



Proyecto final
S-Labeling Problem (SLP)
Análisis y Diseño de Algoritmos

Doctor Eduardo Arturo Rodríguez Tello

Julio Arath Rosales Oliden	A01630738
Lizbeth Ortiz López	A00227346
Marlene Rodríguez Harms	A00227347
Reynaldo Vega Menchaca	A01114523

Viernes 15 de mayo del 2020

Índice

Introducción	3
Definición formal del problema	3
Descripción detallada y pseudocódigo	6
2.1 Algoritmo Evolutivo	6
2.1.1 Representación	6
2.2 Algoritmo genético	8
2.3 Algoritmo memético	9
2.3.1 Creación de la población inicial	9
2.3.1.1 Sumas mínimas	10
2.3.2 Torneo Binario	11
2.3.2.1 Cruza (Implementación genética)	12
2.3.2.2 Hijos Elegidos	13
2.3.2.3 Mutación	13
2.3.2.4 Búsqueda Local	13
2.3.2.5 Selección Superviviente	14
Análisis matemático de la complejidad temporal y espacial	16
3.1 Síntesis de complejidades	16
3.2 Complejidad final	17
3.3 Complejidad espacial	17
Experimentación con el algoritmo desarrollado	19
4.1 Grid Graph	20
4.2 Path Graphs	25
4.3 Cycle Graph	30
Situación práctica real	35
Aportaciones y participaciones	38
Retroalimentación	39
Conclusiones globales del proyecto	42
Referencias	43

Introducción

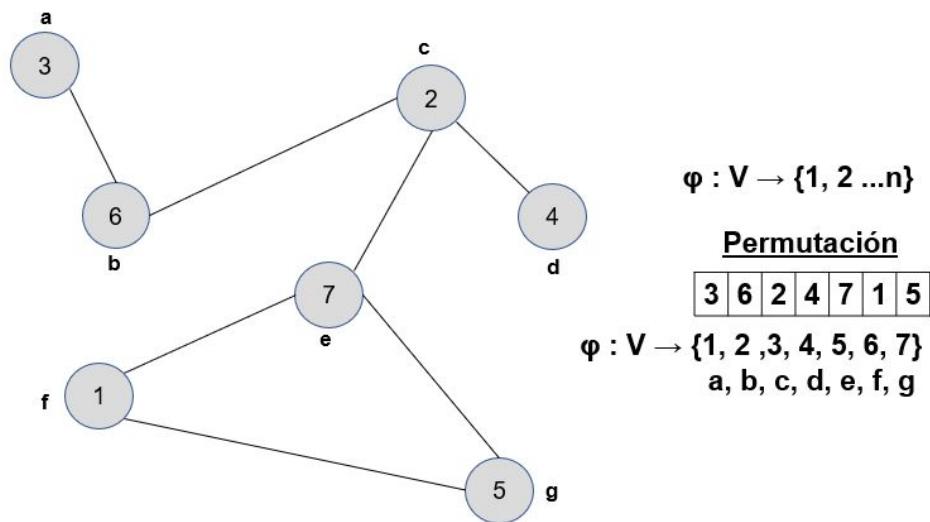
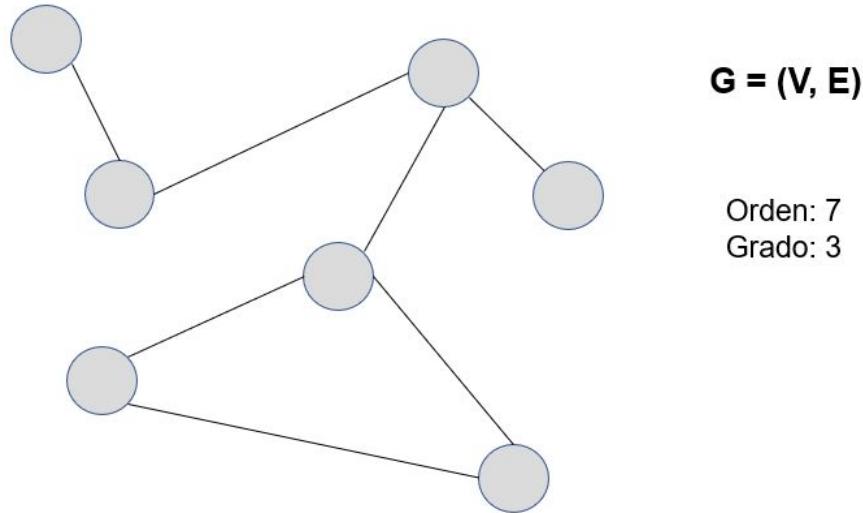
En este documento, se busca discutir detalles, descripción y una posible solución óptima para poder resolver el problema de S-Labeling mediante un algoritmo metaheurístico. Existen algoritmos exactos y metaheurísticos, ambos son secuencias de pasos para dar con la solución hacia un problema; sin embargo, tienen sus diferencias. Los exactos son todos aquellos algoritmos que dan con la respuesta con una eficiencia del 100%, es decir, dan con el resultado correcto para cada uno de sus casos, al contrario que los metaheurísticos, los cuales no dan con la solución en todos ellos, aunque son de gran utilidad para resolver problemas los cuales involucran entradas muy grandes. Un ejemplo podría ser un grafo que contenga más allá de un 1,000,000 de nodos; un algoritmo exacto tiene la desventaja de tener que comparar cada uno de los nodos y en comparación a los metaheurísticos, estos no comparan todos y solo toma una muestra.

1. Definición formal del problema

El problema de S-Labeling consiste en lo siguiente: dado un un grafo $G = (V, E)$, de orden n y grado máximo Δ , donde V es el número de nodos del grafo y E el número de aristas, encontrar un etiquetado que se define con un mapeo biyectivo

$\varphi : V \rightarrow \{1, 2, \dots, n\}$, tal que $SL\varphi(G) = \sum_{\{u, v\} \in E} \min\{\varphi(u), \varphi(v)\}$ sea minimizado.

Ejemplo para ilustrar el problema combinatorio analizado



	Aristas	Etiqueta	Mínimo
1	a, b	3, 6	3
2	b, c	6, 2	2
3	c, d	2, 4	2
4	c, e	2, 7	2
5	e, f	7, 1	1
6	e, g	7, 5	5
7	f, g	1, 5	1

Total = 16

$$SL\phi(G) = \sum_{i=0}^n \{u, v\} \in E \min\{\phi(u), \phi(v)\} = 16$$

2. Descripción detallada y pseudocódigo

Por la naturaleza del problema se determinó que la forma óptima para solucionarlo es a través de un algoritmo memético. Para el entendimiento de este, se deben comprender el término de algoritmia como de tipo evolutivo y genético.

2.1 Algoritmo Evolutivo

Los algoritmos evolutivos son un tipo de solución estocástica (no determinista) de aproximación de soluciones, los cuales operan sobre poblaciones realizando una búsqueda paralela implícita de las mejores soluciones. Sus operadores de variación (combinación y permutación) son el mecanismo que explora y explota los espacios. El mecanismo de selección está directamente relacionado con qué tan bien o mal se resuelve el problema con respecto a la calidad de la solución. En otras palabras, el conjunto de posibles soluciones, evolucionan a mejores soluciones.

2.1.1 Representación

Se deben representar las posibles soluciones en forma de *individuos*.

Población: Es un conjunto finito de *individuos*. Su cardinalidad está acotada (usualmente es de tamaño fijo). En este caso, se habla de las permutaciones de n nodos. La naturaleza del algoritmo es comenzar con una población aleatoria.

Función de evaluación (aptitud) de individuos: En optimización, la aptitud de cada individuo se relaciona con la función objetivo (función de utilidad). En este caso, se trata de encontrar $SL\phi(G) = \sum_{\{u, v\} \in E} \min\{\phi(u), \phi(v)\}$



Representación gráfica del algoritmo evolutivo

Selección de Padres

Cada individuo tiene asignado una cierta probabilidad de reproducción, dependiendo su aptitud. La selección es probabilística, es decir, las soluciones de mayor calidad tienen mayor probabilidad de reproducción.

Operación de Variación

Su razón de ser es generar nuevas soluciones a cada generación. Toman en cuenta el número de individuos involucrados (mutación, crusa, recombinación, etc). Para este algoritmo, la solución eficaz es mediante el uso de *cruza*.

Crusa

- Mezcla la información genética de los padres para reproducir descendientes.
- La decisión de qué información se toma de cada quien es estocástica.
- Se tiene la esperanza de que algunos de los descendientes sean mejores que sus padres.

1	3	5	2	6	4	7	8
8	7	6	5	4	3	2	1

1	3	5	4	2	8	7	6
8	7	6	2	4	1	3	5

Representación de crusa

Mutación

La mutación afecta a un individuo de la población. Se puede sustituir un nodo entero en el individuo seleccionado, o puede simplemente reemplazar la información del nodo. Para mantener la integridad, las operaciones deben ser salvo fallos o el tipo de información que el nodo tiene debe ser tomada en cuenta. Por ejemplo, la mutación debe ser consciente de nodos operación binaria, o el operador debe ser capaz de manejar los valores que faltan.

2.2 Algoritmo genético

Los algoritmos genéticos corresponden a una metaheurística de los algoritmos evolutivos. Es una de las metaheurísticas más utilizadas y funciona con todas las bases evolutivas que se explicaron con anterioridad. A continuación se presenta un ejemplo de algoritmo genético con operación de variación de cruce. Más adelante, en los algoritmos meméticos, se explica la finalidad de este algoritmo.

```
BEGIN /* Algoritmo Genetico Simple */
    Generar una poblacion inicial.
    Computar la funcion de evaluacion de cada individuo.
    WHILE NOT Terminado DO
        BEGIN /* Producir nueva generacion */
            FOR Tamaño poblacion/2 DO
                BEGIN /*Ciclo Reproductivo */
                    Seleccionar dos individuos de la anterior generacion,
                    para el cruce (probabilidad de seleccion proporcional
                    a la funcion de evaluacion del individuo).
                    Cruzar con cierta probabilidad los dos
                    individuos obteniendo dos descendientes.
                    Mutar los dos descendientes con cierta probabilidad.
                    Computar la funcion de evaluacion de los dos
                    descendientes mutados.
                    Insertar los dos descendientes mutados en la nueva generacion.
                END
            IF la poblacion ha convergido THEN
                Terminado := TRUE
        END
    END
```

Representación de un algoritmo genético simple (Moujahid, Inza, Larrañaga, s.f.)

2.3 Algoritmo memético

Es un algoritmo que combina un método poblacional (global) con un método local, el cual involucra información acerca del problema (funciones, restricciones, espacios de búsqueda). La idea es que cada técnica subsane las deficiencias de la otra. Combinan sinérgicamente conceptos tomados de otras metaheurísticas, tales como la búsqueda basada en poblaciones (como en los algoritmos evolutivos), y la mejora local (como en las técnicas de seguimiento del gradiente).

Implementación memética

Algorithm 4. A Basic Memetic Algorithm

```
1 function BasicMA (in  $P$ : Problem, in  $par$ : Parameters): Solution;
2 begin
3    $pop \leftarrow$  Initialize( $par, P$ );
4   repeat
5      $newpop_1 \leftarrow$  Cooperate( $pop, par, P$ );
6      $newpop_2 \leftarrow$  Improve( $newpop_1, par, P$ );
7      $pop \leftarrow$  Compete ( $pop, newpop_2$ );
8     if Converged( $pop$ ) then
9        $pop \leftarrow$  Restart( $pop, par$ );
10      endif
11    until TerminationCriterion( $par$ );
12    return GetNthBest( $pop, 1$ );
13 end
```

Ejemplo de un algoritmo memético básico simple (Neri, Moscato, Cotta, 2012)

2.3.1 Creación de la población inicial

Como observamos en el pseudocódigo anterior, el primer paso del algoritmo memético es la creación de población inicial. Las cualidades de una población son muy simples para este algoritmo: definimos poblaciones de 50 individuos cada una. A su vez, las cualidades de un individuo se definen por: *orden de etiquetación y suma mínima* para esa permutación, esto lo representamos a través de objetos de tipo **Individuo**.

Tomando esto en cuenta, podemos iniciar la primer población creando una primera permutación de individuos random. Esta población posteriormente después de crearse pasará a evaluarse a través de la función *Torneo Binario*.

```
//Generación de población
public static void poblacion(ArrayList<Integer> v) {
    for (Integer i = 0; i < cantIndividuos; i++) {
        Permutacion.Permutacion(v);
        population.add(new Individuo(sumMinIndividuo(v),v));
    }
    torneoBinario(population);
}
```

2.3.1.1 Sumas mínimas

```
public static Integer sumMinIndividuo(ArrayList<Integer> arrayList) {
    // Operaciones
    Integer minAcum = 0, etiqueta1 = 0, etiqueta2 = 0;
    for (int i = 0; i < array1.size(); i++) {
        etiqueta1 = arrayList.get(array1.get(i));
        etiqueta2 = arrayList.get(array2.get(i));

        if (etiqueta1 < etiqueta2) {
            minAcum += etiqueta1;
        } else {
            minAcum += etiqueta2;
        }
    }

    return (minAcum);
```

El algoritmo, recibe un *ArrayList* que contiene la permutación (al individuo), entra dentro de un *for loop*, el cual recorre nuestra estructura de datos para identificar las etiquetas que se van a comparar de la permutación. Posteriormente, las etiquetas mínimas son acumuladas dentro de un acumulador, para poder regresar después de la ejecución, la suma total de las etiquetas con base al grafo establecido en la estructura (Triplet Representation).

Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

2.3.2 Torneo Binario

El método torneo binario, selecciona 4 individuos de la población al azar, de los cuales se hacen dos torneos, donde competirán dos contra dos; posteriormente, los dos ganadores del torneo se *cruzaran* para tener dos hijos hasta tener una población nueva.

```
//Selección de cruce(Torneo Binario)
public static void torneoBinario(ArrayList<Individuo> individuo) {
    //Random
    Random random = new Random();
    //Torneo
    Individuo peleador1 = new Individuo();
    Individuo peleador2 = new Individuo();
    Individuo peleador3 = new Individuo();
    Individuo peleador4 = new Individuo();

    Individuo ganadorTorneo1 = new Individuo();
    Individuo ganadorTorneo2 = new Individuo();

    for(Integer i = 0; i < population.size(); i++) {
        //Primer torneo
        Integer tmp = random.nextInt(population.size());
        peleador1 = population.get(tmp);
```

```

        Integer tmp2 = random.nextInt(population.size());
        while(tmp == tmp2) {
            tmp2 = random.nextInt(population.size());
        }
        peleador2 = population.get(tmp2);

        if(peleador1.minimo < peleador2.minimo) {
            ganadorTorneo1 = peleador1;
        } else {
            ganadorTorneo1 = peleador2;
        }

        //Segundo torneo
        Integer tmp3 = random.nextInt(population.size());
        peleador3 = population.get(tmp3);

        Integer tmp4 = random.nextInt(population.size());
        while(tmp3 == tmp4) {
            tmp4 = random.nextInt(population.size());
        }
        peleador4 = population.get(tmp4);

        if(peleador3.minimo < peleador4.minimo) {
            ganadorTorneo2 = peleador3;
        } else {
            ganadorTorneo2 = peleador4;
        }

        //Cruza de campeones
        cruza(ganadorTorneo1, ganadorTorneo2);
    }
}

```

2.3.2.1 Cruza (Implementación genética)

El método cruza corresponde a la definición que se mencionó en apartados anteriores, utilizando las librerías incluidas en Opt4J. Se toma a dos individuos, los cuales provienen del torneo binario (sus ganadores), para cruzarlos y generar dos hijos. Los hijos serán añadidos a la población total de hijos.

```

//Cruza
public static void cruza(Individuo ganadorTorneo1, Individuo ganadorTorneo2) {
    //Hijos
    Pair<PermutationGenotype<?>> pair = Crossover.combinar(ganadorTorneo1, ganadorTorneo2);
    populationHijos.add(new Individuo(sumMinIndividuo((ArrayList<Integer>) pair.getFirst(), (ArrayList<Integer>) pair.getFirst())));
    populationHijos.add(new Individuo(sumMinIndividuo((ArrayList<Integer>) pair.getSecond(), (ArrayList<Integer>) pair.getSecond())));
}

```

2.3.2.2 Hijos Elegidos

La función hijosElegidos, selecciona al azar de la población total de hijos, el número de individuos igual al tamaño de la población original, obteniendo una población de hijos reducida.

```
//Hijo elegidos
public static void hijosElegidos() {
    //Randomizer
    ArrayList<Integer> random = new ArrayList<Integer>();
    for(Integer i = 0; i < populationHijos.size(); i++) {
        random.add(i);
    }
    Collections.shuffle(random);
    //-----
    //Elegidos
    for(int i = 0; i < population.size(); i++) {
        populationHijosElegidos.add(populationHijos.get(random.get(i)));
    }
}
```

2.3.2.3 Mutación

En mutación se recorre toda la población de supervivientes para mutar cada uno de los individuos, obteniendo de igual manera su permutación y etiquetado mínimo.

```
//Mutacion de los supervivientes
public static void Mutacion() {
    for(Integer i = 0; i < populationSuperviviente.size(); i++) {
        populationSuperviviente.set(i, MutatePermutatationSwap.mutate(populationSuperviviente.get(i)));
    }
}
```

2.3.2.4 Búsqueda Local

El propósito de la función búsqueda local es para buscar los vecinos de una permutación haciendo swaps y los guarda en caso de que disminuyera el valor de su etiqueta, al disminuir el costo de la etiqueta podemos repetirlo e incluso encontrar una etiqueta aún menor y así descubrimos nuevos posibles caminos para llegar a la solución menor.

```

//Búsqueda local
public static void busquedaLocal() {
    for(Integer k = 0; k < populationSuperviviente.size(); k++) {
        Individuo original = new Individuo();
        Individuo nuevo = new Individuo();
        original = populationSuperviviente.get(k);
        int i2=0;
        int j2=0;
        int nuevo1=0;
        for(Integer i = 0; i < original.list.size(); i++) {
            for(Integer j = i + 1; j < original.list.size(); j++) {
                swap(populationSuperviviente.get(k).i,j);
                nuevo1 = populationSuperviviente.get(k).minimo;
                i2=i;
                j2=j;
                swap(populationSuperviviente.get(k),i,j);

                if(original.minimo > nuevo1) {
                    swap(populationSuperviviente.get(k),i2,j2);
                }
            }
        }
    }
}

```

2.3.2.5 Selección Superviviente

En la selección final se hace una unión de la población original y de la población superviviente la cual ya pasó por una etapa de torneo binario, cruza, selección de supervivientes, mutación y una búsqueda local, para posteriormente sacar el individuo con el etiquetado más pequeño. Después se genera la población de la siguiente generación, la cual está compuesta por los mejores individuos de la población total a excepción de 5 individuos, los cuales serán tomados de la población total de manera aleatoria.

```
//Merge de población original con la mejorada
public static void seleccionFinal() {
    //Merge de P y P'
    populationTotal.addAll(population);
    populationTotal.addAll(populationSuperviviente);
    Collections.sort(populationTotal);

    //El mejor de toda la población
    minImp = populationTotal.get(0).minimo;
    //Siguiente generación (45 mejores)
    for(Integer i = 0; i < population.size() - 5; i++) {
        populationNext.add(populationTotal.get(i));
    }

    //Randomizer
    ArrayList<Integer> random = new ArrayList<Integer>();
    for(Integer i = population.size() - 5; i < populationTotal.size(); i++) {
        random.add(i);
    }
    Collections.shuffle(random);
    //-----
    //Diversidad en la población
    for(Integer i = 0; i < 5; i++) {
        populationNext.add(populationTotal.get(random.get(i)));
    }
}
```

3. Análisis matemático de la complejidad temporal y espacial

3.1 Síntesis de complejidades

A continuación se describen las complejidades temporales de los distintos métodos implementados a lo largo del algoritmo

Población

$$\sum_{i=0}^{n-1} 1 = n - 1 - 0 + 1 = n$$

Torneo binario

$$\sum_{i=0}^{n-1} 1 = n - 1 - 0 + 1 = n$$

Cruza

$$o(n)$$

Mutación

$$\sum_{i=0}^{n-1} 1 = n - 1 - 0 + 1 = n$$

Búsqueda local

$$\begin{aligned} \sum_{k=0}^{n-1} * \sum_{i=0}^{o-1} \sum_{j=i+1}^{o-1} 1 &= \sum_{k=0}^{n-1} * \sum_{i=0}^{o-1} [o-1] - i + 1 + 1 = \sum_{k=0}^{n-1} * \sum_{i=0}^{o-1} [o-1 - i] = \\ &\quad \sum_{k=0}^{n-1} \left(\sum_{i=0}^{o-1} (o-1) - \sum_{i=0}^{o-1} i \right) = \\ \sum_{k=0}^{n-1} [(o-1) \sum_{i=0}^{o-1} 1 - \sum_{i=0}^{o-1} i] &= \sum_{k=0}^{n-1} [(o-1) * (o-1 - 0 + 1) + \frac{(o-1)(o-1+1)}{2}] = \\ &\quad \sum_{k=0}^{n-1} (o-1) * (o) + \frac{(o-1)(o)}{2} = \sum_{k=0}^{n-1} \frac{3o^2}{2} - \frac{3o}{2} = \\ \sum_{k=0}^{n-1} \frac{3o^2}{2} - \frac{3o}{2} &= \frac{3o^2}{2} - \frac{3o}{2} \sum_{k=0}^{n-1} 1 = n - 1 - 0 + 1 = n * \left(\frac{3o^2}{2} - \frac{3o}{2} \right) \end{aligned}$$

Selección final

Merge análisis algorítmico

$$\sum_{i=0}^{2n-1} \log(n) = \log(n) \sum_{i=0}^{2n-1} n = \log(n)(2n - 1 - 0 + 1) = 2n * \log(n)$$

Siguiente generación + randomizer

$$\sum_{i=0}^{n-6} 1 + \sum_{i=n-5}^{n-1} 1 = (n - 6 - 0 + 1) + (n - 1 - (n - 5) + 1) = n - 5 + 5 = n$$

Diversidad de la población

$$\sum_{i=0}^4 1 = 4 - 0 + 1 = 5$$

3.2 Complejidad final

En el peor de los casos es:

$$3n + n * \left(\frac{3o^2}{2} - \frac{3o}{2} \right) + 2n * \log(n) + n + 5 = n * \left(\frac{3o^2}{2} - \frac{3o}{2} \right) + 2n * \log(n) + 4n + 5 = \\ n * \left(\frac{3o^2}{2} - \frac{3o}{2} \right) + 2n * \log(n) + 4n + 5 \in \theta(n^3)$$

3.3 Complejidad espacial

La complejidad espacial se conforma a partir de 5 *arraylists*, los cuales están compuestos por las siguiente complejidades:

- Población original: $O(n)$, se define el tamaño de la población.
- Población hijos: $O(2n)$, se define por el tamaño de la población pero por dos, ya que de cada dos individuos de la población se originan dos hijos nuevos que se añaden a esta población.
- Población superviviente: $O(n)$, se define como el tamaño de la población general, ya que en él se guardan la mitad de los hijos.

- Población total: $O(2n)$, se define al hacer un merge de la población original con la población superviviente la cual ya pasó por un torneo binario, cruza, mutación, etc.
- Población siguiente: $O(n)$, se define por el mismo tamaño de la población original, ya con los mejores individuos de la población total, más unos 5 para que haya diversidad.

$$\sum_{i=0}^{n-1} 7 = 7 \sum_{i=0}^{n-1} 1 = 7(n - 1 - 0 + 1) = 7n$$

$$\sum_{i=0}^{n-1} 7o = 7o \sum_{i=0}^{n-1} 1 = 7o(n - 1 - 0 + 1) = 7on$$

4. Experimentación con el algoritmo desarrollado

Se crearon 15 instancias (de tres grafos distintos) y cada uno fue ejecutado 50 veces para verificar sus resultados. Para cada tipo de grafo (Grid, Path y Cycle) se utilizaron fórmulas matemáticas o experimentaciones pasadas obtenidas del artículo “*Algorithmic expedients for the S-labeling problem*” (Sinnl, 2019). El respaldo de los experimentos realizados se encuentra dentro de la carpeta **Respaldo SLP.zip**,

A continuación, se presentan los resultados tentativos según el tipo de grafo, representados por el nombre de experimento, seguido por el número de nodos que contiene el grafo y finalmente arrojando el resultado *SLP* esperado:

GRID GRAPH			PATH GRAPH		
Name	Nodes	SLP	Name	Nodes	SLP
test01	16	96	test06	10	30
test02	25	242	test07	25	168.75
test03	36	514	test08	50	650
test04	49	972	test09	75	1443.75
test05	64	1692	test10	100	2550

CYCLE GRAPH		
Name	Nodes	SLP
test11	45	528.75
test12	65	1088.75
test13	80	1640
test14	95	2303.75
test15	110	3080

Todos los resultados obtenidos en “*Tiempo de ejecución*”(Time execution) están representados en **milisegundos**.

4.1 Grid Graph

test01

GRID GRAPH						GRID GRAPH			
Name: test01			Nodes: 16			Name: test01		Nodes: 16	
Expecting SLP Result: 96									
No. Execution	Result	Time Exe.	No. Execution	Result	Time Exe.	Best SLP	96	Worst SLP	96
1	96	74	26	96	72	Range	0	Mean (SLP)	96
2	96	77	27	96	73	Standard Deviation	0	Succes Rate	50 (100%)
3	96	74	28	96	79				
4	96	79	29	96	75				
5	96	74	30	96	83				
6	96	75	31	96	68	Best Time	63		
7	96	83	32	96	75	Worst Time	93		
8	96	84	33	96	79	Range	30		
9	96	87	34	96	74	Mean (Time)	76.66		
10	96	83	35	96	76	Standard Deviation	5.542452121		
11	96	81	36	96	85				
12	96	80	37	96	72				
13	96	90	38	96	69				
14	96	78	39	96	71				
15	96	93	40	96	76				
16	96	78	41	96	69				
17	96	77	42	96	76				
18	96	74	43	96	79				
19	96	63	44	96	81				
20	96	74	45	96	80				
21	96	74	46	96	77				
22	96	72	47	96	75				
23	96	76	48	96	72				
24	96	73	49	96	71				
25	96	78	50	96	75				

En la siguiente tabla se puede apreciar que el resultado fue el óptimo todas la veces que se corrió, por lo que el algoritmo funcionó de manera óptima en todas la veces que se ejecuta este ejemplo de 16 nodos. Sin embargo los tiempo de ejecución sí variaron dando un promedio de 76.66 milisegundos, donde nuestro peor tiempo sería 93 milisegundos y en el mejor de los casos 63 milisegundos.

test02

GRID GRAPH						GRID GRAPH					
Name: test02			Nodes: 25			Name: test02			Nodes: 16		
Expecting SLP Result: 242											
No. Execution	Result	Time Exe.	No. Execution	Result	Time Exe.	Best SLP	242	Worst SLP	244	Range	2
1	242	185	26	242	197	Mean (SLP)	242.16	Standard Deviation	0.509501557	Succes Rate	45 (90%)
2	242	230	27	242	247	Best Time	175	Worst Time	334	Range	159
3	242	227	28	242	272	Mean (Time)	228.8	Standard Deviation	43.37355538	Succes Rate	45 (90%)
4	242	308	29	242	184	Best Time	175	Worst Time	334	Range	159
5	244	191	30	242	266	Mean (Time)	228.8	Standard Deviation	43.37355538	Succes Rate	45 (90%)
6	242	239	31	242	175	Best Time	175	Worst Time	334	Range	159
7	242	199	32	242	206	Mean (Time)	228.8	Standard Deviation	43.37355538	Succes Rate	45 (90%)
8	242	222	33	242	216	Best Time	175	Worst Time	334	Range	159
9	242	265	34	242	285	Mean (Time)	228.8	Standard Deviation	43.37355538	Succes Rate	45 (90%)
10	242	224	35	242	265	Best Time	175	Worst Time	334	Range	159
11	243	246	36	242	279	Mean (Time)	228.8	Standard Deviation	43.37355538	Succes Rate	45 (90%)
12	242	181	37	242	334	Best Time	175	Worst Time	334	Range	159
13	242	184	38	244	175	Mean (Time)	228.8	Standard Deviation	43.37355538	Succes Rate	45 (90%)
14	242	194	39	242	179	Best Time	175	Worst Time	334	Range	159
15	242	240	40	242	277	Mean (Time)	228.8	Standard Deviation	43.37355538	Succes Rate	45 (90%)
16	242	261	41	242	252	Best Time	175	Worst Time	334	Range	159
17	244	214	42	242	307	Mean (Time)	228.8	Standard Deviation	43.37355538	Succes Rate	45 (90%)
18	242	176	43	242	288	Best Time	175	Worst Time	334	Range	159
19	242	178	44	242	210	Mean (Time)	228.8	Standard Deviation	43.37355538	Succes Rate	45 (90%)
20	242	303	45	242	211	Best Time	175	Worst Time	334	Range	159
21	242	188	46	242	184	Mean (Time)	228.8	Standard Deviation	43.37355538	Succes Rate	45 (90%)
22	242	253	47	242	182	Best Time	175	Worst Time	334	Range	159
23	243	176	48	242	281	Mean (Time)	228.8	Standard Deviation	43.37355538	Succes Rate	45 (90%)
24	242	253	49	242	209	Best Time	175	Worst Time	334	Range	159
25	242	241	50	242	181	Mean (Time)	228.8	Standard Deviation	43.37355538	Succes Rate	45 (90%)

Para un grafo grid con 25 nodos los resultados de casi todas las ejecuciones son los esperados, a excepción tres que tienen un resultado mayor al esperado, sin embargo no se encuentran muy alejados del óptimo, solamente dos unidades. En cuanto al tiempo existen resultados muy variados, dando un promedio de 228.8 milisegundos, donde en el peor de los casos el tiempo sería de 334 y el mejor 175.

test03

GRID GRAPH						GRID GRAPH							
Name: test03		Nodes: 36		Name: test03		Nodes: 36							
Expecting SLP Result: 514													
No. Execution Result Time Exe. No. Execution Result Time Exe.													
1	514	831	26	514	655	Best SLP	514						
2	514	573	27	514	938	Worst SLP	516						
3	514	792	28	514	882	Range	2						
4	514	692	29	514	873	Mean (SLP)	514.14						
5	514	691	30	515	659	Standard Deviation	0.404565779						
6	515	925	31	514	1001	Succes Rate	44 (88%)						
7	514	745	32	515	907								
8	516	664	33	514	767	Best Time	573						
9	514	950	34	514	697	Worst Time	1269						
10	514	1079	35	514	982	Range	696						
11	514	1021	36	514	722	Mean (Time)	836.98						
12	514	583	37	515	670	Standard Deviation	163.6054849						
13	514	1176	38	514	1001								
14	514	945	39	514	1066								
15	514	975	40	514	593								
16	514	817	41	514	639								
17	514	777	42	514	689								
18	514	760	43	514	848								
19	514	731	44	514	900								
20	514	1040	45	514	995								
21	514	944	46	514	942								
22	514	609	47	514	943								
23	514	777	48	514	842								
24	515	681	49	514	944								
25	514	647	50	514	1269								

Para esta muestra de 36 nodos los resultados son más variados, se encuentran 6 ejecuciones donde el resultado no fue en óptimo, pero sí muy cercano a serlo, ya que solo sobrepasa a lo mucho dos unidades, por lo tanto podemos concluir que a pesar de no ser lo esperado en todas las ejecuciones sigue siendo bastante óptimo. Sobre el tiempo de ejecución hay un rango de 696 milisegundos entre el mejor caso que es 573 ms y el peor 1269 ms, una cifra más grande que en las instancias anteriores, pero todavía bastante chica.

En esta instancia es más notorio como el algoritmo al tener entradas más grandes va bajando un poco su precisión y además hay un aumento en tiempo en las ejecuciones.

test04

GRID GRAPH						GRID GRAPH					
Name: test04			Nodes: 49			Name: test04			Nodes: 49		
Expecting SLP Result: 972											
No. Execution	Result	Time Exe.	No. Execution	Result	Time Exe.	Best SLP	Worst SLP	Range	Mean (SLP)	Standard Deviation	Succes Rate
1	972	2789	26	972	2772	972	978	6	973.42	1.928201023	26 (52%)
2	973	2481	27	978	1702						
3	974	2507	28	976	1894						
4	972	3060	29	973	2312						
5	974	3033	30	972	1459						
6	972	2663	31	972	3026	Best Time	1459				
7	973	2718	32	978	2273	Worst Time	3839				
8	972	2067	33	974	2058	Range	2380				
9	975	1891	34	972	2426	Mean (Time)	2393.54				
10	975	2039	35	972	2813	Standard Deviation	506.1529659				
11	976	1954	36	974	2398						
12	972	2744	37	977	2221						
13	972	2428	38	974	2067						
14	973	2264	39	978	1651						
15	978	1521	40	975	1672						
16	972	2238	41	972	2273						
17	972	3839	42	972	2929						
18	972	2436	43	972	2336						
19	973	2743	44	973	2387						
20	972	2960	45	972	2069						
21	972	2508	46	972	3368						
22	974	2670	47	972	2933						
23	972	3124	48	972	1533						
24	972	2236	49	972	2097						
25	976	1750	50	975	2345						

En el caso de una entrada de 49 nodos es donde el algoritmo comienza a ser menos óptimo, ya que varias de las ejecuciones exceden el resultado esperado, dando en el peor de los casos 8 unidades más de lo esperado. Los resultado siguen siendo muy buenos, sin embargo ya hay un exceso notorio con respecto al resultado esperado. El mismo caso ocurre con el tiempo tomado en la ejecución, donde ha crecido el margen de milisegundos entre la mejor ejecución y la peor.

test05

GRID GRAPH						GRID GRAPH	
Name: test05 Nodes: 64			Expecting SLP Result: 1692				
No. Execution	Result	Time Exe.	No. Execution	Result	Time Exe.		
1	1694	4456	26	1692	4053		
2	1692	4962	27	1692	5904		
3	1695	6073	28	1692	5835		
4	1692	5199	29	1693	5279		
5	1696	4225	30	1692	5724		
6	1692	6561	31	1693	4483		
7	1695	4731	32	1693	5265		
8	1692	5575	33	1692	4788		
9	1692	7257	34	1692	5721		
10	1698	7887	35	1692	5302		
11	1693	5779	36	1692	5507		
12	1699	4419	37	1701	4627		
13	1698	3013	38	1692	4586		
14	1709	3104	39	1692	5090		
15	1692	6780	40	1696	4225		
16	1692	4274	41	1692	4149		
17	1698	3185	42	1692	5479		
18	1699	3715	43	1692	6435		
19	1694	4310	44	1692	6384		
20	1704	3442	45	1696	6834		
21	1698	4392	46	1692	4501		
22	1710	3190	47	1693	6374		
23	1695	5162	48	1692	6325		
24	1694	5292	49	1704	3475		
25	1701	3928	50	1702	3331		

Y para la última instancia con una entrada de 64 nodos, notoriamente el rango entre el resultado más óptimo y el peor ha crecido, exactamente en 18 unidades, donde las ejecuciones ya no son del todo óptimas y se puede comprobar con desviación estándar que es de 4.52 unidades. Sin duda el tiempo de ejecución ha variado bastante, con un mínimo de 3013 ms hasta un máximo de 7887 ms, donde el promedio es de 5011.4 .

4.2 Path Graphs

Los path graphs son un caso especial en nuestro algoritmo, ya que da un resultado por debajo de la predicción que encontramos en el artículo “*Algorithmic expedients for the S-labeling problem*” (Sinnl, 2019) con la fórmula siguiente:

$$SL = \frac{n^2}{4} + \frac{n}{2}$$

Donde n es el número de nodos.

test06

PATH GRAPH						PATH GRAPH					
Name: test06			Nodes: 10			Name: test06			Nodes: 10		
Expecting SLP Result: 30											
No. Execution	Result	Time Exe.	No. Execution	Result	Time Exe.	Best SLP	25	Worst SLP	25	Range	0
1	25	49	26	25	47	Mean (SLP)	25	Standard Deviation	0	Mean (Time)	49.7
2	25	54	27	25	48	Succes Rate	50 (100%)	Best Time	39	Standard Deviation	11.611483
3	25	48	28	25	42	Worst Time	125	Range	86	Mean (Time)	49.7
4	25	47	29	25	42	Range	86	Standard Deviation	11.611483	Best Time	39
5	25	44	30	25	48	Worst Time	125	Mean (Time)	49.7	Worst Time	125
6	25	52	31	25	125	Range	86	Standard Deviation	11.611483	Range	86
7	25	46	32	25	45	Mean (Time)	49.7	Succes Rate	50 (100%)	Mean (Time)	49.7
8	25	41	33	25	50	Standard Deviation	11.611483	Best Time	39	Standard Deviation	11.611483
9	25	51	34	25	48	Worst Time	125	Range	86	Worst Time	125
10	25	58	35	25	55	Range	86	Mean (Time)	49.7	Range	86
11	25	52	36	25	46	Mean (Time)	49.7	Standard Deviation	11.611483	Mean (Time)	49.7
12	25	48	37	25	50	Standard Deviation	11.611483	Succes Rate	50 (100%)	Standard Deviation	11.611483
13	25	52	38	25	52	Succes Rate	50 (100%)	Best Time	39	Succes Rate	50 (100%)
14	25	49	39	25	45	Best Time	39	Worst Time	125	Best Time	39
15	25	54	40	25	48	Worst Time	125	Range	86	Worst Time	125
16	25	48	41	25	44	Range	86	Mean (Time)	49.7	Range	86
17	25	47	42	25	45	Mean (Time)	49.7	Standard Deviation	11.611483	Mean (Time)	49.7
18	25	48	43	25	49	Standard Deviation	11.611483	Succes Rate	50 (100%)	Standard Deviation	11.611483
19	25	49	44	25	47	Succes Rate	50 (100%)	Best Time	39	Succes Rate	50 (100%)
20	25	52	45	25	49	Best Time	39	Worst Time	125	Best Time	39
21	25	50	46	25	51	Worst Time	125	Range	86	Worst Time	125
22	25	49	47	25	41	Range	86	Mean (Time)	49.7	Range	86
23	25	42	48	25	47	Mean (Time)	49.7	Standard Deviation	11.611483	Mean (Time)	49.7
24	25	46	49	25	39	Standard Deviation	11.611483	Succes Rate	50 (100%)	Standard Deviation	11.611483
25	25	59	50	25	47	Succes Rate	50 (100%)	Best Time	39	Succes Rate	50 (100%)

Para la primera instancia todas las ejecuciones dan un resultado por debajo de la predicción y este es 25, 5 unidades por debajo de lo esperado como resultado. Y el tiempo de ejecución está dentro de un rango de 39 ms a 125 ms. Esta instancia es una de las más óptimas porque en cada una de sus ejecuciones logró dar un resultado no solo , sino mejor de lo esperado, con un tiempo de ejecución muy bajo.

test07

PATH GRAPH						PATH GRAPH	
Name: test07			Nodes: 25			Name: test07	
Expecting SLP Result: 169						Nodes: 25	
No. Execution	Result	Time Exe.	No. Execution	Result	Time Exe.	Best SLP	156
1	156	169	26	156	148	Worst SLP	156
2	156	151	27	156	163	Range	0
3	156	152	28	156	159	Mean (SLP)	156
4	156	137	29	156	170	Standard Deviation	0
5	156	162	30	156	153	Succes Rate	50 (100%)
6	156	165	31	156	201	Best Time	135
7	156	174	32	156	145	Worst Time	207
8	156	153	33	156	172	Range	72
9	156	179	34	156	158	Mean (Time)	163.46
10	156	165	35	156	169	Standard Deviation	15.37053901
11	156	158	36	156	182		
12	156	168	37	156	207		
13	156	174	38	156	140		
14	156	150	39	156	179		
15	156	156	40	156	180		
16	156	156	41	156	150		
17	156	174	42	156	176		
18	156	160	43	156	159		
19	156	160	44	156	163		
20	156	165	45	156	190		
21	156	135	46	156	170		
22	156	151	47	156	182		
23	156	141	48	156	156		
24	156	151	49	156	166		
25	156	145	50	156	184		

En esta instancia de 25 nodos, ocurre lo mismo, todas las ejecuciones dan un valor menor al esperado, por lo tanto el algoritmo está funcionando de manera optimizada. Su tiempo de ejecución tiene un promedio de 163.46 milisegundos, donde en el peor de los casos tardaría 207 ms y en el mejor de ellos 135 ms.

test08

PATH GRAPH					
Name: test08			Nodes: 50		
Expecting SLP Result: 650					
No. Execution	Result	Time Exe.	No. Execution	Result	Time Exe.
1	627	944	26	626	1421
2	625	1534	27	627	956
3	628	823	28	625	1464
4	627	783	29	627	981
5	625	1565	30	627	1044
6	625	997	31	625	1004
7	625	1254	32	627	1281
8	627	1137	33	626	1591
9	626	1070	34	625	1166
10	627	1149	35	627	1298
11	626	1311	36	626	1383
12	625	1316	37	625	830
13	626	1181	38	626	933
14	626	992	39	627	937
15	627	927	40	626	1080
16	625	1072	41	627	1055
17	626	862	42	628	828
18	627	1207	43	627	1412
19	626	1292	44	625	989
20	627	940	45	625	1127
21	625	1347	46	628	996
22	625	1551	47	625	1055
23	626	964	48	626	1116
24	626	1135	49	626	1446
25	627	949	50	627	1068

PATH GRAPH					
Name: test08			Nodes: 50		
Best SLP			625		
Worst SLP			628		
Range			3		
Mean (SLP)			626.16		
Standard Deviation			0.933722		
Succes Rate			50 (100%)		
Best Time			783		
Worst Time			1591		
Range			808		
Mean (Time)			1135.26		
Standard Deviation			216.12759		

Para una instancia de 50 nodos (ha crecido bastante la entrada), aunque no todas las ejecuciones logren dar lo mínimo que es 625, es un hecho que todas resultan en valores muy por debajo de la predicción de 650, ya que a lo mucho da 628. Efectivamente el tiempo de ejecución ha aumentado con respecto a las instancias anteriores debido al tamaño de la entrada, dando un promedio de 1135.26 ms.

test09

PATH GRAPH						PATH GRAPH	
Name: test09			Nodes: 75			Name: test09	Nodes: 75
Expecting SLP Result: 1444						Best SLP	1406
No. Execution	Result	Time Exe.	No. Execution	Result	Time Exe.	Worst SLP	1415
1	1410	3342	26	1410	3900	Range	9
2	1410	4646	27	1410	4047	Mean (SLP)	1409.98
3	1407	4707	28	1410	3241	Standard Deviation	1.696214272
4	1414	3528	29	1410	4159	Succes Rate	50 (100%)
5	1411	3661	30	1407	5441	Best Time	2782
6	1409	4151	31	1410	3247	Worst Time	6272
7	1410	3479	32	1410	3205	Range	3490
8	1411	4758	33	1411	3530	Mean (Time)	4116.14
9	1409	6272	34	1412	4117	Standard Deviation	737.6356329
10	1410	4900	35	1409	5052		
11	1410	4468	36	1409	3343		
12	1408	4958	37	1410	2782		
13	1411	4439	38	1410	3876		
14	1410	3285	39	1409	4042		
15	1407	5031	40	1410	4591		
16	1410	4174	41	1410	4797		
17	1410	3927	42	1410	4426		
18	1410	3178	43	1411	4504		
19	1411	4246	44	1410	3926		
20	1412	3471	45	1409	4563		
21	1406	5826	46	1407	3362		
22	1410	3922	47	1411	4775		
23	1413	4188	48	1410	3400		
24	1409	3191	49	1413	3320		
25	1408	4184	50	1415	4229		

En 75 nodos de entrada el algoritmo se mantiene con resultados variados entre sus ejecuciones, sin embargo, todos muy por debajo del esperado de 1444, en la mejor ejecución da un resultado de 1406, pero el tiempo que toma esta ejecución de 5826 ms es uno de los más altos, sobre pasando el promedio por aproximadamente dos desviaciones estándar. Sin embargo, el promedio de salida de esta instancia es 1409.98, demostrando resultados muy óptimos aún cuando la entrada es relativamente grande. En el tiempo de ejecución la brecha entre lo mínimo de valor 2782 ms y máximo de valor 6272 ha crecido a ser en este 3490 milisegundos, dando un promedio de 4116.14 ms, esto no nos sorprende debido a que de antemano sabíamos que la complejidad de los procesos, aumentaría crecientemente el tiempo de ejecución a medida que la entrada fuera más grande.

test10

PATH GRAPH						PATH GRAPH					
Name: test10 Nodes: 100			Expecting SLP Result: 2550			Name: test10 Nodes: 100					
No.	Execution	Result	Time Exe.	No.	Execution	Result	Time Exe.				
1	2512	6270		26	2508	10819		Best SLP	2503		
2	2508	7582		27	2512	12377		Worst SLP	2516		
3	2507	7646		28	2508	11032		Range	13		
4	2508	5943		29	2507	8627		Mean (SLP)	2508.58		
5	2508	14746		30	2510	8601		Standard Deviation	2.8577856		
6	2513	10735		31	2510	7517		Succes Rate	50 (100%)		
7	2512	7210		32	2515	5649					
8	2506	13203		33	2512	7781					
9	2509	6638		34	2507	8049					
10	2507	10874		35	2506	11891					
11	2510	8232		36	2503	6513					
12	2516	6089		37	2507	9633					
13	2506	10576		38	2510	9728					
14	2514	7136		39	2507	8729					
15	2508	9571		40	2506	9148					
16	2508	11074		41	2506	8986					
17	2508	6752		42	2504	8953					
18	2506	7332		43	2506	7822					
19	2512	7554		44	2512	9128					
20	2507	10819		45	2508	8206					
21	2508	12377		46	2508	10303					
22	2506	11032		47	2507	7632					
23	2508	8627		48	2510	11190					
24	2507	8601		49	2514	6258					
25	2506	7554		50	2506	6616					

Por último caso en los Path Graphs, tenemos una entrada de 100 nodos, donde la optimalidad del algoritmo sigue presenta, arrojando resultados de en promedio 2508.58, cuando el valor esperado es de 2550. El tiempo de ejecución a su vez sigue creciendo debido al tamaño de la entrada, dando un promedio de 8907.22 ms, manteniéndose en un rango de 5649 ms en el mejor de los casos y 14,746 ms en el peor.

4.3 Cycle Graph

En las siguientes instancias para Cycle Graph los valores esperados fueron dados por la fórmula siguiente encontrada en el artículo “*Algorithmic expedients for the S-labeling problem*” (Sinnl, 2019):

$$SL = \frac{(n+1)^2}{4}$$

Donde n es el número de nodos.

test11

CYCLE GRAPH						CYCLE GRAPH	
Name: test11			Nodes: 45				
Expecting SLP Result: 529							
No. Execution	Result	Time Exe.	No. Execution	Result	Time Exe.	Best SLP	529
1	529	1210	26	530	963	Worst SLP	531
2	529	730	27	529	945	Range	2
3	530	812	28	529	752	Mean (SLP)	529.48
4	531	790	29	530	863	Standard Deviation	0.677329691
5	529	1003	30	529	592	Succes Rate	31 (62%)
6	529	707	31	529	874	Best Time	592
7	530	600	32	529	726	Worst Time	1516
8	531	693	33	529	832	Range	924
9	529	730	34	529	946	Mean (Time)	852.9
10	530	821	35	531	753	Standard Deviation	197.6631075
11	529	940	36	531	729		
12	530	695	37	529	1501		
13	530	712	38	530	673		
14	529	720	39	530	906		
15	530	834	40	529	808		
16	529	853	41	529	917		
17	530	711	42	529	706		
18	529	980	43	529	967		
19	529	1053	44	529	1516		
20	529	689	45	529	1137		
21	529	1101	46	529	745		
22	530	612	47	530	864		
23	529	1073	48	530	921		
24	529	696	49	531	604		
25	529	777	50	529	863		

En la primera instancia para Cycle Graphs, comenzamos con una prueba de 45 nodos, en donde las ejecuciones son variadas, sin embargo todas muy cercanas al valor esperado que es

529, ya que en el peor de los casos el resultado se excede por dos unidades. En cuanto al tiempo de ejecución los tiempos son variados, su promedio es de 924 ms y en el peor de los casos le toma 1516 ms.

test12

CYCLE GRAPH					
Name: test12		Nodes: 65			
Expecting SLP Result: 1089					
No. Execution	Result	Time Exe.	No. Execution	Result	Time Exe.
1	1092	3197	26	1091	3640
2	1093	1940	27	1094	2178
3	1091	4738	28	1091	3425
4	1089	4287	29	1091	2253
5	1093	2479	30	1089	3420
6	1091	3505	31	1091	2557
7	1092	2196	32	1090	2223
8	1091	3209	33	1093	2598
9	1091	2426	34	1091	3389
10	1093	2225	35	1089	4496
11	1090	1859	36	1090	3142
12	1091	2635	37	1092	2815
13	1095	2109	38	1092	2068
14	1091	3661	39	1092	2534
15	1091	2711	40	1093	3350
16	1091	2381	41	1091	2436
17	1094	1769	42	1091	2368
18	1091	3973	43	1091	2843
19	1092	3430	44	1091	2448
20	1089	3109	45	1091	2535
21	1093	2610	46	1091	2568
22	1093	1984	47	1091	2086
23	1090	2832	48	1091	2419
24	1091	4061	49	1092	2736
25	1091	4248	50	1091	1091

CYCLE GRAPH					
Name: test12		Nodes: 65			
Best SLP		1089			
Worst SLP		1095			
Range		6			
Mean (SLP)		1091.38			
Standard Deviation		1.29188993			
Succes Rate		4 (8%)			
Best Time		1091			
Worst Time		4738			
Range		3647			
Mean (Time)		2823.84			
Standard Deviation		770.2676015			

En la segunda instancia para Cycle Graph la entrada es de 65 nodos. Podemos observar que las ejecuciones van perdiendo precisión, pero se mantienen aún muy cerca del resultado esperado que es 1089, en el peor de los casos de ejecución este que rebasa por 6 unidades al valor esperado, lo cual es muy poco, por lo que podemos decir que el algoritmo sigue comportándose bastante bien. El tiempo de ejecución tiene un promedio de 2823.84 ms, debido al crecimiento de la entrada y en el mejor de los casos podría llegar a ejecutarse en 1091 ms.

test13

CYCLE GRAPH						CYCLE GRAPH					
Name: test13			Nodes: 80			Name: test13			Nodes: 80		
Expecting SLP Result: 1640											
No.	Execution	Result	Time	Exe.	No.	Execution	Result	Time	Exe.	Best SLP	1641
1	1641	6071			26	1644	5756			Worst SLP	1649
2	1646	5845			27	1645	4269			Range	8
3	1642	5388			28	1641	5670			Mean (SLP)	1644.44
4	1646	4346			29	1643	4335			Standard Deviation	2.011801913
5	1644	5241			30	1644	5578			Succes Rate	0
6	1645	5329			31	1649	3405			Best Time	3338
7	1644	5967			32	1644	5970			Worst Time	7385
8	1644	5018			33	1643	7024			Range	4047
9	1648	4654			34	1643	5421			Mean (Time)	5139.14
10	1645	5092			35	1646	3872			Standard Deviation	918.9957032
11	1641	5509			36	1642	6122				
12	1646	3630			37	1644	4383				
13	1645	4808			38	1644	6780				
14	1646	4159			39	1645	4742				
15	1644	5005			40	1644	3338				
16	1644	5155			41	1645	5593				
17	1645	5779			42	1645	5455				
18	1643	4677			43	1649	4578				
19	1643	4936			44	1644	4502				
20	1645	4692			45	1641	6432				
21	1644	5267			46	1644	5462				
22	1645	4667			47	1644	6463				
23	1641	7385			48	1649	4152				
24	1644	5256			49	1644	6058				
25	1646	3538			50	1649	4183				

Para nuestra tercera instancia ninguna llega al valor esperado que es de 1640, sin embargo el mínimo es de 1641, sobrepasando únicamente 1 unidad al valor esperado. En este caso las ejecuciones dan resultados variados, pero todos menores a 1649. En cuanto al tiempo de ejecución sucede lo mismo que con los otros grafos, al ser procesos muy pesados, este tiende a crecer bastante cada vez que la entrada crece. En este caso el tiempo para ejecución promedio es de 5139.14 ms.

test14

CYCLE GRAPH					
Name: test14		Nodes: 95			
Expecting SLP Result: 2304					
No.	Execution	Result	Time Exec.	No.	Execution
1	2309	9771	26	2314	5937
2	2308	8528	27	2308	8066
3	2312	9341	28	2312	6789
4	2315	6844	29	2312	7267
5	2310	7910	30	2306	11710
6	2315	6691	31	2311	5802
7	2314	5593	32	2309	10319
8	2311	8530	33	2314	9042
9	2309	6306	34	2311	6929
10	2310	8574	35	2310	8412
11	2312	6400	36	2315	5806
12	2310	6344	37	2317	5802
13	2310	11007	38	2307	10319
14	2310	10016	39	2315	9042
15	2310	6058	40	2310	6929
16	2310	8489	41	2313	8412
17	2306	8412	42	2310	5806
18	2314	7342	43	2306	11443
19	2316	7514	44	2310	6849
20	2313	7836	45	2312	8105
21	2312	7761	46	2310	6605
22	2312	9030	47	2316	5818
23	2310	8230	48	2315	9448
24	2312	7624	49	2310	7100
25	2312	5114	50	2311	9316

CYCLE GRAPH					
Name: test14		Nodes: 95			
Best SLP					2306
Worst SLP					2317
Range					11
Mean (SLP)					2311.32
Standard Deviation					2.668332813
Succes Rate					0
Best Time					5114
Worst Time					11710
Range					6596
Mean (Time)					7846.76
Standard Deviation					1620.72985

En una entrada de 95 nodos el algoritmo de nuevo no da con el resultado esperado que en este caso de 2,304, pero se mantiene muy cerca de él dando un mínimo de 2,306 y a lo mucho 2,317. El tiempo de ejecución se encuentra en un rango de 5,114 ms a 11,710 ms, con un promedio de 7846.76 ms.

test15

CYCLE GRAPH						CYCLE GRAPH					
Name: test15			Nodes: 110			Name: test15			Nodes: 110		
Expecting SLP Result: 3080											
No. Execution	Result	Time Exe.	No. Execution	Result	Time Exe.	Best SLP	3086	3100	Range	14	
1	3090	9449	26	3089	11946	Worst SLP			Mean (SLP)	3091.62	
2	3093	8726	27	3089	11923	Standard Deviation	3.294336513		Succes Rate	0	
3	3086	16841	28	3089	12160						
4	3093	11323	29	3095	7250	Best Time	7223				
5	3091	8661	30	3097	7223	Worst Time	17401				
6	3088	16839	31	3091	10567	Range	10178				
7	3086	10281	32	3091	11818	Mean (Time)	10696.78				
8	3093	8489	33	3087	11657	Standard Deviation	2606.434513				
9	3093	9194	34	3090	12025						
10	3096	10305	35	3089	14381						
11	3089	14109	36	3090	10486						
12	3090	9609	37	3096	7918						
13	3096	10391	38	3094	8299						
14	3097	9588	39	3090	9501						
15	3095	8185	40	3091	8563						
16	3088	8338	41	3094	10685						
17	3098	10313	42	3089	9714						
18	3093	9710	43	3088	8983						
19	3092	9008	44	3090	13832						
20	3093	9439	45	3095	10687						
21	3092	8736	46	3089	8292						
22	3097	9515	47	3090	15579						
23	3091	12366	48	3088	16912						
24	3100	8584	49	3088	17401						
25	3091	9442	50	3091	9596						

Y por última instancia, la entrada fue de 110 nodos, donde se puede observar que en el mejor de los casos la ejecución arroja un resultado de 3086, cuando el valor esperado es de 3080. El algoritmo ha perdido precisión notoriamente, pero se mantiene cerca del valor esperado dando un promedio de 3091.62. El tiempo de ejecución da un promedio de 10,696 ms, donde el peor de los casos la ejecución puede tomar 17,401 ms, los valores son altos debido al tamaño de la entrada.

5. Situación práctica real

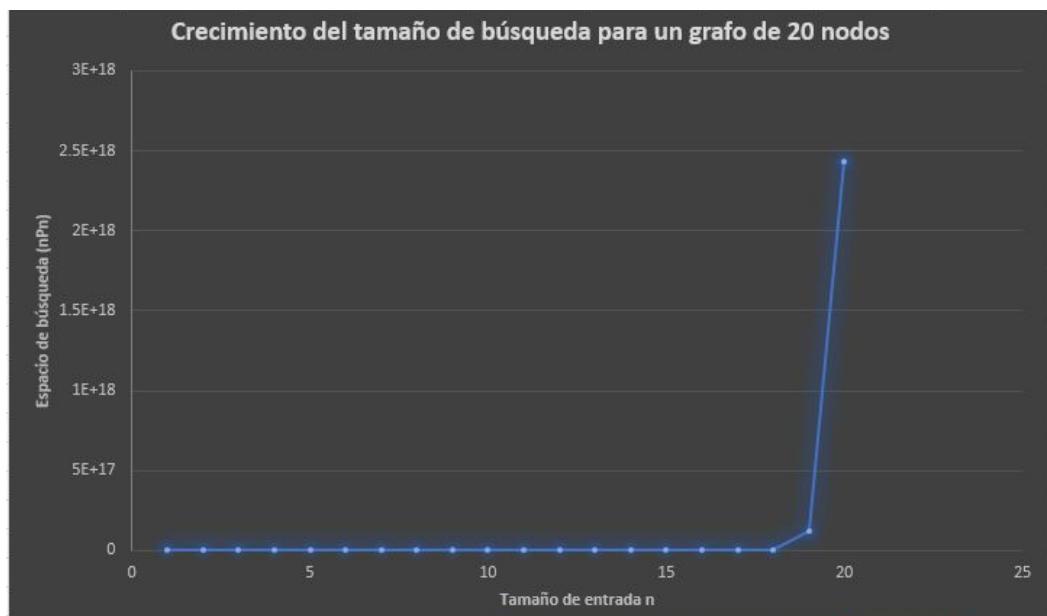
Una empresa busca instalar infraestructuras en su negocio. En este problema las infraestructuras tienen mayor soporte que otras, estas se representan con una etiqueta la cual significa que tan eficaz es como soporte, las de menor valor son las mejores, mientras que las grandes son las peores; las infraestructuras están relacionadas con al menos una o más infraestructuras, sacar el orden de la infraestructuras de tal manera que las mejores apoyen más a las peores.

El algoritmo metaheurístico que utilizamos busca además del etiquetado más chico del grafo total, el orden en el que tiene que estar organizado las etiquetas de tal manera que en las comparaciones exista un valor más pequeño que el otro para poder devolver el valor mínimo hasta el final. Con el problema de estructuras buscamos como hacer que las infraestructuras con mayor soporte puedan apoyar a las peores de tal manera que haya un balance dentro de la empresa. El **S-Labeling Problem** busca que las etiquetas más chicas sean comparadas con las más grandes en el mayor margen que se puede.

Un ejemplo sería dónde poner infraestructuras de metal donde hay de madera, de tal manera que el metal soporte a la madera y la infraestructura esté balanceada.

Dado el tamaño de la instancia del problema a tratar, determinamos una expresión matemática que define el crecimiento del tamaño del espacio de búsqueda (eje Y) con respecto al tamaño de la entrada (eje X, en este caso, número de infraestructuras). En la siguiente gráfica se puede observar el comportamiento puro de este problema.





La fórmula matemática para sacar las posibles soluciones de un grafo fue nPn

Como se puede observar, el número de posibles soluciones para un grafo crece a nivel *factorial*; entre más grande es la entrada, la cantidad de combinaciones es mucho más alta y por consecuencia hay más posibles soluciones. Si se tratase de resolver el problema buscando todas las posibles soluciones sería casi imposible en cuestión de tiempo, debido a que crece de manera factorial el número de combinaciones. En tiempo de cómputo llevaría días en encontrar cada una de ellas o incluso mucho más, es por eso que se decidió implementar un *algoritmo memético*, el cual solo toma una muestra y mediante propiedades de permutaciones y procesos se intenta alcanzar la mejor solución.

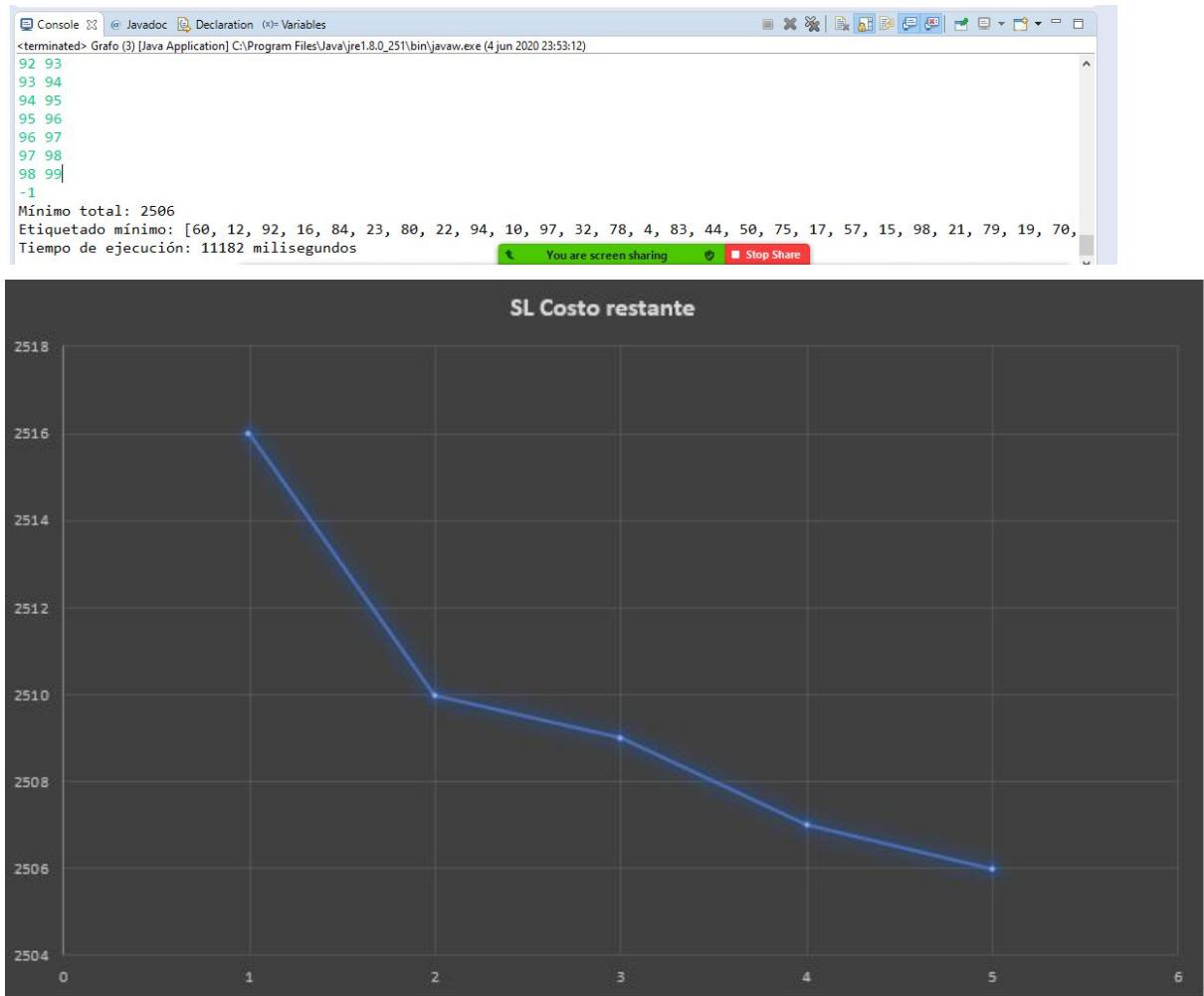
En la solución del problema, se obtiene una solución de las muchas “posibles” (resultado de la complejidad de nivel factorial) que se pueden obtener. Se obtiene el orden de las infraestructuras donde las mejores pueden apoyar a las peores. Cabe destacar que está podría ser una posible implementación del **S-Labeling problem**, no necesariamente sea la mejor manera de resolver el problema. La ventaja de implementar el algoritmo memético como base para solucionar el problema es que se puede obtener una posible solución óptima de miles o millones de posibles soluciones, al igual que se obtiene en un tiempo de ejecución bajo. A continuación se presenta un ejemplo:

Possible solución para un grafo de 100 nodos

Mínimo total: 2506

Etiquetado mínimo: [60, 12, 92, 16, 84, 23, 80, 22, 94, 10, 97, 32, 78, 4, 83, 44, 50, 75, 17, 57, 15, 98, 21, 79, 19, 70, 11, 95, 9, 68, 43, 81, 25, 59, 39, 56, 26, 86, 33, 63, 7, 76, 6, 64, 27, 77, 47, 48, 85, 18, 96, 46, 51, 66, 28, 72, 45, 93, 38, 71, 8, 69, 49, 34, 67, 24, 90, 36, 89, 37, 99, 35, 62, 3, 61, 1, 87, 29, 82, 30, 53, 40, 55, 31, 73, 41, 91, 42, 58, 14, 54, 52, 20, 100, 2, 74, 13, 65, 5, 88]

Tiempo de ejecución: 11182 milisegundos



Como podemos observar con tan solo hacer una iteración pudimos obtener una aproximación muy cercana a la solución que estamos buscando, sin necesidad de tener que encontrar todas las posibles soluciones de un grafo experimental; además, conforme seguimos realizando iteraciones del algoritmo, nos acercamos aún más a la solución, es decir, hay mejores posibilidades de encontrar una posible solución sin realizar lo dicho anteriormente.

6. Aportaciones y participaciones

Las aportaciones del equipo fueron las siguientes:

Reynaldo	Durante el desarrollo del proyecto, realicé todas las implementaciones de las librerías que necesitábamos para el desarrollo del código; además, de programar las operaciones a seguir para obtener los resultados del problema, tales como la generación de la población, el torneo binario, la selección de supervivientes, mutación, la selección final de las poblaciones y algo de la búsqueda local. De igual manera hice un generador de grafos para probar el programa.
Julio	Calcule la complejidad computacional y espacial y explique algunos de las funciones. Edite la búsqueda local, lo cual incrementó la eficiencia muy abruptamente(pasamos de 1000 pruebas a 1 para dar la respuesta de una instancia mediana)
Lizbeth	Durante el proyecto realicé la programación detrás de la ejecución de la función creadora de población, así como la lógica detrás de cada individuo. Realicé en conjunto con mi compañero Reynaldo la implementación de la clase Permutaciones con ayuda de la librería Opt4J. Me encargué de la estadística de las ejecuciones e interpretación de los datos arrojados. Aporté la teoría descriptiva de las metaheurísticas y sus correspondientes algoritmos, al igual que su implementación.
Marlene	Al transcurso de este proyecto colaboré en documentación del reporte, también en la codificación la clase de <i>Crossover</i> y en esta última entrega realicé las predicciones correspondientes al número de entradas que tuviera el grafo.

7. Retroalimentación

Miembros	Aprendizaje	Qué fue lo que más nos agradó	Qué fue lo que no nos gustó
Reynaldo	Durante el transcurso del curso he aprendido bastantes cosas interesantes, tales como la importancia de los algoritmos en la vida cotidiana, al igual que sus propiedades y sus aplicaciones. Aprendí más acerca de las sumatorias, sobre tipos de algoritmos, identificar qué es óptimo y no, soluciones eficientes pero no del todo eficaces, entre muchos otros temas	Me gustó bastante el curso, sin duda alguna, desde mi opinión personal, uno de los mejores cursos que he tenido en mi vida, me encantó absolutamente todo, aprendí mucho, las clases súper bien hechas, hubo variedad de actividades, temas interesantes, proyectos que eran un reto. Me encantó absolutamente todo	En lo personal, no hubo algo que no me gustaría del curso, superó por mucho mis expectativas, todo me gustó. Lo único que no me gustó fue ejecutar 750 veces las instancias del proyecto final
Julio	Aprendí mucho sobre la complejidad computacional, como la forma de calcularla, como mejorarla y su repercusión en los programas. También aprendí sobre nuevos algoritmos que no conocía y los cuales pienso me servirán mucho en un futuro	Me pareció una clase muy entretenida, porque yo realmente esperaba algo muy tedioso como leer libros completos. Me gusto mucho la disposición del maestro y su manera de enseñar, yo creo que ha sido uno de mis mejores 5 profesores del Tec.	Creo que lo único que no me gusto fue la tarea cuando hicimos pruebas con la computadora para ver cómo afectaba la complejidad al tiempo en el que el programa corre, pero si fue útil para visualizar ese crecimiento. Y no me gusto porque mi computadora se estaba muriendo con las últimas instancias.
Lizbeth	Lo que más aprendí de este curso es la visualización que tengo acerca de los algoritmos hoy en día. Me es difícil ver un algoritmo sin ponerme	Lo que más me ha gustado de esta clase es la forma de enseñanza, el cómo usted profesor se preocupaba por nosotros. Realmente	Me hubiera gustado programar más, hacer más ejercicios prácticos de grafos. Al igual que no me gustó tener que ejecutar 750 veces el mismo

	<p>a pensar en la complejidad que tendrá; sus repercusiones en tiempo y almacenamiento. Realmente fue muy enriquecedor para mi el investigar a gran medida distintas técnicas de análisis de algoritmos, al igual que aprender a gran escala acerca de grafos.</p>	<p>admiré todas las tardes que se dedicó a darnos asesoría y eso la verdad se aprecia muchísimo. Al igual que Julio, lo considero de los mejores profesores que he tenido dentro del Tec.</p>	<p>algoritmo. Fue agobiante.</p>
Marlene	<p>Este curso fue por demás provechoso para mí, la verdad es que desde que empezó la materia me pareció muy interesante el temario que íbamos a ver y cumplió ampliamente las expectativas que tenía. Logré aprender el cómo analizar algoritmos a profundidad y también los distintos tipos de algoritmos que existen, lo cual considero importante, ya que como programadores podemos apoyarnos mucho en la investigación para hacer algoritmos eficientes.,</p>	<p>Realmente me agració mucho la manera en que el maestro daba la clase. Se notaba mucho el interés por que aprendiéramos y en sí por la materia, nunca dudó en darnos asesorías</p>	<p>En sí todo me gustó, pero si tuviera que decir algo que no me agració tanto sería que el proyecto fue muy pesado, en sí todo el proyecto está muy genial de desarrollar, pero me ha ocasionado mucha presión las entregas, pero reitero no es que no me guste, solo fue lo único que me causó más estrés en toda la materia</p>

8. Conclusiones globales del proyecto

El proyecto final fue una excelente manera para retornos como programadores y aplicar lo visto durante el curso. Desde el principio supimos que sería un gran reto para nosotros desarrollar el análisis y algoritmo del problema SLP, sin embargo esto no nos desmotivó, al contrario logró motivarnos aún más, porque sería una forma perfecta de demostrar que somos capaces de cumplir con proyectos de esta magnitud.

No fue fácil, tuvimos que llenarnos de vídeos e incontables bibliografías, inclusive múltiples asesorías con el profesor, solamente para entender qué era este algoritmo LP, pero finalmente gracias a las pequeñas entregas, entendimos mucho mejor la estructura del algoritmo, hasta lograr entenderlo en su totalidad.

Durante el desarrollo de este proyecto, pudimos concluir varias cosas acerca del S-Labeling Problem. En efecto, el S-Labeling problem es un problema NP-completo, pudimos comprobar mediante nuestras experimentaciones que las soluciones dadas por la implementación de nuestro algoritmo memético, no tenían un tiempo polinomial sobre el

problema, ya que el tiempo variaba de un lapso de tiempo corto a uno largo para poder encontrar la solución, al igual que no en todos los casos daba la solución correcta.

Finalmente, pese a que al principio estábamos totalmente inexpertos e incluso perdidos, nunca nos desmotivamos y eso ayudó a que pudiéramos completar la entrega final, fue un trabajo en equipo lleno de retos personales y profesionales donde la perseverancia de trabajar diario en el proyecto fue nuestro mejor aliado para lograr los objetivos establecidos.

9. Referencias

Fertin, G., Rusu, I., & Vialette, S. (s.f.). *Algorithmic Aspects of the S-Labeling Problem*

[PDF].

Moujahid, A., Inza, I., Larrañaga, P. (s.f.) *Tema 2. Algoritmos Genéticos*. [PDF]. Universidad del País Vasco–Euskal Herriko Unibertsitatea

Neri, F., Moscato, P., Cotta, C. (2012). *Handbook Of Memetic Algorithms*. Springer.

Sinnl, M. (2019). *Algorithmic expedients for the S-labeling problem* [PDF]. Elsevier Ltd.