Optimización de sincronización de semáforos en barrio Centro

Sofía Bordagorry
Instituto de Computación
Facultad de Ingeniería, UdelaR
Montevideo, Uruguay
sofia.bordagorry@fing.edu.uy

Juan Pacheco
Instituto de Computación
Facultad de Ingeniería, UdelaR
Montevideo, Uruguay
juan.pacheco@fing.edu.uy

***Abstract*—Se plantea un algoritmo evolutivo para el problema de sincronización de semáforos en el barrio Centro de Montevideo, explorando los conceptos teóricos que son abarcados por este para luego poder hacer una comparación con un algoritmo de tipo greedy. Finalmente, se presentan conclusiones hechas a partir del análisis de los resultados extraídos.**

***Keywords*—algoritmo genético; optimización; semáforo; fase; offset**

I. Introducción

Uno de los principales desafíos en las ciudades modernas es gestionar eficientemente el tráfico en las intersecciones con semáforos. Se busca reducir la congestión del tráfico y acordar los tiempos de espera para los conductores.

Obtener soluciones a este problema no es una tarea sencilla si se utilizan algoritmos deterministas convencionales ya que, por lo general, no obtienen buenas soluciones en un tiempo razonable. Por esto se propone una solución al problema usando un algoritmo evolutivo.

II. Descripción del Problema

El problema consiste en definir las duraciones de las fases y los offsets de los semáforos del Centro de Montevideo, teniendo como objetivo la optimización del flujo vehicular. En definitiva, lo que se busca es poder maximizar la velocidad promedio de los vehículos que recorren la zona definida a partir de la aplicación de un algoritmo genético.

A. Parámetros de entrada

Los parámetros de entrada del problema son los siguientes:

- La instancia inicial, cuya representación se explica más adelante.
- La cantidad de generaciones.
- La población de cada generación, la cual debe ser mayor o igual a tres.

B. Descripción de escenario

El escenario del algoritmo evolutivo se basa en la representación y simulación de la red de semáforos en el barrio Centro de Montevideo utilizando herramientas como SUMO y Netedit. SUMO (Simulation of Urban MObility) se emplea como el software principal de simulación de tráfico, mientras que Netedit se utiliza para editar y configurar la red de semáforos.

La información para la creación de la red se obtiene de fuentes como el Sistema de Información Geográfica de la Intendencia de Montevideo y OpenStreetMap. Estos datos proporcionan detalles sobre la ubicación y configuración de las intersecciones y semáforos en el Centro de Montevideo. Se incorpora información geográfica real del Centro de Montevideo para garantizar la precisión del modelo. Esta información incluye datos sobre la disposición de las calles, la topología de las intersecciones y la ubicación de los semáforos.

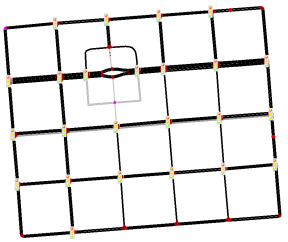
C. Descripción de las instancias

Las instancias del problema se derivan de una representación abstracta de la red de calles y semáforos en el Centro de Montevideo. Esta representación utiliza un modelo de red, donde los nodos representan intersecciones de calles y las aristas indican segmentos de calles conectadas. Cada instancia implica la configuración específica de los semáforos en la red, incluyendo las duraciones de las fases y los offsets.

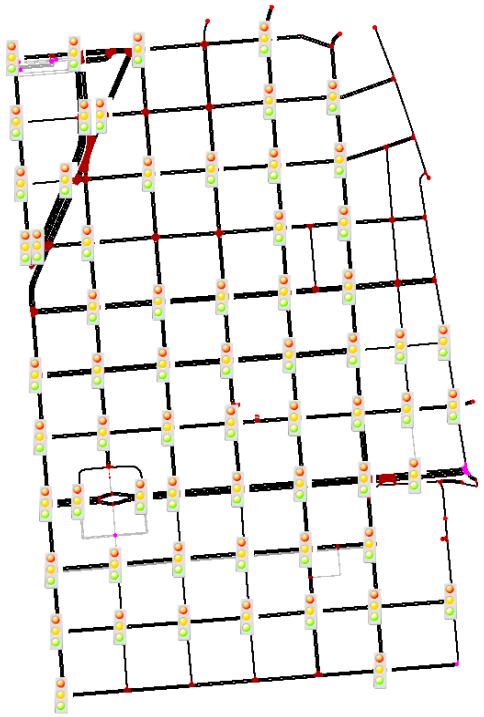
Para abordar esta optimización, se emplea un enfoque basado en algoritmos evolutivos, específicamente un algoritmo genético. La variación en las duraciones de las fases y los offsets de los semáforos se convierte en genes en el proceso evolutivo. Este enfoque busca encontrar soluciones que maximicen la velocidad promedio de los vehículos que atraviesan la zona definida en el Centro de Montevideo.

Para nuestro proyecto elegimos tres instancias de distintos tamaños que serán evaluadas usando el algoritmo genético.

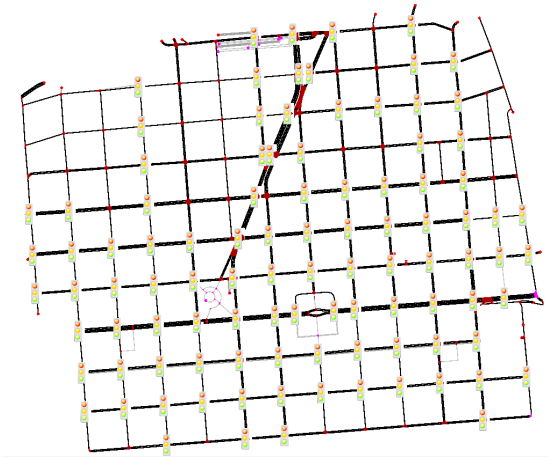
- Instancia pequeña: Red que abarca una pequeña porción del barrio Centro, aproximadamente un cuarto de este, y tiene 24 semáforos.



- Instancia intermedia: Red que abarca la mitad del Centro y tiene 64 semáforos.



- Instancia grande: Red que abarca todo el barrio Centro y tiene 100 semáforos.



III. Representación abstracta

A. Representación de la Instancia

Para la representación del problema usamos una red que muestra las intersecciones de las calles del Centro (o de algunas calles, para las instancias más pequeñas) como nodos, las calles como aristas, y en determinados nodos se tienen los semáforos a configurar.

B. Representación de Soluciones

Para representar las soluciones, se tienen vectores cuyo tamaño es la cantidad de semáforos multiplicada por tres ya que cada semáforo tiene tres variables, la duración de la fase verde de la calle transversal, la duración de la fase verde de la calle principal y el offset. Cada vector solución está ordenado de forma que para el semáforo número *i* se cumple que en la posición *i* se encuentra la duración de la fase 1, en *i+1* la duración de la fase 2 y en *i+2* el offset.

Cabe aclarar que para simplificar un poco el problema, asumimos que todas los semáforos del Centro tienen únicamente una fase de verde para la calle transversal y otra fase de verde para la calle principal, pero por investigaciones que realizamos sabemos que esto no se cumple siempre en la realidad.

IV. Estrategia de resolución

En esta sección se muestran los conceptos teóricos que se usaron para poder llegar a la solución implementada.

A. Algoritmo utilizado

Se implementó un algoritmo genético el cual en cada iteración evalúa la generación actual y la ordena de manera que los individuos que tiene una mejor función de fitness están al principio de un arreglo. Luego, se aplican los operadores de selección, cruzamiento y mutación que serán detallados a continuación. Al finalizar además se agregan a la nueva generación los dos mejores individuos de la generación actual.

```
procedimiento AlgoritmoGenetico():

begin

   inicializar la población actual

    para cada generación en rango(maxGeneraciones):

          si tamaño(poblacionActual) > 2:

                    rankear la población actual

          iniciar poblacionHijos como vacío

          inicializar padresSeleccionados como la mitad de la
          población rankeada

          para cada indicePadre en padresSeleccionados

          seleccionar al padre de índice actual y al que lo
          sigue

          aplicar cruzamiento entre los padres seleccionados

          agregar los hijos obtenidos a la población de hijos

          para cada solución en poblacionHijos:

                    aplicar mutacion

          agregar a la población de hijos al mejor de la
          población actual rankeada

          agregar a la población de hijos al segundo mejor de la
          población actual rankeada

          actualizar la población actual como la población de
          hijos

end
```

B. Función de Fitness

La función de fitness que se quiere maximizar es $f = v_p$, siendo v_p la velocidad promedio resultante de la configuración de semáforos de una determinada solución.

C. Operadores evolutivos

1) Selección: La elección de los padres se realiza partiendo la población en dos partes luego de ser evaluada y rankeada, la primera parte serán los individuos que hayan tenido un mejor desempeño en la función de fitness y por lo tanto serán los que se usarán para el cruzamiento. Las "parejas" del cruzamiento se eligen de forma consecutiva (es decir, el individuo *i* se va a cruzar con el individuo *i+1*).

2) Cruzamiento: Dados dos padres el cruzamiento (que tiene una cierta probabilidad de suceder) empieza seleccionando un punto de corte aleatorio dentro de la longitud del vector de variables que tienen los padres, para luego verificar que este corte es válido y no se ha realizado en medio de la configuración de un semáforo (por ejemplo, que corte en la duración de la fase de la calle transversal de un semáforo y el offset se pierda en la otra parte del corte). Si el corte es válido, sea este en el punto *i*, se genera un hijo que del gen 0 al gen i va a tener los mismos valores que el primer padre y que del gen *i+1* al último gen va a tener los valores del segundo padre, y otro hijo que será análogo pero del gen 0 al *i* tendrá los valores del segundo padre y del *i+1* al último los valores del primer padre.

3) Mutación: Se itera a través de las variables de un individuo y para cada una de ellas se verifica si debe ser mutada basándose en la probabilidad de mutación. Si se selecciona para mutación, entonces su valor se verá afectado con la suma de un número aleatorio dentro de un cierto rango de permutación. Además se asegura que esté dentro de los límites establecidos y en caso de ser necesario, se ajustan los valores de las otras variables del semáforo que se cambió para que siga siendo coherente (si se cambia la duración de una fase, se ajusta la de la otra con respecto al cambio).

Debido a nuestra implementación, para una población de *n* individuos, la mitad de ellos (*n/2*) pasarán a ser padres y por la forma en la que realizamos los pares que van a cruzarse y en la que se hace el cruzamiento, se generarían n-2 individuos (iría disminuyendo la población en cada generación). Por ello, al final de cada iteración también agregamos a los dos mejores de la generación anterior lo cual también ayuda a que no se pierdan los mejores valores.

D. Configuración paramétrica

En la búsqueda de una configuración para lograr conseguir una buena variabilidad de soluciones por generación, para llegar a la configuración óptima realizamos testeos de varias configuraciones distintas, variando los distintos parámetros de tamaño de la población, la probabilidad de cruzamiento, la probabilidad de mutación y tomando una cantidad de generaciones más reducida (30). Estos testeos fueron realizados sobre instancias más pequeñas. Comenzamos con configuraciones como la siguiente: 20, 0.5, 0.01 respectivamente, pero con probabilidades tan bajas de cruzamiento y mutación la mejora de la velocidad promedio era extremadamente pequeña en cada generación por lo que comenzamos a testear configuraciones con probabilidades más altas y concluimos que la siguiente configuración fue la mejor obtenida y por lo tanto fue la que se utilizó para ejecutar el algoritmo:

1) Tamaño de la población: 30

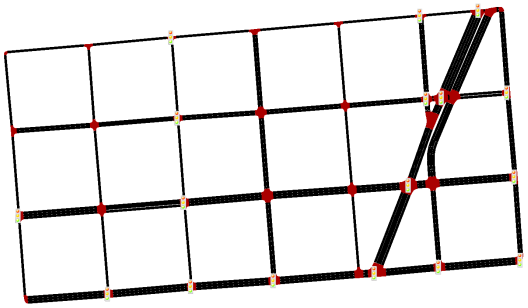
2) Probabilidad de cruzamiento: 75% (0.75)

3) Probabilidad de mutación: 8% (0.08)

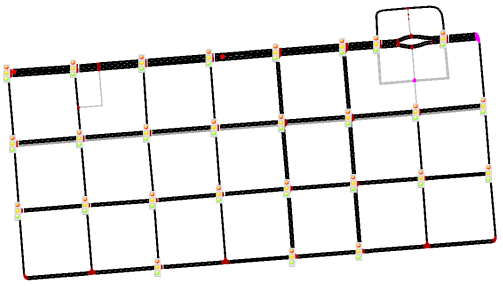
El problema que nos encontramos al tomar probabilidades más altas era que en varias generaciones habían muchos individuos que empeoraban y por lo tanto la mejora de la velocidad promedio de generación en generación era muy pequeña. Pero con la configuración mencionada anteriormente conseguimos el mejor balance entre la variedad de individuos por generación y mejora de la velocidad promedio.

Las instancias utilizadas para lograr la mejor configuracion fueron las siguientes, todas ellas son porciones pequeñas del Centro:

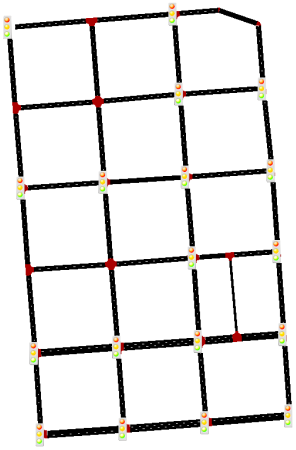
- InstanciaConf1: Red de 17 semáforos.



- InstanciaConf2: Red de 17 semáforos



- InstanciaConf3: Red de 18 semáforos.



A continuación se presenta una tabla que muestra algunas de las combinaciones usadas junto a las mejores velocidades que se pudieron alcanzar para cada una de las instancias que se utilizaron para los testeos.

InstanciaConf1			
Individuos por Generación	Probabilidad de Cruzamiento	Probabilidad de Mutación	Mejor fitness
20	0.5	0.01	6.65 m/s
10	0.75	0.075	6.82 m/s
10	0.9	0.08	6.83 m/s
20	0.66	0.08	6.91 m/s
30	0.75	0.08	6.92 m/s

InstanciaConf2			
Individuos por Generación	Probabilidad de Cruzamiento	Probabilidad de Mutación	Mejor fitness
20	0.5	0.01	5.84 m/s
10	0.75	0.075	5.62 m/s
10	0.9	0.08	6.01 m/s
20	0.66	0.08	5.90 m/s
30	0.75	0.08	6.02 m/s

InstanciaConf3			
Individuos por Generación	Probabilidad de Cruzamiento	Probabilidad de Mutación	Mejor fitness
20	0.5	0.01	7.03 m/s
10	0.75	0.075	6.80 m/s
10	0.9	0.08	6.76 m/s
20	0.66	0.08	6.98 m/s
30	0.75	0.08	7.31 m/s

V. Implementación del Algoritmo

A. Generación de la población inicial

La generación de las poblaciones iniciales fue realizada mediante un script en lenguaje Python, *"generador.py"*, que lee un archivo *"semaforos.txt"*, el cual contiene la representación de cada semáforo en un formato específico que es reconocido por el script y a partir de eso genera otro archivo de texto *"soluciones.txt"*.

B. Uso de SUMO

Para evaluar las configuraciones que genera el algoritmo genético realizamos simulaciones en las instancias mencionadas. Una simulación consta de una red, un conjunto de rutas de vehículos y un tiempo máximo de simulación. En nuestro caso el tiempo de simulación es 3600 segundos (1 hora).

Las rutas de los vehículos fueron generadas utilizando una herramienta de SUMO, llamada "randomTrips.py". Este archivo python genera un conjunto de rutas aleatorias dentro de una red, pero para que la simulación sea un poco más realista configuramos la generación de estas rutas de tal manera que las calles principales sean las que tengan más tráfico, es decir, que más vehículos pasen por las calles principales.

Además, basándonos en información recaudada sobre el conteo vehicular de las calles del centro de Montevideo, tenemos que:

Tamaño de la instancia	Cantidad de vehículos que circulan durante la simulación
Pequeña	100
Intermedia	250
Grande	500

Estas simulaciones son llamadas por un script de Python que las usa para calcular la función de fitness.

C. Cálculo de la velocidad

Al ser necesario usar las simulaciones para poder calcular la velocidad promedio y así tener los valores de la función de fitness, se usó un script *"velocidadPromedio.py"* que dada la simulación calculaba este valor. Para poder implementar esto se utiliza la librería traCI que facilita la interacción con SUMO mediante Python, permitiendo la ejecución de simulaciones y la recolección de información de las mismas.

D. Ejecución del algoritmo

El algoritmo se implementó en el lenguaje Java, usando el framework jMetal.

Para poder pasar del archivo *"soluciones.txt"* a los individuos en el algoritmo genético, implementamos una clase llamada "JMetalSUMOIntegration" que lee este archivo y convierte cada línea (que corresponde a un semáforo) en un objeto de clase "Semaforo", los cuales luego son utilizados para crear población inicial, que es una lista de IntegerSolution.

Cada generación de la población está compuesta por un conjunto de IntegerSolution, se utilizan este tipo de soluciones ya que son las que se adaptan mejor a nuestro problema debido a que todas las variables (offset y fases) son números enteros.

Con los datos mencionados, se invoca al algoritmo genético para que se ejecute durante un número específico de generaciones y finalmente retorna la mejor solución encontrada durante su ejecución.

VI. Análisis de la solución propuesta

Se realizaron 30 ejecuciones de las tres instancias de nuestro problema(Pequeña, Intermedia, y Grande).

A. Comparación con algoritmo greedy

Se implementó un algoritmo greedy para poder usarlo como punto de comparación con el algoritmo genético presentado. La forma en la que funciona este algoritmo es que, a partir de una configuración de semáforos inicial y durante una cantidad definida de iteraciones (cantIteracionesSemaforos) se ajustan los distintos parámetros de cada semáforo en diferentes direcciones (aumentando o disminuyendo en 1 la duración de las fases y el offset). Tras cada ajuste se evalúa la velocidad promedio utilizando el archivo "*velocidadPromedio.py*", el cual como mencionamos anteriormente ejecuta una simulación de SUMO para poder realizar esta evaluación. Luego de realizar el ajuste para cada semáforo se almacena en una variable la configuración con la que se obtuvo la mayor velocidad promedio y se vuelve a iterar.

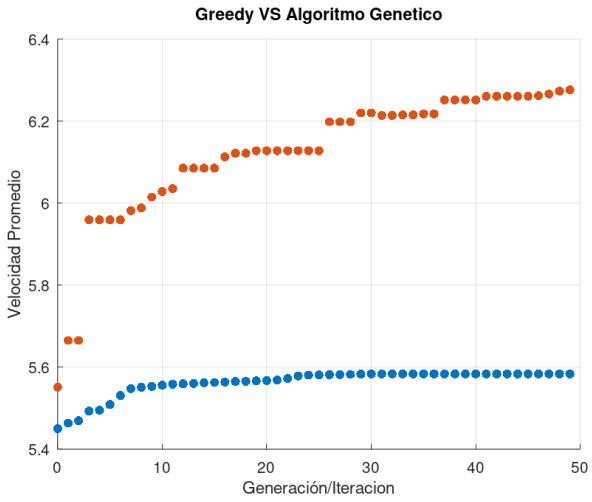
Se adjunta un pseudocódigo para que sea más sencillo de comprender:

```
procedimiento greedy():

begin

    crear la configuración inicial

    para cada iteración

         para cada semáforo

                  modificar la primera fase sumando uno y
                  evaluar la configuración con la modificación

                  modificar la primera fase restando uno y
                  evaluar la configuración con la modificación

                  modificar la segunda fase sumando uno y
                  evaluar la configuración con la modificación

                  modificar la segunda fase restando uno y
                  evaluar la configuración con la modificación

                  modificar el offset sumando uno y evaluar la
                  configuración con la modificación

                  modificar el offset restando uno y evaluar la
                  configuración con la modificación

         end For

         guardar la configuración que haya mejorado la
         velocidad promedio

     end For

end
```

Tener que calcular la velocidad promedio para cada cambio de semáforo hace que este algoritmo no sea para nada eficiente, ya que para una instancia cualquiera la cantidad de veces que se tendría que ejecutar está dada por la fórmula: $cant_{iter\ totales} = 6 * cant_{semáforos} * cant_{iter\ sobre\ semáforos}$, y, por ejemplo, para una instancia en la que se quiere iterar 10 veces sobre 100 semáforos terminaría invocando 6000 veces el script de Python que se usa para calcular la velocidad promedio de la red a partir de una simulación, el cual, a pesar de estar implementado de forma sencilla, implica grandes tiempos de respuesta. Luego, este algoritmo no es óptimo.

Es por ello que para realizar la comparación entre el algoritmo greedy planteado y el algoritmo genético decidimos realizar la evaluación de ambos algoritmos sobre la instancia pequeña (24 semáforos) con 50 generaciones/iteraciones y comparar los resultados obtenidos aplicando ambos algoritmos sobre dicha instancia.

Al aplicar el algoritmo greedy pudimos ver que efectivamente la función evaluada (velocidad promedio usando la simulación de SUMO) mejora con cada iteración y, como se ve en la gráfica, converge en muy pocas iteraciones a una posible solución. Aún así, esta solución no es la óptima ya que el valor de la velocidad promedio es considerablemente menor que la solución que obtiene el algoritmo genético presentado.



La mejor velocidad obtenida con el algoritmo greedy fue 5.584 y la mejor velocidad obtenida con el algoritmo genético fue 6.230 tomando en cuenta que se ejecutaron 50 iteraciones/generaciones de cada uno.

Además, como se mencionó anteriormente, el tiempo de ejecución del algoritmo greedy es bastante más extenso que el tiempo de ejecución del algoritmo greedy. En este caso, el algoritmo genético tardó aproximadamente 1 hora y 45 minutos en realizar las 50 generaciones y el algoritmo greedy tardó 12 horas en ejecutar 50 iteraciones.

1. Criterio de significancia estadística

Con los datos recopilados a partir de 30 ejecuciones de la instancia de 24 semáforos, se aplicó el criterio de significancia estadística para comprobar que efectivamente el algoritmo evolutivo es mejor que el algoritmo greedy.

Dicho criterio implica que el algoritmo genético es mejor que el algoritmo greedy si se cumple:

$$|fAVG(AE) - fAVG(G)| > max\,(std(fAE),\ std(fG))$$

siendo:

- $fAVG(AE)$: Valor promedio (media) de las velocidades obtenidas por el algoritmo genético, con respecto a las ejecuciones realizadas.
- $fAVG(G)$: Media de las velocidades obtenidas por el algoritmo greedy.
- $std(fAE)$: Desviación estándar de los resultados obtenidos por el algoritmo genético en este caso.
- $std(fG)$: Desviación estándar de los resultados obtenidos por el algoritmo greedy. Al ser greedy y por lo tanto determinista, en este caso el valor es 0.

Con los valores obtenidos a partir de nuestras ejecuciones, las variables de la fórmula presentada pasan a tener los siguientes valores:

$$|6.230 - 5.584| > max\,(0.046, 0)$$

$$|0.646| > 0.046$$

Esta última desigualdad es la que confirma que se cumple el criterio de significancia estadística y por lo tanto el algoritmo evolutivo es mejor que el algoritmo greedy.

2. Test de Mann-Whitney

Para saber si existen diferencias significativas entre las medianas de los dos conjuntos (las velocidades obtenidas por el algoritmo genético y las velocidades obtenidas por el greedy), usamos un test de Mann-Whitney.

Los resultados que obtuvimos del test fueron;

- Un z-score de -6.6456, este valor indica cuántas desviaciones estándar está la diferencia entre los dos conjuntos de muestra, el valor obtenido indica una gran diferencia entre los conjuntos.
- Por otra parte, el p-valor obtenido fue muy pequeño, menor a 0.00001, lo cual significa que hay una probabilidad muy baja de que la diferencia entre los conjuntos sea resultado del azar y por lo tanto es altamente significativa estadísticamente hablando.

B. Test de normalidad para las instancias evaluadas

Los datos obtenidos de 30 ejecuciones de cada una de las instancias trabajadas fueron usados para realizarle a cada conjunto de datos un test de normalidad de Kolmogorov-Smirnov, que arrojaron los siguientes resultados.

1. Instancia pequeña

- Media: 6.23048
- Mediana: 6.25099
- Desviación estándar: 0.047337

Además, el p-valor obtenido fue 0.14428. Teniendo en cuenta que nuestro nivel de significancia fue de 0.05 y que el p-valor obtenido es mayor a este número, entonces los datos no difieren significativamente de una distribución normal según este test.

2. Instancia intermedia

- Media: 5.93824
- Mediana: 5.949579
- Desviación estándar: 0.063245

En este caso, el p-valor fue 0.86084. Por ser mayor a 0.05, se aplica el mismo criterio que para la pequeña y por lo tanto los datos se asemejan a una distribución normal.

3. Instancia grande

- Media: 5.66764
- Mediana: 5.672137
- Desviación estándar: 0.030872

El p-valor obtenido fue 0.93965. Nuevamente, podemos afirmar que los datos no difieren significativamente de una distribución normal.

C. Comparando con una solución real

Como último comentario acerca de la solución presentada, gracias a la investigación realizada sobre los semáforos del centro, obtuvimos que el valor actual de la velocidad promedio de los vehículos que circulan el centro (importante

destacar que durante el horario vespertino) es de 5.14 m/s utilizando la misma simulación de 500 vehículos que se utilizó para la instancia grande.

También debido a la investigación realizada se sabe que la velocidad promedio de tránsito de vehículos de las calles del centro está entre 20 y 30 km/h y nuestro algoritmo llega a soluciones que se encuentran dentro de ese rango. Esta correspondencia entre las velocidades reales y las generadas por nuestro algoritmo indican su efectividad para generar resultados coherentes.

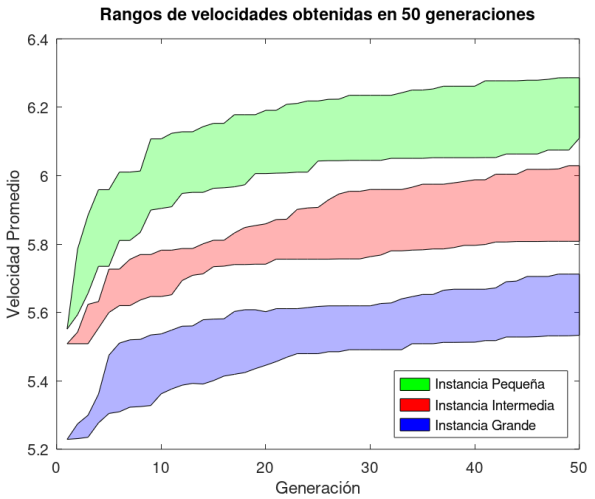
VII. Conclusión

Se pudo implementar un algoritmo evolutivo que resuelve el problema presentado con respecto a velocidades, a continuación se presentan más detalladamente los resultados obtenidos.

A. Resultados numéricos

Para cada una de las instancias se realizaron 30 ejecuciones de 50 generaciones cada una de ellas.

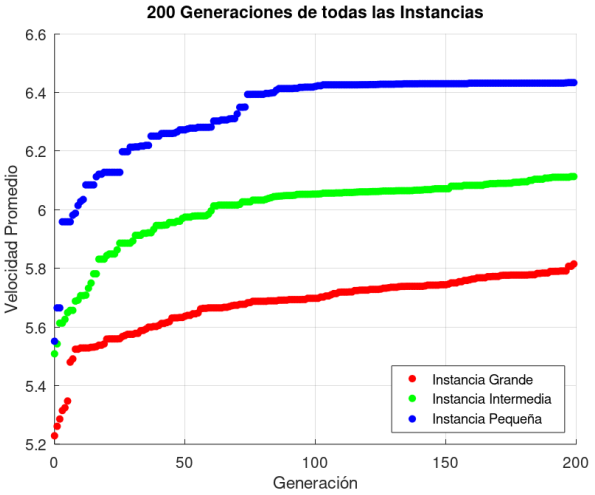
En la siguiente gráfica se puede observar el rango de velocidades obtenidas de cada generación al ejecutar 30 veces el algoritmo genético usando cada una de las Instancias.



Adjuntamos una tabla con datos relevantes con respecto a los valores de fitness obtenidos para cada instancia durante las 30 ejecuciones de 50 generaciones.

Instancia	Mejor velocidad alcanzada	Promedio de las velocidades alcanzadas	Desviación estándar
Pequeña	6.28196 m/s	6.23048 m/s	0.047337
Intermedia	6.02914 m/s	5.93824 m/s	0.063245
Grande	5.71213 m/s	5.66764 m/s	0.030872

Además, para cada instancia también se realizó una ejecución de 200 generaciones para poder determinar su convergencia. Cabe mencionar que las ejecuciones duraron 7 horas para la instancia Pequeña, 10 horas la instancia Intermedia y 20 horas para la instancia Grande. Se utilizaron 2 PC diferentes para las ejecuciones.



Viendo la gráfica y analizando el comportamiento de los puntos en cada generación, se llegaron a las siguientes conclusiones:

La instancia pequeña, compuesta por 24 semáforos, demuestra una rápida convergencia en comparación con sus contrapartes más grandes. Esta eficiencia se atribuye al tamaño reducido de la red, que representa sólo una fracción de los semáforos presentes en las instancias de mayor tamaño. La menor complejidad y el número limitado de semáforos facilitan iteraciones más ágiles y eficientes en la búsqueda de soluciones óptimas.

En cuanto a las instancias intermedia (64 semáforos) y grande (100 semáforos), observamos que muestran un avance más pausado en la búsqueda de mejoras en la velocidad promedio. Aunque se espera que converjan eventualmente a un valor óptimo debido al diseño del algoritmo, la complejidad y la mayor cantidad de semáforos provocan cambios menos notorios en cada iteración por lo que el proceso de hallar el valor óptimo para la función de fitness es computacionalmente mucho más costoso.

B. Puntos a mejorar

Debido a los recursos utilizados, ejecutar la instancia “grande” resultó muy difícil pues las computadoras usadas no tenían los requerimientos óptimos para esto, generando problemas como que por ejemplo Python dejará de ejecutarse repentinamente, por lo cual no pudimos llegar a ver el valor de convergencia de la función de fitness para esta instancia.

References

[1] Sistema de Información Geográfica de Montevideo, https://sig.montevideo.gub.uy/, retr. 20/11/2023.

[2] SUMO User Documentation, https://sumo.dlr.de/docs/index.html, retr. 1/12/2023.

[3] Universidad de la República, Facultad de Ingeniería, Instituto de Computación, “Algoritmos Evolutivos aplicados a la sincronización de semáforos en el Corredor Garzón”, 2014, Acuña, Arreche, Nesmachnow.

[4] JMetal repository, https://github.com/jMetal/jMetal/tree/main, retr. 10/12/2023.

[5] Diapositivas del curso de Algoritmos Evolutivos de la Facultad de Ingeniería de UdelaR, https://eva.fing.edu.uy/course/view.php?id=1049, retr. 14/12/2023.

[6] Prueba de Kolmogorov-Smirnov , https://www.ibm.com/docs/es/spss-statistics/saas?topic=tests-one-sample-kolmogorov-smirnov-test, retr. 15/12/2023

[7] Velocidad promedio vehicular en las principales avenidas del centro, https://catalogodatos.gub.uy/dataset/intendencia-montevideo-velocidad-promedio-vehicular-en-las-principales-avenidas-de-montevideo, retr. 29/12/202