convergencia en la población, lo cual no es el caso.
Otro punto importante es que para la presente implementación no se añadió una búsqueda local al algoritmo y esto puedo haber favorecido a la falta de convergencia que se tuvo.
Como trabajo futuro se propone modificar los operadores de cruza y mutación en el sentido de que lleguen a ser mas greedy y por lo mismo no sean tan disruptivos, de la misma forma añadir una búsqueda local o una metaheurística de trayectoria

a la implementación del genético con el multi dyn para obtener mejores resultados.

REFERENCIAS

[1] C. Segura, S. Peña, S. Botello and A. Hernandez. *The Importance of Diversity in the Application of Evolutionary Algorithms to the Sudoku Problem*, CIMAT A.C, 2016, IEEE.

[2] Raidl Gunther R, Drexel Christina. *A Predecessor Coding in an Evolutionary Algorithm for the Capacitated Minimum Spanning Tree Problem*, Favoritenstraße 9–11/1861, 1040 Vienna, Austria.

[3] Estéfane George Macedo de Lacerda, Manoel Firmino de Medeiros Junior. *A genetic algorithm for the capacitated minimum spanning tree problem*, 2006, IEEE Congress on Evolutionary Computation.

[4] Ruiz Efrain, Albareda-Sambola Maria, Fernández Elena, G.C. Resende Mauricio. *A biased random-key genetic algorithm for the capacitated minimum spanning tree problem*, Computers & Operations Research 57 (2015) 95–108, 2014.

[5] Torres-Jiménez José, Pinto-Elías Raúl, García-Romero Agustín. *A Simulated Annealing Approach for the Capacitated Minimum Spanning Tree Problem*, ITESM-Morelos, CENIDET.

[6] Kruskal's algorithm,
$https://en.wikipedia.org/wiki/Kruskal's_algorithm$

[7] Prims's algorithm,
$https://en.wikipedia.org/wiki/Prim'27s_algorithm$

[8] Esau-Williams algorithm,
$http://citeseerx.ist.psu.edu/viewdo/summary?doi=$ 10,1,1,4,346

[9] Union-Find algorithm,
$https://en.wikipedia.org/wiki/Disjoint-set_data_structure$