Comparación entre el Algoritmo Genético Simple y el Algoritmo de Optimización Inspirado en Magnetismo con Repulsión de Corto Alcance aplicados al Problema del Agente Viajero

Andrea Itzel González Vargas
Carlos Gerardo Acosta Hernández

Cómputo Evolutivo 2017-1
Facultad de Ciencias UNAM

1. Introducción

Panorama general

Para este proyecto final decidimos realizar una comparación entre el Algoritmo Genético Simple (**AGS**) y el Algoritmo de Optimización Inspirado en Magnetismo (**MOA**), con un operador enfocado a la **exploración**, propuesto en el artículo[1] que escogimos para la exposición en clase.

Como punto de referencia, elegimos implementar un programa similar al que hicimos primero para encontrar soluciones óptimas de varias instancias del Problema del Agente Viajero (**TSP**) usando el MOA. Lo anterior por nuestra experiencia previa con el desarrollo del primer proyecto[2], que nos sirvió de guía para llevar a cabo la construcción de nuestra propia versión de un MOA aplicado a TSP.

La idea es verificar si el MOA, siendo una propuesta reciente[3] en el mundo del cómputo evolutivo, es una buena opción para resolver problemas de este tipo como ya hemos visto con el AGS y decidir si es en algún aspecto competitivo respecto a su adversario. La entrada de ambos programas es una instancia del TSP, para la que ambos algoritmos mostrarán la mejor solución encontrada en cada una de sus iteraciones.

Para manejar y procesar las instancias del TSP, proporcionadas por la colección *TSPLIB*[4] y recopiladas en archivos individuales bajo el directorio *tsp/*, hacemos uso de una biblioteca para *Java*, llamada *TSPLIB4J*[5].

[1] Magnetic-inspired optimization algorithms: Operators and structures
[2] Referencia al repositorio del primer proyecto: Proyecto-TSP
[3] Propuesto por *N.M.H Tarayani* y *T.M.R. Akbarzadeh* en 2008
[4] TSPLIB: página oficial
[5] TSPLIB4J: *GitHub*

Marco teórico

Creemos que ya no es necesario ser muy específicos respecto al AGS implementado y presentado previamente en nuestro primer proyecto, por lo que esta sección referirá únicamente al MOA.

En la teoría de los campos magnéticos, las partículas pueden atraerse o repelerse unas con otras dependiendo de su polaridad. Como muchos algoritmos en el campo de estudio de la optimización, el MOA es una propuesta inspirada en este principio observado en la naturaleza. Para este algoritmo, el conjunto de posibles soluciones son partículas con propiedades magnéticas, diseminadas en el espacio de búsqueda, donde cada partícula es capaz de incidir una sobre la otra con una fuerza de atracción de largo alcance -tal como el concepto de fuerza electromagnética.

Similar a los algoritmos de optimización por enjambre o nube de partículas (**PSO**[6]), donde los individuos buscan imitar a aquel con mayor grado de adaptación, las partículas en el MOA con mejor calificación poseen masa y campo magnético mayores, resultando en una mayor fuerza de atracción ejercida sobre las otras partículas.

Por un lado, este comportamiento cubre el concepto de explotación revisado en nuestro curso, sin embargo, el MOA cuenta además con una mejora considerable respecto a los PSO convencionales.

Es considerado una de las debilidades de los PSO, el hecho de que sólo las mejores partículas, son las que tienen una repercusión significativa sobre las demás menos adaptadas, que no tienen incidencia en otras. En este sentido, estos algoritmos dejan fuera de consideración, importante información que pueden contener las partículas con bajo grado de adaptación. Es por ello que en el esquema del MOA que revisamos, se considera la posibilidad de que todos las partículas cuenten con cierto potencial de atracción, independientemente de su grado de adaptación (aunque éste lo afecte en magnitud).

Asimísmo, se incluye un operador de repulsión de corto alcance (**SRR**[7]). Este operador -que explicaremos más ampliamente en secciones posteriores-, permite que el algoritmo no se concentre únicamente en las partículas más cercanas a la solución, pues actúa como a manera de repulsión cuando la distancia entre dos partículas es menor a cierto umbral predefinido, manteniendo la diversidad y previniendo una convergencia prematura en óptimos locales.

Además del esfuerzo implementado para la optimización de "tours" en las instancias del TSP, el MOA ya ha sido puesto a prueba en varias aplicaciones de Inteligencia Artificial, como el entrenamiento de redes neuronales y ha sido propuesta ya una versión binaria del algoritmo. Las implementaciones de los MOA, resultan en un paradigma dual de interacción entre partículas con la fuerza de repulsión/atracción, que puede devenir un balance entre exploración y explotación.

Siguiendo esa línea, las virtudes que supone el esquema del MOA parecen prometedoras frente a la contraparte de este proyecto, el AGS.

[6]En inglés: Particle Swarm Optimization

[7]En inglés: Short-Range Repulsion

Herramientas

Como herramienta fundamental, proporcionamos el pseudo-código del MOA básico que no contempla más operadores que el de actualización de posición de una partícula dada su velocidad.

ALGORITHM 1: Magnetic-inspired Optimization Algorithm (MOA)

$t \leftarrow 0$; $X^t \leftarrow randomValidPopulation()$;
repeat
  $t \leftarrow t + 1$;
  for *each* x *in* X^t **do**
    evaluate x and store performance in magnetic fields B^t;
  end
  Normalize B^t;
  for *each* x *in* X^t **do**
    evaluate mass of x and store it in M^t;
  end
  for *all* x_{ij}^t *in* X^t **do**
    $F_{ij} \leftarrow 0$;
    find Nij;
    for *each* x_{uv}^t *in* N_{ij} **do**
      $F_{ij} \leftarrow F_{ij} + \dfrac{(x_{uv}^t - x_{ij}^t) \times B^t}{D(x_{ij}^t, x_{uv}^t)}$;
    end
  end
  for *all* x_{ij}^t *in* X^t **do**
    $v_{ij,k}^{t+1} = \dfrac{F_{ij,k}}{M_{ij,k}} \times R(u_k, l_k)$;
    $x_{ij,k}^{t+1} = x_{ij,k}^t + v_{ij,k}^{t+1}$;
  end
until $t = maxGen$;

N_{ij} := Conjunto de vecinos de la partícula x_{ij}.
F_{ij} := Fuerza de atracción ejercida sobre la partícula x_{ij}.
$D(x_1, x_2)$:= Distancia entre x_1 y x_2.

Esquema

Las figuras siguientes refieren a dos elementos más del esquema de implementación que consideramos relevantes destacar: El operador de repulsión y la forma de la estructura de la población del MOA.

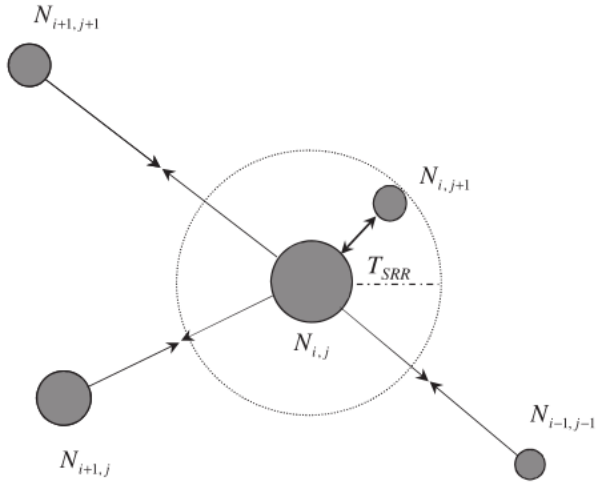

Figura 1: Representación del umbral T_{SRR} de acción del operador SRR.

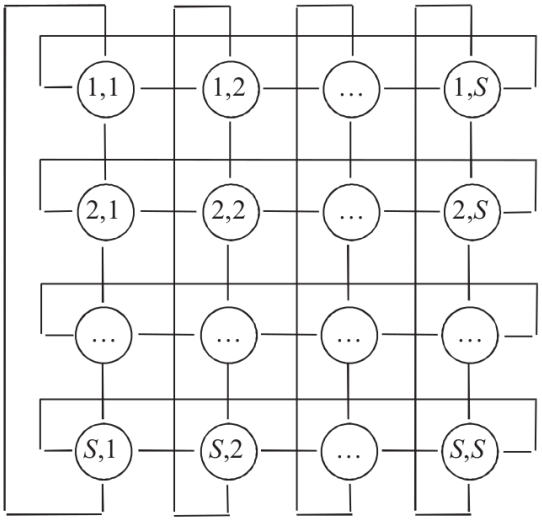

Figura 2: Estructura de la población. Cada nodo es una partícula.

Datos de la computadora experimental

Memoria RAM: 7.7GiB
Procesador: Intel® Core™ i7-3630QM CPU @ 2.40GHz x 8

Diseño de implementación

La implementación de nuestro MOA es un enfoque original, basado en el algoritmo básico recién mostrado. Intentamos seguir las especificaciones de 6.[2], sin embargo, por falta de información nos resultó imposible seguir el esquema de votaciones por ciudad en el vector de posición de la partícula, no por que la codificación no fuese clara, sino porque no logramos empatar los conceptos de velocidad y movimiento de partículas con esa propuesta, además de que no nos pareció muy claro en dicho artículo. Es por ello que decidimos conservar una codificación similar a la del AGS implementado en el primer proyecto y nos auxiliamos del concepto de movimiento de partículas por secuencias de operadores de permutación (como en 6.[4]), con una probabilidad determinada por la fuerza de atracción ejercida sobre las partículas.

Más detalles sobre el diseño de la implementación se pueden encontrar en la Sección 3 y siempre se puede consultar el código fuente del proyecto en el directorio *src/*.

A partir de ahora, nos referiremos a la implementación del MOA como SRMOA, por el operador que incluye en su ejecución.

2. Planteamiento

Objetivo

Determinar si el SRMOA (Tipo III) es una opción cuyo desempeño es competente respecto al algoritmo genético simple (AGS) en la resolución del problema del agente viajero (TSP).

Hipótesis

Esperamos que ambos tengan un desempeño similar para las distintas instancias del TSP. Creemos también que conforme la dimensión de las instancias del TSP crezcan, el SRMOA tendrá una mejora significativa respecto a instancias menores y pueda superar al AGS en tales magnitudes. Respaldando ésta hipótesis tenemos las conclusiones del artículo sobre el MOA que mostraba que mientras mas crecía el problema, el SRMOA mejoraba su desempeño. Mientras que por otro lado nuestras pruebas con el AGS nos hacían ver que mientras más crecía el problema, disminuía el desempeño de éste algoritmo.

Asimismo, suponemos que la implementación del SRMOA será más sencilla que la del AGS.

Metodología

Implementaremos el SRMOA y utilizaremos el AGS que previamente implementamos para el Proyecto 1. Probaremos los algoritmos con los archivos *burma14*, *ulysses16*,*gr17*, *kroA100* y *pr264*. Para cada archivo las tuplas de parámetros de las tablas siguientes un total de cinco veces. En conjunto se harán 25 corridas por algoritmo.

Los **parámetros** que se utilizarán para el **AGS** serán:

Pc	**Pm**
0.6	0.001

Y para el **SRMOA**:

α	ρ	T_{SRR}	C_{SRR}
100	100	1	0.1

Para ambos algoritmos, se fijará el tamaño de la **población** en 225 individuos y se establecerá 1000 como máximo número de **generaciones** (siendo este el criterio de detención de las iteraciones).

Para hacer la comparación entre ambos algoritmos graficaremos el exceso porcentual promedio de peso del mejor circuito vs el número de generaciones de cada archivo para cada algoritmo, a demás del promedio del conjunto de los cinco archivos. Compararemos también el tiempo promedio que se tardó cada algoritmo para correr cada archivo.

3. Especificación

Campos magnéticos

Una vez evaluados los campos magnéticos de todas las partículas, estos son normalizados de manera que tomen un valor en [0, 1], esto se hace por medio de la siguiente fórmula

$$B_{ij} = \frac{B_{ij} - B_{min}}{B_{max} - B_{min}} \tag{1}$$

donde B_{max} y B_{min} son el campo más grande de todas las partículas y el menor respectivamente. El resultado es que las partículas con las mejores rutas tendrán los campos más grandes, y las que tengan peores rutas tendran campos más pequeños.

Masa

El siguiente paso es asignarle una masa M_{ij} a cada partícula x_{ij}, lo cual se hace simplemente con la fórmula

$$M_{ij} = \alpha + \rho \times B_{ij} \tag{2}$$

Fuerza de atracción

Se procede a evaluar la fuerza F_{ij} de atracción que es ejercida en cada partícula por sus vecinos. Definimos la fuerza como un vector de manera que $F_{ij} = (f_0, f_1, ..., f_{n-1})$, recordando que n es el número de ciudades.
El valor de cada f_k en F_{ij} para x_{ij} lo definimos como:

$$f_k = \sum_{x_{uv} \in N_{ij}} \frac{(x_{uv}[k] - x_{ij}[k]) \times B_{ij}}{D(x_{ij}, x_{uv})} \tag{3}$$

donde $x[k]$ es el número de la ciudad en la posición k en x y la distancia $D(x_{ij}, x_{uv})$ la calculamos tomando a las partículas x_{ij} y x_{uv} como dos puntos en un espacio $\mathbb{R}^n$ de manera que cada ciudad en el conjunto de ciudades de cada partícula representa una coordenada.

Velocidad

La velocidad de las partículas la definimos de igual manera como un vector $V_{ij} = (v_0, v_1, ..., v_{n-1})$. Para calcular la velocidad de x_{ij} tomamos F_{ij} y M_{ij} y aplicamos la siguiente función para todos los valores v_k en V_{ij}

$$v_k = \frac{f_k}{M_{ij}} \times R(0, n) \tag{4}$$

donde $R(0, n)$ es un número aleatorio entre 0 y n. Después normalizamos los valores v_k para que tengan un valor en [0, 1], la razón se explica a continuación.

Movimiento

Como decidimos codificar el problema representando las rutas con los índices de las ciudades, tuvimos que adecuar el algoritmo MOA para que al recalcular la posición de cada partícula cada ruta siga siendo válida y sin la necesidad de hacer un corrector, para lo cuál decidimos que las entradas de cada vector de velocidades fueran una probabilidad, por eso es que las normalizamos. Éstas probabilidades nos permiten definir si se va a realizar una permutación en cada índice de la ruta, i.e. para cada x_{ij} se toma su velocidad V_{ij} y cada uno de sus valores v_k funciona como una probabilidad de que la ciudad en $x_{ij}[k]$ sea permutada con otra ciudad. Para esto se procede a obtener un número aleatorio r para cada ciudad, si $r \leqslant v_k$ entonces $x_{ij}[k]$ es permutada.

Cada permutación no es aleatoria, la manera en que se decide con qué ciudad se va a permutar cada ciudad es la siguiente:

Se toma al vecino $x_{uv} \in N_{ij}$ con el campo magnético más grande. Para cada ciudad $x_{ij}[k]$ a permutar si $x_{ij}[k] = x_{uv}[k]$ no se hace nada. De lo contrario se busca el índice m tal que $x_{ij}[m] = x_{uv}[k]$ y se permuta $x_{ij}[k]$ con $x_{ij}[m]$. Claro que en todo caso se evita cambiar la primer ciudad que siempre va a ser la primera en visitarse.

Operador SRR

La última parte del algoritmo consiste en lo siguiente:
Para cada vecino x_{uv} de cada partícula x_{ij}, si $D(x_{ij}, x_{uv}) < T_{srr}$ entonces se le aplica el operador SRR a x_{ij}. Éste operador permuta las ciudades de x_{ij} de manera aleatoria si es que se cumple una probabilidad, i.e. hacemos un vector $A_{ij} = (a_0, a_1, ..., a_{n-1})$ similar al vector de velocidad, pero los valores a_k estan definidos de ésta manera:

$$a_k = C_{srr} \times R(0, n) \tag{5}$$

Procedemos a normalizar los valores de A_{ij} para convertirlos en probabilidades, y finalmente por cada a_k obtenemos un número aleatorio r en [0, 1], si $r \leqslant a_k$ entonces se permuta la ciudad $x_{ij}[k]$ con otra ciudad aleatoria, evitando siempre permutar la primer ciudad.

4. Experimentación

Con los parámetros determinados en la sección del planteamiento del proyecto (2 *Metodología*), en esta sección incluímos las siguientes gráficas, una por cada instancia del TSP que elegimos, para representar visualmente el exceso porcentual promedio respecto de las generaciones del total de ejecuciones realizadas sobre cada archivo. Para hacer aún más sencilla la observación comparativa, están representados con acotaciones de color ambos algoritmos.

4.1. Instancia burma14

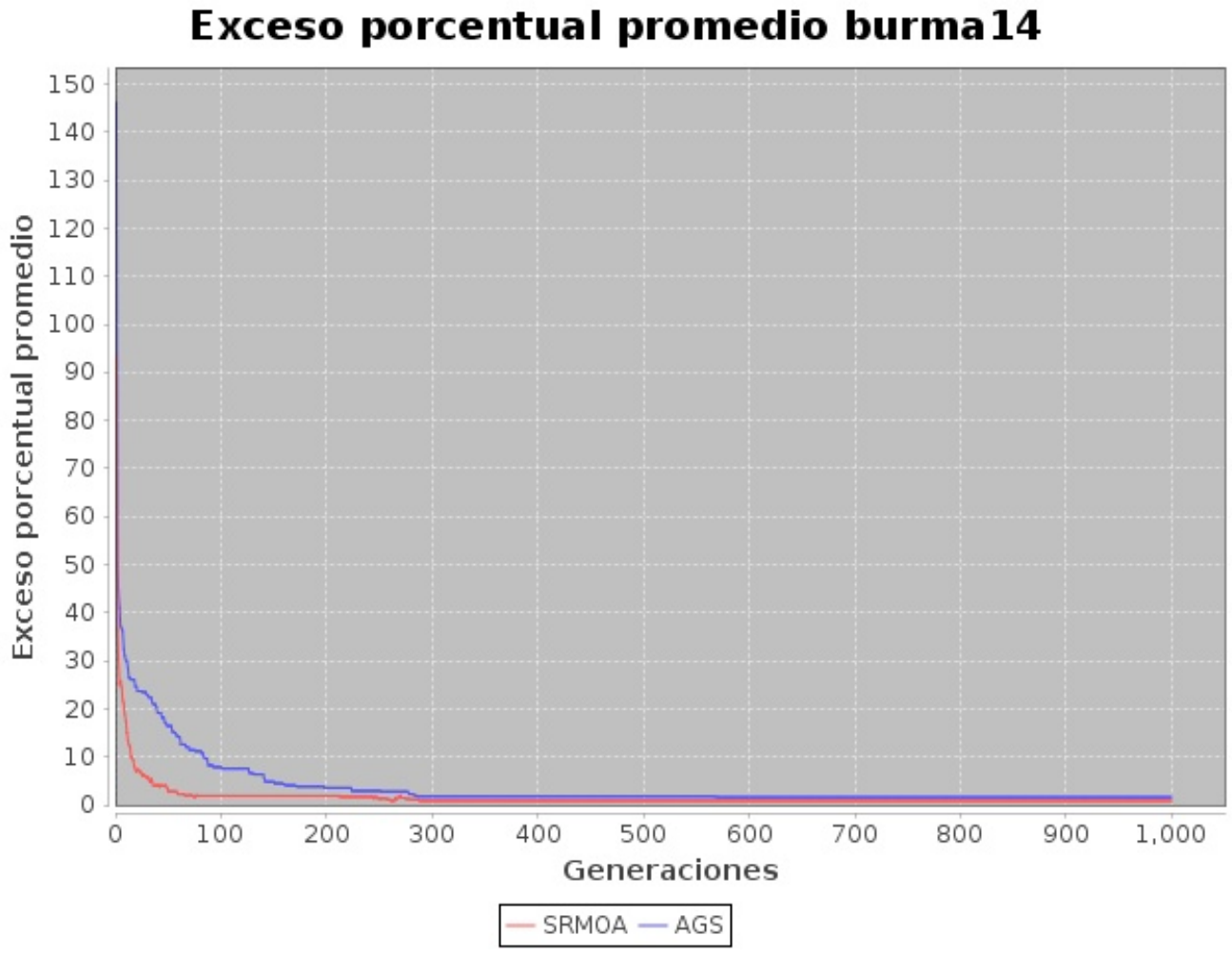

Figura 3: Exceso porcentual promedio por generación en instancia de TSP con 14 ciudades.

4.2. Instancia ulysses16

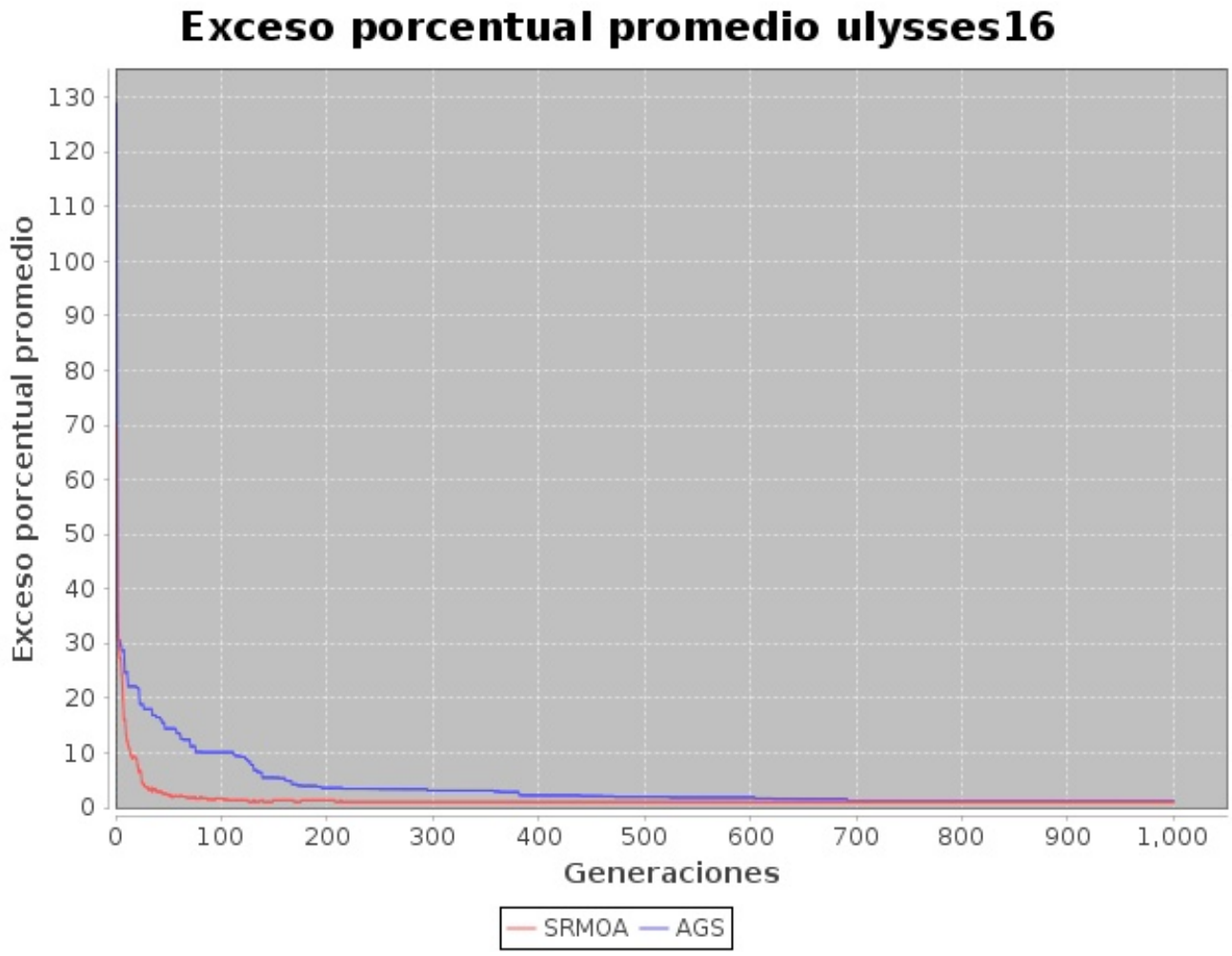

Figura 4: Exceso porcentual promedio por generación en instancia de TSP con 16 ciudades.

4.3. Instancia gr17

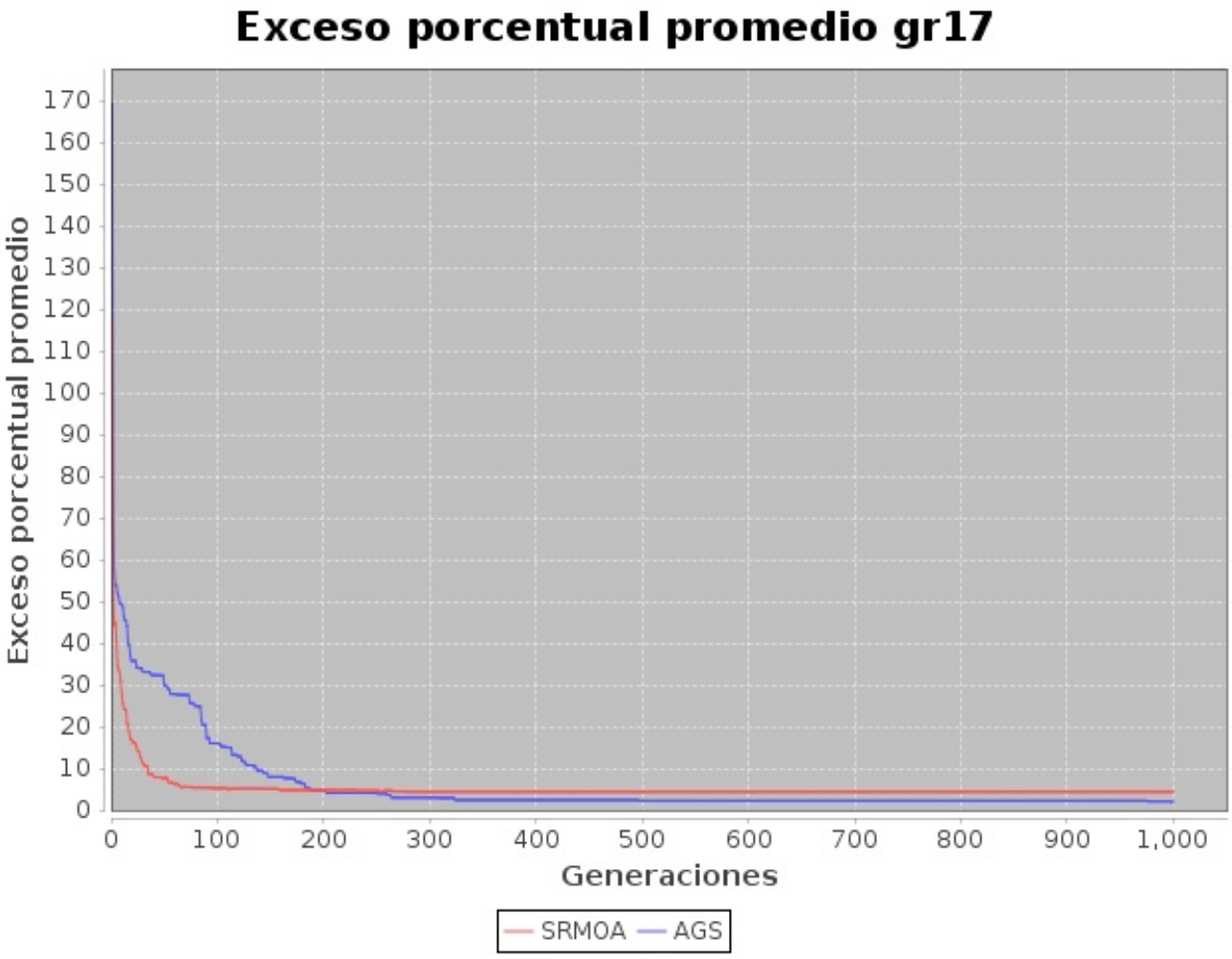

Figura 5: Exceso porcentual promedio por generación en instancia de TSP con 17 ciudades.

4.4. Instancia kroA100

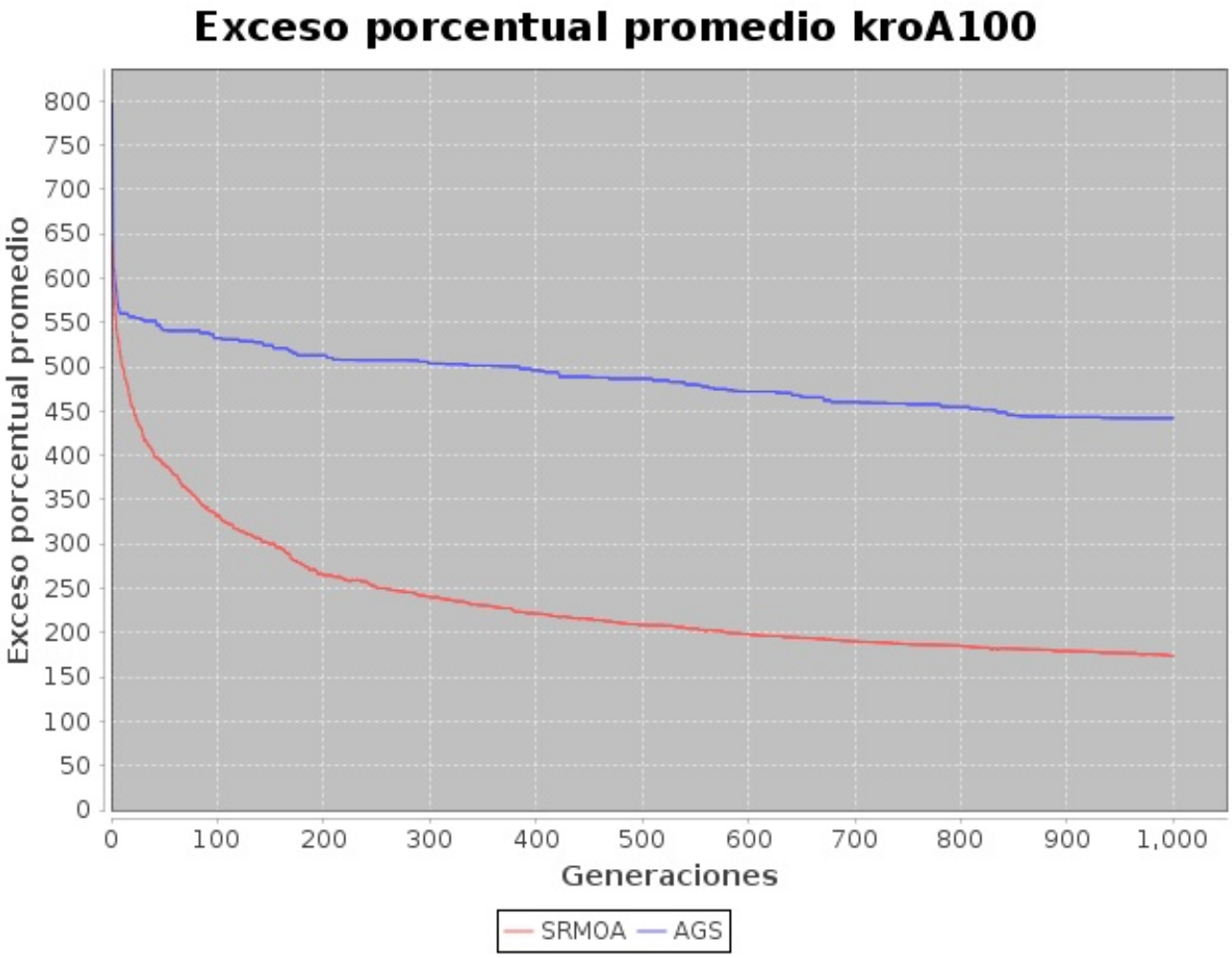

Figura 6: Exceso porcentual promedio por generación en instancia de TSP con 100 ciudades.

4.5. Instancia pr264

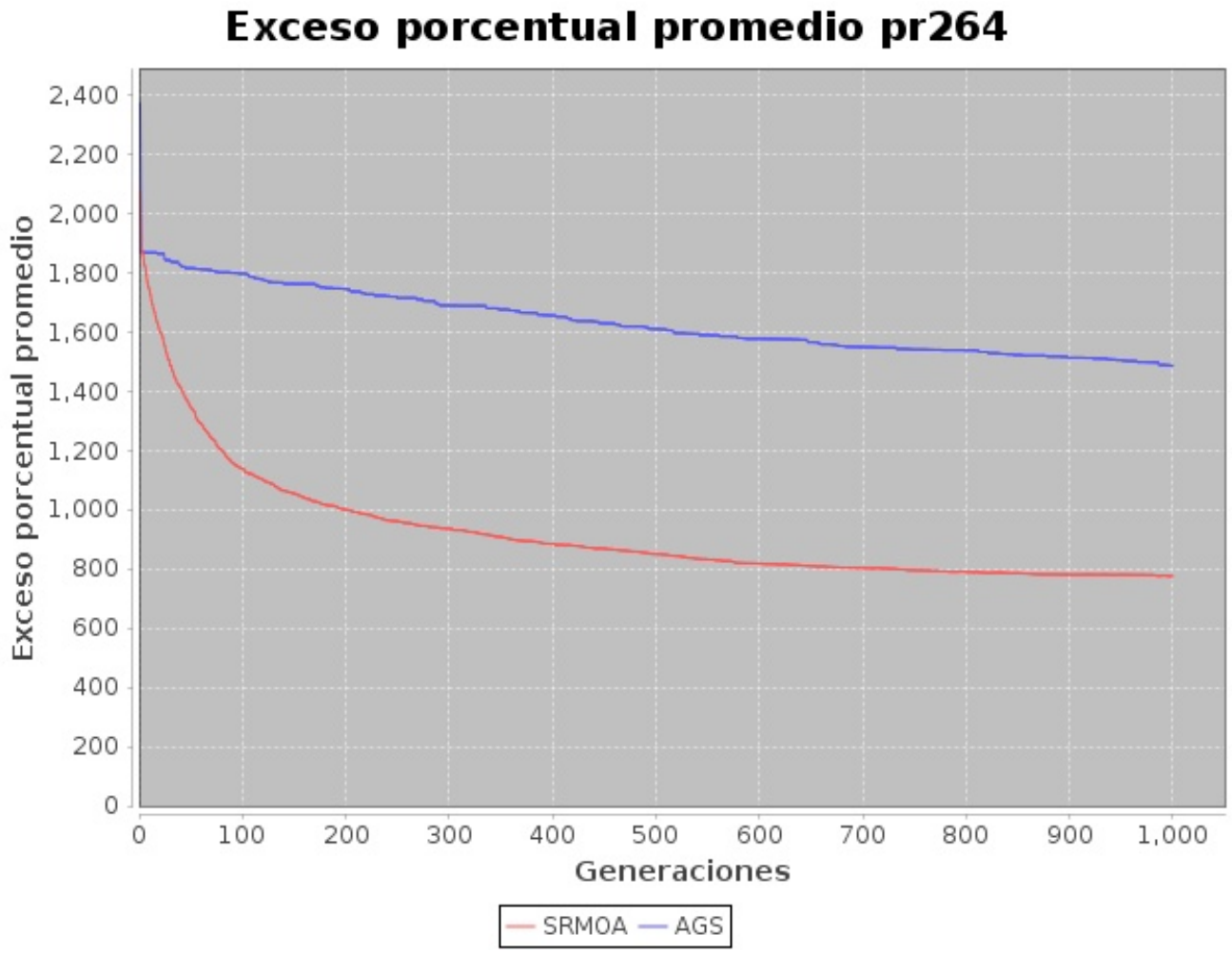

Figura 7: Exceso porcentual promedio por generación en instancia de TSP con 264 ciudades.

4.6. Total

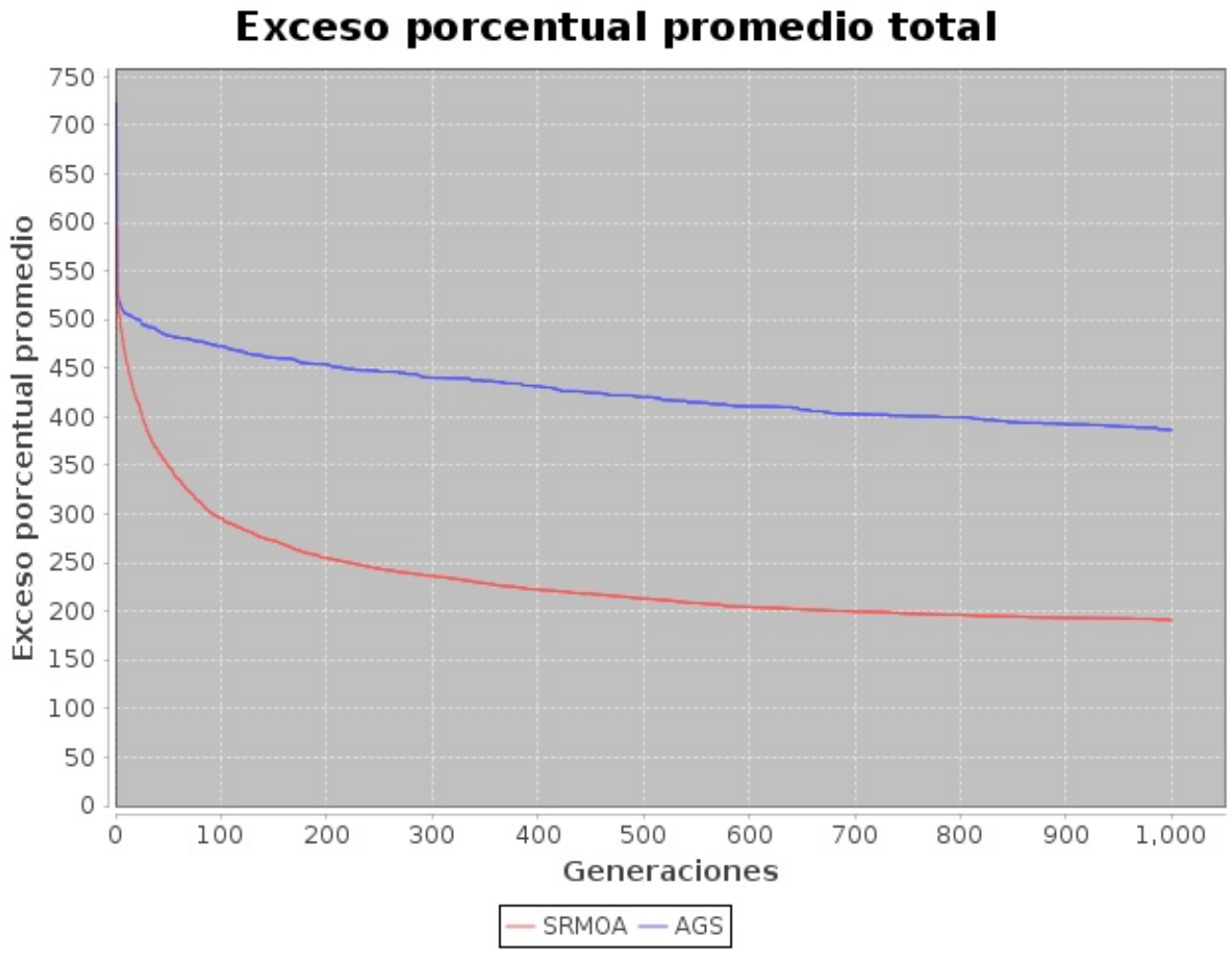

Figura 8: Exceso porcentual promedio por generación para todas las instancias de TSP escogidas (burma14, ulysses16, gr17, kroA100, pr264).

4.7. Tiempo de ejecución

Con el fin de ser más específicos respecto de la representación visual previa de la experimentación que se llevó a cabo, proporcionamos los siguientes cuadros de datos. En conjunto, forman una tabla completa de valores de exceso porcentual y tiempo de ejecución por generación en el procesamiento de cada instancia del TSP. Se contemplan en el primer cuadro, *burma14*, *ulysses16*, *gr17*, *kroA100*, *pr264*.

Cuadro 1: Exceso porcentual promedio y tiempo de ejecución por generación

Generación	burma14				ulysses16				gr17			
	SRMOA		AGS		SRMOA		AGS		SRMOA		AGS	
	Exceso %	Tiempo (s)	Exceso %	Tiempo (s)	Exceso %	Tiempo (s)	Exceso %	Tiempo (s)	Exceso %	Tiempo (s)	Exceso %	Tiempo (s)
1	93.50	0.0034	146.19	0.0182	69.89	0.0032	128.91	0.0032	117.99	0.0010	169.26	0.0010
50	2.82	0.3458	16.46	0.2032	2.38	0.3842	14.44	0.2098	7.54	0.0838	29.88	0.1044
100	1.88	0.6656	7.85	0.3990	1.58	0.7614	10.09	0.4278	5.28	0.1656	15.99	0.1944
150	1.88	0.9826	4.83	0.6004	1.31	1.1378	5.45	0.6168	5.10	0.2474	7.93	0.2876
200	1.88	1.3000	3.54	0.8102	1.31	1.5136	3.55	0.8178	4.79	0.3290	4.52	0.3798
250	1.39	1.6190	2.90	0.9868	0.99	1.8904	3.39	1.0036	4.66	0.4106	4.03	0.4652
300	0.85	1.9368	1.76	1.1594	0.99	2.2666	3.10	1.2038	4.36	0.4936	2.96	0.5478
350	0.85	2.2546	1.76	1.3646	0.99	2.6424	3.10	1.3970	4.36	0.5746	2.38	0.6684
400	0.85	2.5712	1.76	1.5266	0.99	3.0182	2.22	1.5856	4.36	0.6658	2.38	0.7786
450	0.85	2.8880	1.76	1.6884	0.99	3.3940	2.09	1.7700	4.36	0.7538	2.38	0.8808
500	0.85	3.2046	1.76	1.8606	0.99	3.7878	1.95	1.9508	4.36	0.8354	2.22	0.9812
550	0.85	3.5212	1.76	2.0224	0.99	4.1638	1.85	2.1316	4.36	0.9170	2.22	1.0768
600	0.85	3.8378	1.59	2.1810	0.99	4.5392	1.85	2.3140	4.36	0.9980	2.22	1.1714
650	0.85	4.1548	1.59	2.3378	0.99	4.9330	1.55	2.5072	4.36	1.0790	2.22	1.2732
700	0.85	4.4710	1.59	2.5034	0.99	5.3084	1.23	2.6960	4.36	1.1596	2.22	1.3606
750	0.85	4.7882	1.59	2.6880	0.99	5.6840	1.23	2.9016	4.36	1.2412	2.22	1.4518
800	0.85	5.1052	1.59	2.8474	0.99	6.0600	1.23	3.1092	4.36	1.3228	2.22	1.5438
850	0.85	5.4226	1.59	3.0052	0.99	6.4356	1.23	3.3062	4.36	1.4040	2.22	1.6290
900	0.85	5.7474	1.59	3.1816	0.99	6.8112	1.23	3.5036	4.36	1.4872	2.22	1.7130
950	0.85	6.0736	1.59	3.3486	0.99	7.1868	1.23	3.7024	4.36	1.5696	2.22	1.8182
1000	0.85	6.3904	1.59	3.5432	0.99	7.5624	1.23	3.8978	4.36	1.6516	1.96	1.9106

Cuadro 2: Exceso porcentual promedio y tiempo de ejecución por generación

Generación	kroA100				pr264				Total			
	SRMOA		AGS		SRMOA		AGS		SRMOA		AGS	
	Exceso %	Tiempo (s)	Exceso %	Tiempo (s)	Exceso %	Tiempo (s)	Exceso %	Tiempo (s)	Exceso %	Tiempo (s)	Exceso %	Tiempo (s)
1	639.55	0.0040	796.88	0.0044	2079.40	0.0078	2373.03	0.0218	600.06	0.0039	722.85	0.0097
50	390.30	1.6788	541.39	0.1890	1345.29	10.1086	1816.00	0.4350	349.67	2.5202	483.63	0.2283
100	331.52	3.3818	532.43	0.3832	1138.73	20.3458	1797.52	0.8486	295.80	5.0640	472.78	0.4506
150	299.99	5.0824	524.30	0.5742	1054.86	30.6830	1763.33	1.2424	272.63	7.6266	461.17	0.6643
200	264.98	6.7842	513.05	0.7600	1000.38	40.8878	1746.00	1.6788	254.67	10.1629	454.13	0.8893
250	251.32	8.4844	507.65	0.9336	961.16	51.2190	1715.38	2.0952	243.90	12.7247	446.67	1.0969
300	240.36	10.1868	503.81	1.1434	935.74	61.4600	1689.98	2.5344	236.46	15.2688	440.32	1.3178
350	231.01	12.3874	501.44	1.3416	908.32	71.7118	1676.69	2.9432	229.11	17.9142	437.08	1.5430
400	221.74	14.3016	496.07	1.5458	884.66	81.9812	1656.17	3.3402	222.52	20.5076	431.72	1.7554
450	215.54	16.0026	488.88	1.7366	868.81	92.3034	1629.72	3.7606	218.11	23.0684	424.97	1.9673
500	208.69	17.7026	486.69	1.9244	849.94	102.5096	1610.44	4.1588	212.97	25.6080	420.61	2.1752
550	205.02	19.4038	480.06	2.1016	831.57	112.8282	1590.01	4.5450	208.56	28.1668	415.18	2.3755
600	198.33	21.1072	472.19	2.3104	818.97	123.0452	1577.40	4.9460	204.70	30.7055	411.05	2.5846
650	194.22	22.8328	466.55	2.4882	809.98	133.2412	1565.83	5.3180	202.08	33.2482	407.55	2.7849
700	190.22	24.5404	460.37	2.6734	803.33	143.4490	1549.86	5.7156	199.95	35.7857	403.05	2.9898
750	187.11	26.2438	457.88	2.8984	795.74	153.7648	1541.38	6.0938	197.81	38.3444	400.86	3.2067
800	185.48	27.9664	454.80	3.0986	790.33	164.0196	1538.45	6.4878	196.40	40.8948	399.66	3.4174
900	179.36	31.3692	443.36	3.4708	780.48	184.3680	1513.69	7.2832	193.21	45.9566	392.41	3.8304
950	177.12	33.0692	442.28	3.6446	781.22	194.7260	1505.57	7.6670	192.91	48.5250	390.58	4.0362
1000	173.76	34.7700	442.14	3.8190	775.49	205.0244	1487.29	8.0442	191.09	51.0798	386.84	4.2430

5. Conclusiones

Después de recolectar la información en nuestras tablas de datos y representarlas visualmente con ayuda de las gráficas, exhibidas en la sección anterior, podemos verificar nuestras hipótesis prevías en el planteamiento del proyecto.

En primer lugar, esperábamos encontrarnos con un desempeño similar de ambos algoritmos para las cinco instancias del TSP. Observamos que al menos para las primeras tres instancias (*burma14*, *ulysses16* y *gr17*), esto se cumplió. Sin embargo, para las dos instancias restantes (*kroA100* y *pr264*), de un orden de magnitud mayor, la diferencia entre el exceso porcentual de ambos algoritmos fue mucho más pronunciada, favoreciendo al SRMOA, que redujo considerablemente el error porcentual desde las primeras 200 generaciones, manteniendo consistentemente esa reducción en la ejecución.

Lo anterior mencionado, da pie a confirmar nuestra segunda hipótesis: el desempeño del SRMOA muestra una mejora conforme crece la dimensión del problema. Lo que pudimos observar en las ejecuciones de las instancias de menor tamaño respecto de las dos más grandes, fue que el comportamiento del AGS no fue tan sensible como el SRMOA a tamaños mayores de las instancias del TSP, mientras que el SRMOA se vio afectado con una convergencia más temprana que la de su adversario. Desafortunadamente, y contrario a lo que nos hubiese gustado ver, ninguno de los dos logró llegar en estos casos a soluciones óptimas dentro del límite de generaciones que establecimos.

También señalamos dentro de las hipótesis el supuesto de una implementación más sencilla del SRMOA que la del AGS. Si bien, no erramos al suponer eso, subestimamos el tiempo y el esfuerzo necesario para entender los conceptos propuestos en el algoritmo básico que estudiamos. Afortunadamente, una vez planteada nuestra propia versión de algoritmo, poner manos a la obra y concluir nuestra construcción, fue mucho más breve y ameno. Cabe mencionar que la estructura de la implementación del primer proyecto -original del ayudante de Laboratorio, Roberto Monroy-, sirvió de guía para la organización del programa del SRMOA, inspirándonos confianza para considerar ambos materiales, candidatos válidos para la comparativa que queríamos analizar en este trabajo final.

Una de las cosas que no podemos dejar sin mención es el elevado costo en tiempo que exige el SRMOA en sus ejecuciones. Podemos observar en la tabla de datos los tiempos de ambos algoritmos, que para instancias de mayor tamaño se dispara el tiempo requerido por el SRMOA paa completar las generaciones. Aunque, en clase ya habíamos establecido que la complejidad en tiempo no era la más atractiva, el último resultado en tiempo del SRMOA antes del total (*pr264*), que fue seis veces mayor al anterior (*kroA100*), nos convenció de que el AGS es un adversario demasiado fuerte para el SRMOA. En términos de tiempo, declaramos que esta contienda la gana la propuesta más antigua.

Finalmente, consideramos que el *Algortimo de Optimización Inspirado en Magnetismo con Repulsión de Corto Alcance* que implementamos, demostró ser de utilidad en la optimización de soluciones para instancias del *Problema del Agente Viajero* y hemos decidido que es una buena opción para aplicaciones con estas características; y una competente, tomando como referencia al *Algoritmo Genético Simple*. Sólo queda decir que realmente fue interesante trabajar con una propuesta tan reciente y bastante gratificante construir con ideas nuestras, producto de lo que aprendimos en el curso; ideas que creímos demasiado aventuradas al momento del desarrollo de

esta versión del algoritmo, pero que resultaron funcionar mejor de lo que esperábamos, incluso de lo que imaginábamos.

6. Referencias

1. N.M.H. Tarayani, T.M.R. Akbarzadeh, Magnetic optimization algorithm, a new synthesis, in: IEEE World Congress on Computational Intelligence, Hong Kong, 2008.

2. M.M. Ismail, M.I. Zakaria, A.F.Z. Abidin, J.M. Juliani, A. Lit, S. Mirjalili, N.A. Nordin, M.F.M. Saaid, Magnetic optimization algorithm approach for travelling salesman problem, in: World Academy of Science, Engineering and Technology, 2012.

3. N.M.H. Tarayani, T.M.R. Akbarzadeh, Magnetic-inspired optimization algorithm: Operators and structures, in: Swarm and Evolutionary Computation, 2012.

4. K.P. Wang,L. Huang, C.G., et.al. Particle swarm optimization for traveling salesman problem, in: International Conference on Machine Learning and Cybernetics 3, 2003.