UNIVERSIDAD NACIONAL DE COLOMBIA

Proyecto Introducción a la Inteligencia Artificial
A-A*pex: Effcient Anytime Approximate Multi-Objective Search

Oscar Ivan Ulises Gutierrez Palacios, Juan Diego Gonzalez Layton, Santiago Botero Daza

Prof. Diego Gerardo Roldan Jiménez

1. Introducción

El algoritmo **A-A*pex** es una extensión del algoritmo **A*pex**, diseñado para resolver problemas de búsqueda multiobjetivo en grafos donde cada arista tiene múltiples costos asociados, como tiempo, distancia o riesgo. Su enfoque *anytime* le permite encontrar rápidamente soluciones aproximadas y refinarlas progresivamente hasta alcanzar una frontera de Pareto cercana a la óptima. Para mejorar su eficiencia, A-A*pex emplea estrategias avanzadas como la **fusión de nodos**, el uso de **heurísticas basadas en Dijkstra**, y un control dinámico del **factor de aproximación** ϵ. Además, en cada iteración, decide de manera inteligente si **reutilizar soluciones previas o reiniciar desde cero**, optimizando el balance entre tiempo de ejecución y calidad de la solución.

2. Objetivos

2.1. Objetivo General

Evaluar y ejecutar el algoritmo **A-A*pex**, sus variantes y compararlo respecto a la litera actual de algoritmos disponibles en el github del paper.

2.2. Objetivos Específicos

- Evaluar la efectividad del enfoque *anytime* del algoritmo, midiendo la calidad de la aproximación a la frontera de Pareto en función del tiempo de ejecución.
- Analizar el impacto de la reutilización de soluciones previas frente a reinicios completos en el rendimiento del algoritmo.
- Realizar experimentos en un conjunto de datos realista, como la red de calles de Nueva York, para probar la escalabilidad y eficiencia de **A-A*pex**.
- Implementar un método para calcular el **error de aproximación** a la frontera de Pareto y comparar el desempeño de **A-A*pex** con otros algoritmos.
- Aplicar el algoritmo en un caso de estudio práctico, como la optimización de rutas en el sistema de transporte masivo de Bogotá (**TRANSMILENIO**), adaptando la heurística y considerando objetivos mixtos (minimización y maximización).
- Desarrollar e integrar nuevas heurísticas basadas en el contexto del problema de transporte, incluyendo estimaciones de tiempo, distancia y flujo de pasajeros.

3. Experimentos

NY Road Network (Red de calles de Nueva York)

- Grafo con **264,000 nodos** y **730,000 aristas**.
- Objetivos usados:
  - t = tiempo de viaje
  - d = distancia

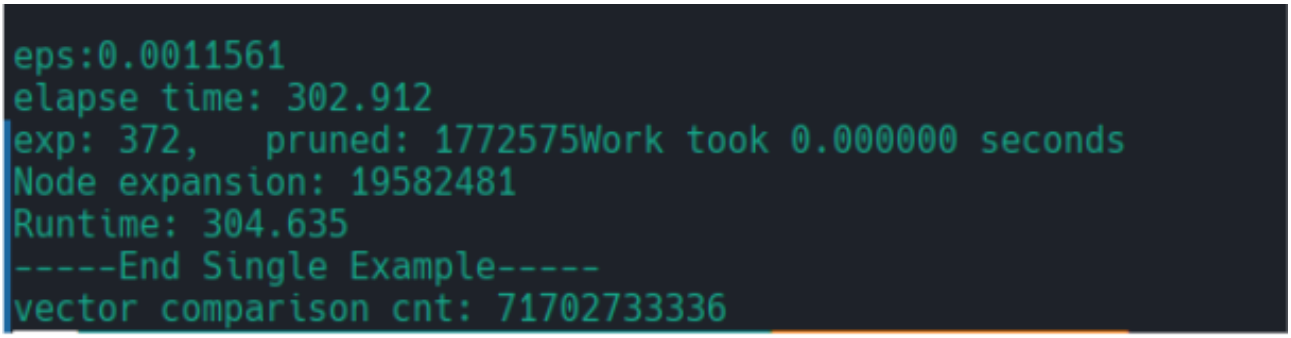
- m = costo económico
- l = número de calles usadas
- r = factor aleatorio de costos

Inicialmente para las pruebas del paper, se puso un límite de 5 minutos por instancia, y la tabla presentaba cuántos problemas podría resolver en ese determinado tiempo dada 50 simulaciones. Contaban con un computador M1 Pro con 32GB de RAM.

Para esta prueba se definen las 3 metas a minimizar:

- t = tiempo de viaje
- d = distancia
- m = costo económico

Dadas las limitaciones de hardware de nuestro equipo (Core i5 **8GB** de RAM), para esta tabla comparativa quitamos el límite de 5 minutos ya que dado ese límite ningún problema alcanzaba a completarse correctamente, como lo muestra la siguiente imagen.



Llegando alrededor de los 300 sg (5 minutos preestablecidos)

Para replicarlo dejaremos un tiempo libre inicialmente para estimar el límite según las capacidades actuales.

Parámetros: Eps inicial = 0.1, Decrementos: 1.5

Prueba inicial:

```
./mohs -m ../maps/NY-m.txt ../maps/NY-t.txt ../maps/NY-d.txt
--output output_tmp.txt -s 178689 -g 1476 --alg AnytimeApexRestart
--verbal 1 --logsolutions 1
```

Al correr esto sin límite de tiempo la memoria se ha llenado, y el sistema ha matado el proceso.

Vamos a hacer la prueba con dos objetivos:

- **t** = tiempo de viaje
- **m** = costo económico

Pruebas realizadas:

```
./mohs -m ../maps/NY-m.txt ../maps/NY-t.txt ../maps/NY-d.txt
--output output_tmp.txt -s 178689 -g 1476 --alg AnytimeApexRestart
--verbal 1 --logsolutions 1

./mohs -m ../maps/NY-m.txt ../maps/NY-t.txt ../maps/NY-d.txt
--output output_tmp.txt -s 178689 -g 1476 --alg AnytimeApexEnh
--verbal 1 --logsolutions 1

./mohs -m ../maps/NY-m.txt ../maps/NY-t.txt ../maps/NY-d.txt
--output output_tmp.txt -s 178689 -g 1476 --alg AnytimeApexHybrid
--verbal 1 --logsolutions 1

./mohs -m ../maps/NY-m.txt ../maps/NY-t.txt ../maps/NY-d.txt
--output output_tmp.txt -s 178689 -g 1476 --alg LTMOAR
--verbal 1 --logsolutions 1
```

Algoritmo	Nodos Exp	Comparaciones #	Runtime
LTMOA* n=1.5	8,673,631	36,517,567	8.60069
Anytime_Apex_R n=1.5	10,822,844	23,336,910	97.615
AnytimeApexEnh n=1.5	11,358,391	13,005,422,138	50.3718
Anytime_Apex_Hybrid n=1.5	14,274,206	6,426,565,285	39.0541
Anytime_Apex_R n=2	69,532,696	15,002,7877	61.3451
AnytimeApexEnh n=2	10,787,422	12,424,680,889	44.4538
Anytime_Apex_Hybrid n=2	33.9679	5,879,874,270	33.9679
Anytime_Apex_R n=4	36,038,026	77,738,260	30.9072
AnytimeApexEnh n=4	10,245,466	11,731,579,308	36.2983
Anytime_Apex_Hybrid n=4	11,354,255	5,599,038,488	26.9881
Anytime_Apex_R n=8	27,259,023	5,884,7142	27.0545
AnytimeApexEnh n=8	9,798,134	11,205,582,821	32.016
Anytime_Apex_Hybrid n=8	10,379,759	5,525,598,738	26.3617

Resultados de la prueba:

- **Anytime_Apex_R** (Reinicia la búsqueda en cada iteración):
  - Expande más nodos que las otras variantes.
  - Tiene el mayor tiempo de ejecución, especialmente en $n = 1{,}5$ (97.6s) y $n = 2$ (61.3s).
  - **Conclusión**: Es menos eficiente porque no reutiliza nodos de iteraciones previas.
- **AnytimeApexEnh** (Con dominancia mejorada):
  - Reduce la cantidad de comparaciones en valores grandes de n.
  - En $n = 8$, tiene el menor número de nodos expandidos (9.8M), pero sigue siendo más lento que **Anytime_Apex_Hybrid**.
- **Anytime_Apex_Hybrid** (Combina reinicio con reutilización de nodos):
  - Tiene el menor tiempo de ejecución en cada valor de n (ej. 26.3s en $n = 8$).
  - En $n = 8$, expande solo 10.3M nodos, lo que es 3 veces menos que **Anytime_Apex_R** con el mismo n.

Calculo de Error Frontera de Pareto:
En un problema de optimización multiobjetivo, se busca aproximar la frontera de Pareto exacta P^* con un conjunto de soluciones aproximadas A. Para medir la calidad de esta aproximación, se utiliza el **error de aproximación**, el cual evalúa la distancia relativa entre las soluciones aproximadas y la frontera exacta.

3.1. Factor de Dominancia

Dado un vector de costos de una solución aproximada $\mathbf{a}$ y un vector de costos de una solución óptima $\mathbf{p}$, se define el **factor de dominancia** $DF(\mathbf{a}, \mathbf{p})$ como:

$$DF(\mathbf{a}, \mathbf{p}) = \max \left\{ \max_i \left(\frac{a_i}{p_i} - 1 \right), 0 \right\} \tag{1}$$

donde a_i y p_i representan los valores de los objetivos en las soluciones aproximada y exacta, respectivamente. Este valor mide qué tan lejos está una solución aproximada de una solución óptima en términos relativos.

3.2. Cálculo del Error de Aproximación

El error de aproximación $e(A)$ se define como el peor caso entre todas las soluciones en la frontera de Pareto exacta. Es decir, para cada solución óptima $\mathbf{p} \in P^*$, se encuentra la solución aproximada $\mathbf{a} \in A$ que minimiza el factor de dominancia DF. Luego, el error de aproximación se obtiene como:

$$e(A) = \max_{\mathbf{p} \in P^*} \min_{\mathbf{a} \in A} DF(\mathbf{a}, \mathbf{p}) \tag{2}$$

Este cálculo representa la peor distancia relativa entre una solución en la frontera de Pareto exacta y su mejor aproximación en el conjunto A. Un valor de $e(A)$ cercano a 0 indica una mejor aproximación a la frontera de Pareto.

3.3. Interpretación

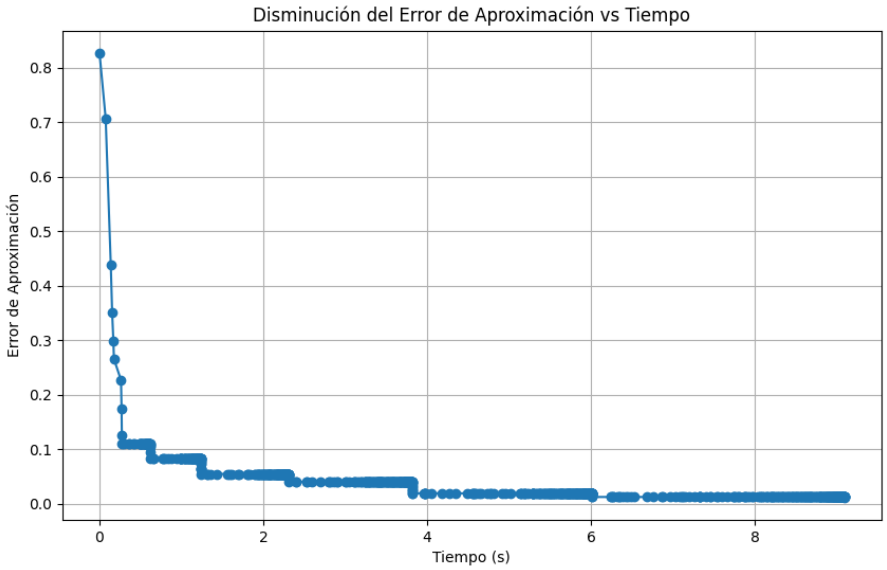
A medida que se encuentran mejores soluciones en A, el error de aproximación $e(A)$ debería disminuir con el tiempo. Esto se observa gráficamente al trazar $e(A)$ en función del tiempo, mostrando cómo la calidad de la aproximación mejora a lo largo de la ejecución del algoritmo.

El código del repositorio no presentaba ningún método para el calculo del error, pero si presentaba una opción para acceder a los registros del log del set de soluciones para cada ejecución

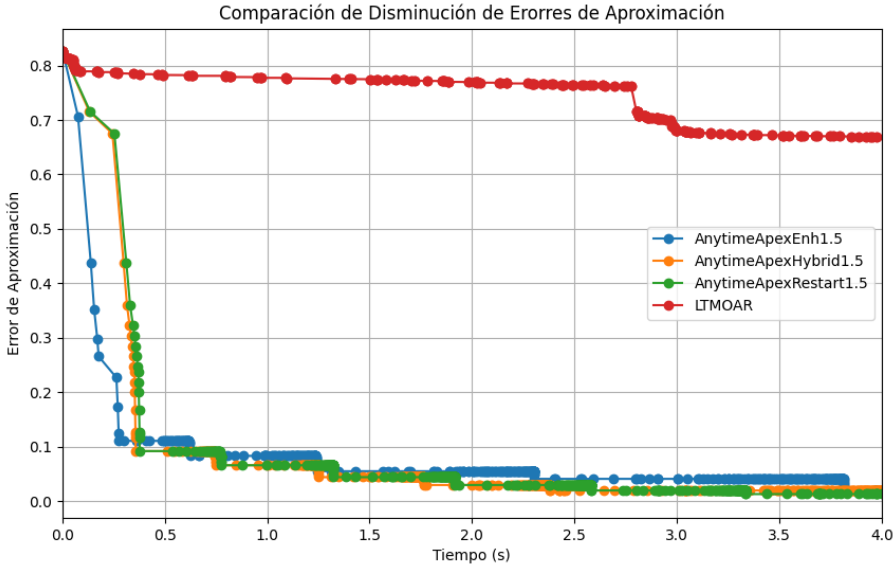
Con estos registros, se implemento en python las funciones respectivas para leerlos, y calcular el error en cada iteración.

```
Solution(vector<cost_t> _apex = {}, vector<cost_t> _cost = {}, double time=-1):
    apex(_apex), time_found(time) {
        if (_cost.size() == 0){
            cost = _apex;
        } else {
            cost = _cost ;
        }
    }
};
```

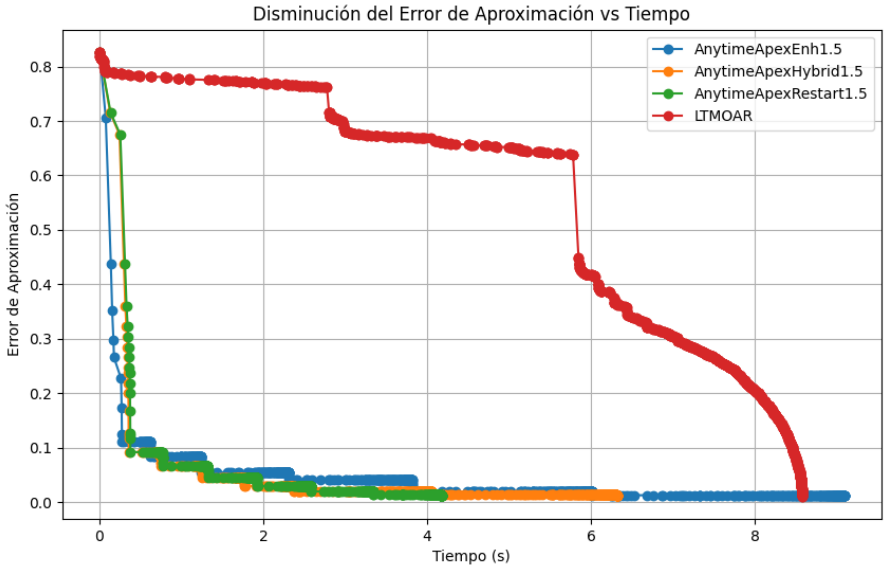
Donde el segundo vector $[n_1, ..., n_n]$ guarda los costos asociados a cada objetivo, y cada linea del registro representaba una solución encontrada en determinado tiempo



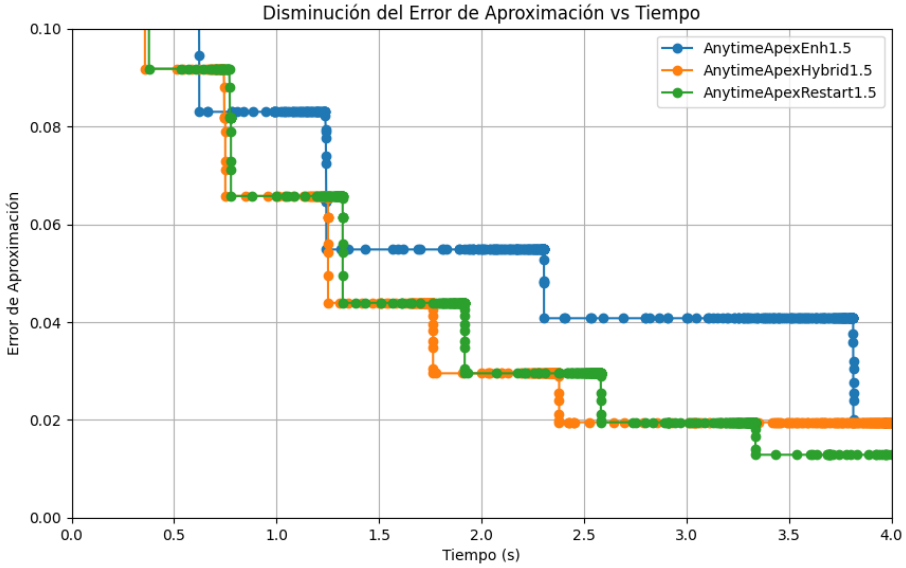
Ejemplo con el algoritmo AnytimeApexEnh



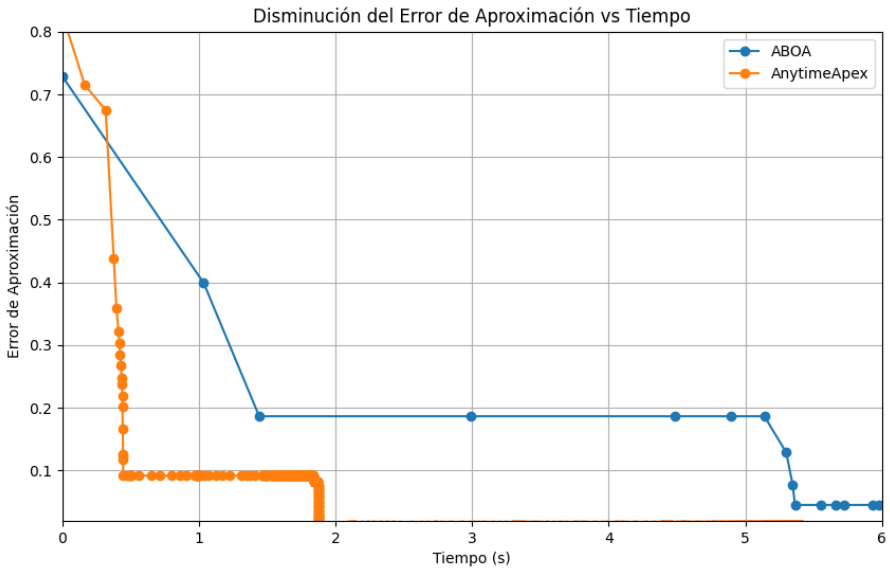
Comparación de error de A-a*pex modificado vs LTMOAR*, se puede apreciar el "*anytime*" y su rápida aproximación



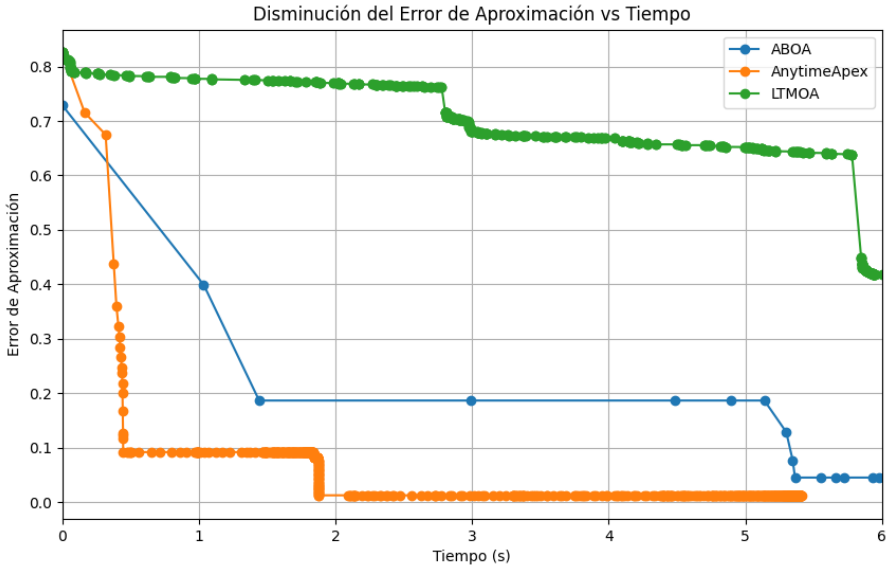
Después de determinado tiempo LTMOA* logra baja su error de aproximación a la frontera drásticamente



Detalle de refinamiento de las modificaciones de A-a*pex



A-a*pex tiende en esta caso de dos objetivos a una aproximación más drástica respecto al tiempo



Comparación con LMTOA*

Implementación propuesta

Después de haber estudiado el algoritmo, quisimos formular un problema en el que el algoritmo **A-A*pex** pudiera proporcionar una solución. Para ello, planteamos un problema de búsqueda de rutas en el sistema de transporte masivo de la ciudad de Bogotá, **TRANSMILENIO**. A diferencia de la implementación presentada por los autores, realizamos nuestra implementación en Python.

El problema se centra en encontrar una de las mejores rutas entre dos estaciones específicas. En el ejemplo presentado, las estaciones son **Portal Norte** y **Portal 20 de Julio**. Para esto, primero creamos un grafo que representa varias estaciones del sistema **TRANSMILENIO**, lo cual puede verse en el script `transmilenio_map.py`.

Posteriormente, planteamos algunos objetivos a minimizar, como el tiempo de viaje y la distancia recorrida en kilómetros. Además, introdujimos una variante al problema en la que se maximiza un objetivo: el número de pasajeros transportados. Para lograr esto, realizamos algunas modificaciones.

La primera modificación fue la inclusión de un vector de direcciones para indicar qué objetivos se deben minimizar y cuáles maximizar. Por ejemplo, si el vector de costos de una arista es $\mathbf{v} = (x, y, z)$ y queremos minimizar x y y, pero maximizar z, entonces el vector de direcciones sería $\mathbf{v} = (1, 1, -1)$. De esta manera, convertimos el problema mixto en uno de minimización.

También propusimos nuevas heurísticas para abordar este problema. Estas son:

```python
heuristics_min = [[dist_min.get(i, 0) * 5, dist_min.get(i, 0) * 1, 0] if i in dist_min else [0, 0, 0] for i in range(len(stations))]
heuristics_max = [[0, 0, 200] for _ in range(len(stations))]
```

La función `heuristics_min` primero calcula la distancia mínima en número de saltos desde cada nodo hasta el nodo objetivo (nodo 45). Para esto, implementamos un algoritmo BFS llamado `dist_min`, que almacena en un diccionario de Python el número mínimo de saltos desde nuestro nodo inicial hasta un nodo adyacente. Esto nos proporciona una medida de "distancia.en términos de saltos dentro del grafo. Luego, multiplicamos este valor por el costo mínimo por salto. Suponemos que el tiempo mínimo por salto es de 5 minutos (valor estimado) y que la distancia mínima por salto es de 1 km (valor estimado). Para el valor de z, lo dejamos en 0.

En cuanto al objetivo de pasajeros por minuto (que es de maximización), en `heuristics_min` no necesitamos estimar un mínimo, por lo que lo dejamos en 0. En cambio, para el valor a maximizar, establecemos un valor de 200, lo que indica que, en el mejor caso, el promedio de pasajeros por minuto en la ruta desde i hasta el objetivo será 200. Esta es una estimación optimista, ya que el valor real será menor o igual. Además, multiplicamos la heurística `heuristics_max` por -1 para que tenga sentido en el contexto del problema.

Finalmente, al implementar el algoritmo **A-A*pex** con estas nuevas heurísticas y estableciendo un valor inicial de $e = 0{,}2$, encontramos una solución a nuestro problema. Para ver esto en ejecución, asegúrese de tener tanto `main.py` como `transmilenio_map.py`. La solución obtenida es:

```
Soluciones encontradas en el frente de Pareto aproximado desde Portal Norte hasta Portal 20 de Julio:
Solución 1:
  Tiempo total: 128.8 min, Distancia total: 51 km, Promedio de pasajeros: 42.691787550117574 pasajeros/min
  Ruta: Portal Norte -> Toberin -> Calle 161 -> Mazuélen -> Calle 146 -> Calle 142 -> Alcalá -> Prado -> Calle 127 -> Biblioteca -> Portal 20 de Julio
```

En este ejemplo, se encontró una única solución no dominada. Si desea probar el algoritmo para otras rutas, tenga en cuenta que no siempre se garantizará una solución, ya que no todas las estaciones son adyacentes y el grafo no incluye todas las troncales de TRANSMILENIO. Por ello, es importante elegir dos estaciones que cumplan con las condiciones requeridas.

Referencias

Referencias

[1] Jennifer G. Walston. *Search Techniques for Multi-Objective Optimization of Mixed-Variable Systems Having Stochastic Responses*. Interim Report IR-07-014, Integrated Modeling Environment Project, IIASA, May 2007.

[2] H. Zhang, O. Salzman, A. Felner, C. Hernández Ulloa, S. Koenig. *A-A*pex: Efficient Anytime Approximate Multi-Objective Search*. Proceedings of the Seventeenth International Symposium on Combinatorial Search (SoCS 2024), 2024.

[3] C. Hernández, W. Yeoh, J. A. Baier, H. Zhang, L. Suazo, S. Koenig, O. Salzman. *Simple and Efficient Bi-Objective Search Algorithms via Fast Dominance Checks*. Artificial Intelligence, vol. 314, 2023, pp. 103807.

[4] H. Zhang, O. Salzman, T. K. S. Kumar, A. Felner, C. Hernández, S. Koenig. *Anytime Approximate Bi-Objective Search*. Proceedings of the Symposium on Combinatorial Search (SoCS), 2022.

[5] C. Hernández, W. Yeoh, J. A. Baier, H. Zhang, S. Koenig, O. Salzman. *Multi-Objective Search via Lazy and Efficient Dominance Checks*. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2023.

[6] H. Zhang, O. Salzman, S. Koenig. *Speeding Up Dominance Checks in Multi-Objective Search: New Techniques and Data Structures*. Proceedings of the Symposium on Combinatorial Search (SoCS), 2024.

[7] H. Zhang, O. Salzman, T. K. S. Kumar, A. Felner, C. Hernández, S. Koenig. *A*pex: Efficient Approximate Multi-Objective Search on Graphs*. Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2022.

[8] H. Zhang, O. Salzman, A. Felner, C. Hernández Ulloa, S. Koenig. *A-A*pex: Efficient Anytime Approximate Multi-Objective Search*. Proceedings of the Symposium on Combinatorial Search (SoCS), 2024.

[9] H. Zhang, O. Salzman, S. Koenig. *Bounded-Suboptimal Weight-Constrained Shortest-Path Search via Efficient Representation of Paths*. Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2024.