Algoritmo Evolutivo Paralelo para la Asignación de Rutas de un Servicio de Recolección de Residuos

1st Guillermo Toyos
5.139.879-9
Facultad de Inegniería
Montevideo, Uruguay
guillermo.toyos@fing.edu.uy

2nd Federico Vallcorba
5.102.078-4
Facultad de Ingeniería
Montevideo, Uruguay
federico.vallcorba@fing.edu.uy

***Resumen*—El artículo a continuación trata el problema de asignación de rutas para un servicio de recolección de residuos, así como el uso de algoritmos evolutivos para resolver dicho problema. Se busca la configuración paramétrica que brinda mejores resultados, para luego comparar el rendimiento del algoritmo implementado contra un algoritmo ávido que resuelve el problema. Si bien el tiempo de ejecución del algoritmo evolutivo es por varios ordenes mayor que al del ávido, se observa un porcentaje de mejora en la calidad de las soluciones de más del 50 % para instancias grandes del problema.**

***Index Terms*—Algoritmos Evolutivos Paralelos, Smart Cities, Administración de Residuos, VRP**

I. Introducción

En un mundo plagado de basura, donde prácticamente todo lo que nos rodea tarde o temprano se transformará en un residuo, resulta crucial disponer de un proceso de recolección de basura exitoso. Además del reciclaje y reuso de los desechos, que son fundamentales para reducir la presión sobre el sistema de recolección (más del 30 % de los residuos que se generan en Montevideo se pueden reciclar), es de vital importancia evitar al máximo la acumulación de basura en los centros poblados.

En Montevideo los hogares generan unas 1200 toneladas de basura a diario. Es fundamental disponer de un sistema de recolección de residuos que permita recolectarla de forma correcta, cuidando la salud de todos los ciudadanos. Para esto, la Intendencia de Montevideo dispone de una flota de camiones que trabaja las 24 horas, y más de 13000 contenedores de residuos. [3]

Resulta relevante minimizar la distancia y tiempo que recorren los camiones de basura diariamente con el afán de reducir las emisiones de CO_2, ya que la flota de camiones de Montevideo, como en gran parte del mundo, utiliza combustibles fósiles. Además de reducir el impacto ambiental, buscar rutas eficientes reduce la cantidad de camiones que se necesitan para proveer un buen servicio y la cantidad de combustible que debe comprarse, así reduciendo los costos operativos.

Asignar rutas a los camiones de basura puede entenderse como una instancia del Vehicle Routing Problem [10], donde se busca minimizar ciertos parámetros, como la distancia, debiéndose cumplir ciertas condiciones del servicio. Este problema pertenece a la familia NP-Hard, por lo que es necesario el uso de meta-heurísticas para hallar soluciones de buena calidad en un tiempo razonable.

En esta línea de trabajo, se propone un algoritmo evolutivo paralelo para resolver el problema de la asignación de contenedores en las rutas de los camiones de basura. Este artículo se organiza de la siguiente manera: La sección III trata sobre el modelado del problema en el contexto del algoritmo genético y su implementación. La sección IV presenta el diseño del algoritmo genético en sí mismo. En la sección V se presenta la evaluación experimental del algoritmo. Finalmente, la sección VI presenta los resultados y conclusiones del trabajo.

II. Descripción Problema

El artículo a continuación trata sobre la optimización del recorrido que realiza la flota de camiones recolectores de basura en distintas instancias de la realidad. Cada instancia se encuentra determinada por la cantidad de camiones recolectores, la capacidad de cada uno de ellos (número máximo de contenedores que puede levantar), la cantidad de contenedores de residuos, su ubicación y el número máximo de días que pueden pasar sin ser levantados por un camión. También se conoce la ubicación del vertedero, desde donde comienza y finaliza el recorrido de cada camión. Se considerará que los camiones realizan dos turnos diarios de 8 horas cada uno (uno matutino y uno nocturno), por lo que cada recorrido no puede durar más de 8 horas.

El problema en concreto es, sabiendo cuantos días pasaron desde la última vez que cada contenedor fue levantado, encontrar la planificación de los recorridos de la flota de camiones que minimiza la distancia que recorren y la cantidad de camiones necesarios para cumplir con la demanda, sujeto a que ningún contenedor supera el número máximo de días que puede pasar sin ser levantado. En el caso que no sea posible cumplir con esto último, se busca el itinerario que deja la menor cantidad posible de contenedores sin levantar.

III. Modelado del Problema

Se considerará una instancia genérica en la cual se dispone de k camiones de basura con capacidad p y n contenedores. Para cada contenedor n se conoce la cantidad máxima de días que pueden pasar sin que sea levantado. Para ello se define el vector $Fr \in \mathbb{N}^n$ el cual contiene la frecuencia de recolección de cada contenedor.

Sea $c : \mathbb{N} \to \mathbb{N}^n$, una función que indica cuantos días pasaron desde que cada contenedor fue vaciado para el día d. La mismo se encuentra definida por:

$$c(d+1) = \begin{cases} f(d) + \mathbb{1} & d > 0 \\ c_0 & d = 0 \end{cases}$$

Donde c_0 es un vector arbitrario y $f : \mathbb{N} \to \mathbb{N}^n$ la función que indica si un contenedor es levantado en el día d. Sea $f(d)_i$ la salida í-esima de $f(d)$, esta se define como:

$$f(d)_i = \begin{cases} c(d)_i & \text{sí no se levanta el contenedor } i \text{ el día } d \\ 0 & \text{sí se levanta el contenedor } i \text{ el día } d \end{cases}$$

Por otro lado, la planificación de los recorridos de la flota queda determinada por los itinerarios que se le definen a cada camión recolector de basura. A cada camión se le define un itinerario por turno (dos por día), que indica qué contenedores debe levantar el mismo en dicho turno. Cada itinerario se define como una matriz $I \in \{0,1\}^{k \times n}$, donde cada índice indica si el k-ésimo camión levanta el n-ésimo contenedor o no. Para diferenciar los itinerarios de los diferentes turnos y días, se denomina $I_m(d)$ a la función que retorna el turno matutino del día d, y $I_n(d)$ para el turno nocturno. Es importante dar la planificación de los Q días siguientes, siendo $Q = \text{máx}_n\{F_n\}$, debido a que de otra manera no se estarían contemplando problemas que pudieran surgir más adelante con otros contenedores. Dado que no todos los camiones tienen que tener rutas asignadas, se define x como la suma de todos los camiones que se usan en todos los turnos que se realizan.

No obstante, es importante conocer también el orden en que se recorren los contenedores en cada día. Para ello se define, la función $R : \mathbb{N}^n \to \mathbb{N}^{p+2}$ la cual dado el itinerario de un tuno, retorna un vector ordenado con el orden en que se visita cada contenedor identificado por su índice. La primera y última entrada corresponden al basurero donde el camión debe comenzar y terminar su recorrida.

Para calcular el tiempo y la distancia que implica cada recorrido de los camiones, defínase $D, T \in \mathbb{R}^{n+1 \times n+1}$ que indican la distancia y el tiempo respectivamente que le lleva a un camión ir de cada contenedor al resto. La última fila y columna refieren a los costos de viajar desde el punto de partida (basurero) a un contenedor y viceversa.

Con todo esto, el problema se resume a dado c_0, hallar las funciones que minimizan:

$$\begin{aligned} \min_{I_n, I_m, x} \quad & \sum_{d=1}^{Q} \sum_{i=1}^{p+1} \sum_{j=1}^{p+2} D_{R(I_n(d)_i), R(I_n(d))_j} \\ & + D_{R(I_m(d)_i), R(I_m(d))_j} + x \\ \text{s.t.} \quad & F_i - c(d)_i > 0 \ \forall i, \\ & \sum_{i=1}^{p+1} \sum_{j=1}^{p+2} T_{R(I_n(d)_i), R(I_n(d))_j} < 8 \ \forall d, \\ & \sum_{i=1}^{p+1} \sum_{j=1}^{p+2} T_{R(I_m(d)_i), R(I_m(d))_j} < 8 \ \forall d, \\ & \sum_{j=1}^{k} I_{n_{i,j}} \leq p \wedge \sum_{j=1}^{k} I_{m_{i,j}} \leq p \ \forall i \end{aligned} \tag{1}$$

En caso de que no exista solución factible, se busca la solución que cumpla lo mejor posible con la frecuencia de recolección:

$$\begin{aligned} \min_{I_n, I_m} \quad & || \sum_{d=1}^{Q} c(d)_i || \\ \text{s.t.} \quad & \sum_{i=1}^{p+1} \sum_{j=1}^{p+2} T_{R(I_n(d)_i), R(I_n(d))_j} < 8 \ \forall d, \\ & \sum_{i=1}^{p+1} \sum_{j=1}^{p+2} T_{R(I_m(d)_i), R(I_m(d))_j} < 8 \ \forall d, \\ & \sum_{j=1}^{k} I_{n_{i,j}} \leq p \wedge \sum_{j=1}^{k} I_{m_{i,j}} \leq p \ \forall i \end{aligned} \tag{2}$$

IV. Estrategia de resolución

IV-A. *Representación de las soluciones*

Dado que las soluciones van a ser las matrices binarias $I_m(d)$ y $I_n(d)$ que representan los contenedores que levantará cada camión en cada itinerario, se utilizará una representación binaria donde se concatenarán las matrices representándolas como una tira de bits. Por lo tanto, las soluciones tendrán un largo de $n \times k \times Q \times 2$ bits. La figura 1 muestra la codificación para un único turno de 2 camiones y 4 contenedores.

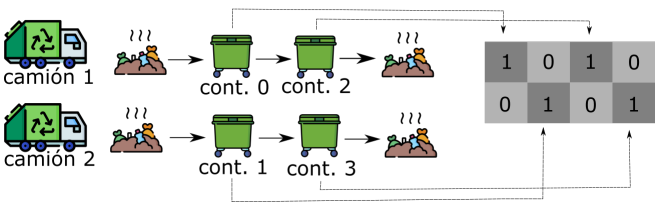

Figura 1. Representación de una solución

IV-B. *Inicialización de la población*

La generación inicial de individuos se genera a partir de varias estrategias distintas. Para empezar, el primer individuo se genera a partir del algoritmo greedy ”puro” presentado en

V-E. Luego, se dispone de tres estrategias distintas para crear individuos:

1. Greedy con componente aleatorio: Se utiliza el algoritmo greedy en su versión aleatoria (imponiendo qué contenedor levantar primero).
2. Greedy con componente aleatorio e intercambio de turnos: A la estrategia anterior se le agrega una rotación aleatoria en sus turnos y días. Es decir, el itinerario correspondiente a un determinado día en un determinado turno se intercambia con otro itinerario elegido de forma aleatoria.
3. Aleatoria: A cada camión se le asigna contenedores de forma aleatoria en cada uno de sus turnos.

Cada vez que se desea crear un individuo se elige una de estas tres estrategias. Las mismas son elegidas con una probabilidad de $16{,}\widehat{6}\,\%$, $16{,}\widehat{6}\,\%$ y $66{,}\widehat{6}\,\%$ respectivamente con el fin de asegurar la diversidad de la población y aún así contener soluciones iniciales de buena calidad.

IV-C. Función de Fitness

La función de fitness se platea como el problema inverso de las ecuaciones 1 y 2 de la sección III. A su vez, es preferible que los camiones junten la basura durante el turno nocturno antes que el matutino para minimizar su impacto en el trafico de la ciudad. Tomando lo anterior en consideración, sea x una solución, $S(x)$ el conjunto de itinerarios que codifica x, $Dist(S(x))$ la distancia total del costo de los recorridos (los recorridos en el turno diurno son más costosos), $Camiones(S(x))$ la cantidad de camiones que se emplean y $d = Desborde(S(x))$ la suma de días de retraso de recolección para los contenedores que no se respetó su frecuencia horaria correspondiente. Se define el fitness $f(x)$:

$$f(x) = \begin{cases} K - Dist(S(x))^2 - \alpha Camiones(S(x)) & d = 0 \\ -d^2 & d > 0 \end{cases}$$

Donde K es una constante fija tal que $d = 0 \rightarrow f(x) > 0$. De esta forma, las soluciones que no respetan la frecuencia de recolección van a ser inferiores a las que si lo hacen. α es un factor para equipar el costo de la distancia recorrida con el costo base de emplear un camión en un itinerario. El ajuste de este parámetro depende de la realidad del problema.

IV-D. Operadores evolutivos

IV-D1. Cruzamiento: Se realizaron varias pruebas exploratorias con distintos operadores analizando los resultados y tiempo de ejecución. Se optó por un operador de medio cruzamiento uniforme - *Half Uniform Corssover* (HUX) - el cual, por el tipo de solución trabajada, tiene un buen rendimiento y preserva las características compartidas de los progenitores.

IV-D2. Mutación: De forma similar, se exploraron distintas alternativas y se escogió la mutación de un bit por su simplicidad y poca capacidad destructiva sobre las soluciones.

IV-D3. Reparación: Dado que los operadores de mutación y cruzamiento pueden generar soluciones invalidas: No se respeta las capacidades de los camiones o el tiempo de recorrida supera las 8 horas. En este caso se aplica un operador de reparación que asegura que la solución respeta las restricciones mencionadas. Se trata también que las soluciones cumplan con las frecuencias de recolección y tengan un componente aleatorio para mantener la diversidad. Se presenta un pseudocódigo de este operador (se le llama ruta ociosa a aquella que es posible asignarle un contenedor y respetar la capacidad del camión que la realiza):

Algorithm 1 Reparación

Require: $Rutas$
  $U \leftarrow \emptyset$
  for each $R \in$ Rutas **do**
    while R no respeta la capacidad máxima **do**
      $c \leftarrow$ contenedor aleatorio levantado en R
      $R \leftarrow R - \{c\}$
      if c no es levantado en ninguna ruta **then**
        $U \leftarrow U \cup \{c\}$
      end if
    end while
  end for
  while $U \neq \emptyset \wedge \exists$ Rutas ociosas **do**
    $c \leftarrow$ contenedor aleatorio $\in U$
    $R \leftarrow$ ruta ociosa aleatoria
    $R \leftarrow R \cup \{c\}$
    $U \leftarrow U - \{c\}$
  end while

IV-D4. Selección y Reemplazo: La estrategia de solución utilizada es selección por torneo y el esquema de reemplazo $\mu + \lambda$.

IV-E. Arquitectura del Algoritmo

Se aplica el clásico algoritmo genético simple [1] modificado para seguir el modelo *master/slave* de tipo asíncrono. Es decir, se implementó un algoritmo evolutivo paralelo (PEA) el cual tiene un hilo maestro que dirigirá la búsqueda evolutiva y múltiples hilos encargados de evaluar las soluciones de forma asíncrona. Dado que evaluar las soluciones es una operación costosa, esto acelera de forma considerable el tiempo computacional del algoritmo.

V. Evaluación experimental

V-A. Obtención de Instancias

Para probar el algoritmo propuesto, se generaron instancias basadas en datos de la realidad de la ubicación geográfica de los contenedores en Montevideo (Uruguay). A partir de la ubicación de cada contenedor, se construyeron las matrices de tiempo y distancia de viaje de un camión a cada contenedor tomando en consideración las calles de la ciudad y sus distintas características (sentido, velocidad máxima, tráfico, etc.) Para ello se utilizaron los siguientes *datasets*,servicios y herramientas:

- Catálogo de datos abiertos: Es un repositorio de datos abiertos del gobierno Uruguayo. Contiene datos y estadísticas sobre educación, salud, infraestructura, economía, transporte, etc. Particularmente utilizamos un *dataset* que contiene la ubicación de contenedores de residuos domiciliarios con la frecuencia de recolección programada, y los circuitos de recolección a los que pertenecen. [2] El Sistema de Referencia de las coordenadas definidas en los datos es SIRGAS2000 ITRF2000, proyección UTM 21S.
- EPSG.io: Es un servicio gratuito de conversión de coordenadas para distintos sistemas de referencia [6]. El mismo fue utilizado para transformar las coordenadas de los contenedores del sistema de referencia mencionado a latitud/longitud.
- Graphhopper: Una aplicación de mapas de código abierto [5]. Como otros servicios de mapa, Graphopper permite la creación de pasos para llegar a alguna dirección permitiendo calcular el tiempo necesario y la distancia recorrida entre las ubicaciones considerando las calles y diversos factores. Esta aplicación fue utilizada para la construcción de las matrices de costo y distancia de los contenedores.

Se calculó las matrices de tiempo y distancia para los 11556 contenedores de la ciudad de Montevideo. El *dataset* elaborado (de un tamaño aproximado de 6.5GB) se encuentra disponible en [4]. Los scripts, realizados en Python, para la obtención de los datos y procesamiento se encuentran disponibles en el repositorio del proyecto [13].

V-B. Implementación

La implementación del algoritmo evolutivo se realizó en Java utilizando utilizando la librería *jMetal* [7] la cual provee diversos esqueletos de algoritmos evolutivos y operadores los cuales se pueden extender fácilmente utilizando técnicas de orientación a objetos. El esqueleto de solución utilizado fue el *AsynchronousMultiThreadedGeneticAlgorithm*, el cual implementa un algoritmo PEA asíncrono. Por otro lado, la función de fitness requiere calcular los costos de recorrer un conjunto de contenedores para cada camión. Dado que el orden en que se visitan los contenedores afecta la distancia recorrida en gran medida, resulta indispensable obtener un orden de recorrida relativamente bueno para evaluar de manera justa los distintos itinerarios. Para ello, se utilizó la librería *jsprit* [11] la cual permite resolver problemas VRP con restricciones. La misma utiliza una meta-heurística llamada *Ruin & Recreate* [12] junto con *simulated annealing*. Adicionalmente al paralelismo del algoritmo per se, la función de evaluación del fitness también es multi-hilada. Dado que se tiene que obtener el costo de cada ruta del itinerario, es posible calcular el costo de cada ruta de forma concurrente. Permitiendo así un speedup teórico máximo de $4n$, donde n es la cantidad de camiones.

V-C. Hardware Utilizado

Todas las evaluaciones fueron realizadas en ClusterUY [9]. Un clúster de computadoras que permite ejecutar varios trabajos en paralelo y utilizando múltiples CPUs de última generación. Para las ejecuciones de los algoritmos se asignaron siempre los siguientes recursos:

- CPU: 39 núcleos Intel Xeon-Gold 6138 2.00GHz
- Memoria: 100GB de memoria RAM.
- Sistema Operativo: GNU/Linux, distribución CentOS7

V-D. Ajuste Paramétrico

Dado que los algoritmos evolutivos son métodos de búsqueda estocásticos, y con el fin de conseguir un algoritmo que brinde la mejor calidad de resultados, es necesario hacer un ajuste de los parámetros del algoritmo. Con este objetivo se construyeron 3 instancias de tamaño medio, y diferentes a las de las otras pruebas experimentales para evitar sesgos.

En dicho ajuste se probaron distintas configuraciones de los tres parámetros principales: tamaño de la población, probabilidad de cruzamiento y probabilidad de mutación. Para cada parámetro, se tomaron 3 valores candidatos:

- Tamaño de la población: $\{50, 100, 150\}$
- Probabilidad de cruzamiento: $\{0.6, 0.75, 0.95\}$
- Probabilidad de mutación: $\{ \frac{1}{n \times k \times 4}$,0.001, 0.01$\}$

Se tomaron en cuenta todas las combinaciones posibles de los parámetros (27) para luego ser analizadas. Para dicho análisis se realizaron 50 ejecuciones independientes de 10.000 generaciones cada una. Para cada configuración se reporta el máximo y mínimo fitness, fitness promedio y desviación estándar del mismo. Los valores de los resultados pueden consultarse en el anexo y se encuentran presentados como boxplots en las figuras 4, 5 y 6. Las 27 configuraciones fueron numeradas del 1 al 27 siguiendo el orden de las listas de candidatos definidas anteriormente recorriendo primero la de mutación, luego cruzamiento y finalmente tamaño de la población.

Luego de realizadas las ejecuciones, se realizaron tests de normalidad para comprobar si los resultados obtenidos siguen una distribución normal. Para realizar dichas pruebas se utilizó el test de normalidad de Anderson-Darling. Para todas las configuraciones el test arrojó resultados que rechazan la hipótesis nula de que los datos siguen una distribución normal. Debido a ello, para comparar las distintas configuraciones paramétricas se deben utilizar tests no paramétricos. Para saber si existen diferencias significativas entre las medianas de los resultados de las distintas configuraciones se decidió utilizar el test de Kruskal-Wallis. El mismo fue utilizado para comparar cada pareja de configuraciones, y así obtener cuales configuraciones se puede asegurar que son mejores que otras. Las imágenes 7, 8 y 9 muestran el resultado del p-valor para cada par de algoritmos, indicando si se rechaza la hipótesis nula (rojo) o no (azul) con una significancia del 5 %. Rechazar la hipótesis indica que la mediana de los dos algoritmos es significativamente distinta y por lo tanto existe una diferencia estadística entre ellos. Los test de Anderson-Darling y Kruskal-Wallis fueron aplicados utilizando la librería SciPy [8] de Python.

A partir de los resultados de este test, se decidió seleccionar la siguiente configuración (algoritmo 25):

- Tamaño de la población: 150
- Probabilidad de cruzamiento: 0.95
- Probabilidad de mutación: $\frac{1}{n \times k \times 4}$

Esta configuración fue elegida debido a que dio el mejor fitness promedio entre las tres instancias. A su vez, el test de Kruskal-Wallis muestra que existe una diferencia en las medianas para la mayoría de los algoritmos que se lo compara así concluyendo que los resultados del algoritmo son estadísticamente significativos.

V-E. Comparación con otras técnicas

Tanto para comparar el algoritmo evolutivo, como para inicializarlo se construyó un algoritmo ávido (greedy) determinista que brinde soluciones razonablemente buenas al problema. El algoritmo consiste en asignarle a cada camión el contenedor que se encuentra más próximo a su ubicación actual en cada momento. Además, se le da prioridad a los contenedores que están llenos y al turno nocturno. A continuación se presenta un pseudocódigo del mismo.

Algorithm 2 Greedy

```
for each day do
    cOb ← conjunto de contenedores llenos
    cam ← primer camión
    cAct ← vertedero    (contenedor actual)
    for each turno do
        while cOb ≠ ∅ ∧ turno no lleno do
            cont ← contenedor ∈ cOb más cercano a cAct
            cam ← cont    (se le asigna)
            cOb ← cOb − {cont}
            cAct ← cont
            if cam lleno then
                cam ← vertedero    (se le asigna)
                cam ← siguiente camión
                cAct ← vertedero
            end if
        end while
        while turno no lleno ∧ quedan contenedores sin
levantar do
            cont ← contenedor no vaciado más cerca de cAct
            cam ← cont    (se le asigna)
            cAct ← cont
            if cam lleno then
                cam ← vertedero    (se le asigna)
                cam ← siguiente camión
                cAct ← vertedero
            end if
        end while
    end for
end for
```

Para usar el algoritmo greedy en la inicialización de la población se lo forzó a iniciar el recorrido por un contenedor i elegido de forma aleatoria, de modo que no todos los individuos generados fueran iguales.

Para comparar el algoritmo evolutivo con el greedy se generaron tres instancias de 1000, 800 y 600 contenedores (numeradas del 1 al 3 respectivamente). Se realizaron 30 ejecuciones de 100.000 generaciones para cada instancia, para luego comparar los resultados obtenidos con los dados por el greedy. Como se puede apreciar en la figura 2, el algoritmo supera ampliamente al greedy cuando se trabaja con instancias de gran tamaño, mientras que con instancias más pequeñas la mejora no es tan importante.

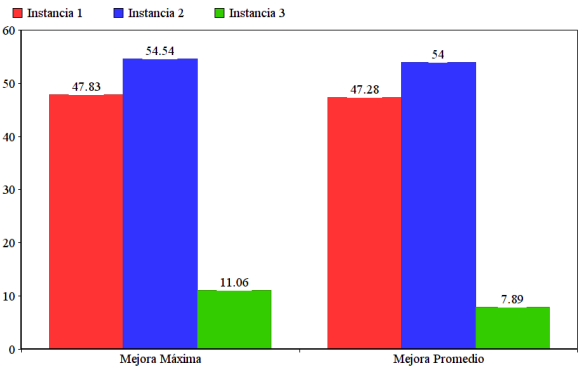

Figura 2. Porcentaje de mejora del algoritmo con respecto al greedy

Como se puede apreciar en el gráfico, el desempeño del algoritmo evolutivo es considerablemente mejor para las instancias más grandes. Para verificar estadísticamente que la diferencia de fitness es significativa, se comprueba que se cumple el criterio de significancia estadística $|AE_{avg} - greedy_{avg}| > \max\{\sigma(AE), \sigma(greedy)\}$ (donde $\sigma(greedy) = 0$ al ser determinista) para las 3 instancias. Comprobándose que el algoritmo evolutivo es mejor al greedy en las 3 instancias. No obstante, cabe destacar que el tiempo de ejecución del algoritmo greedy (en el orden de los milisegundos) es mucho menor al del algoritmo evolutivo, el cual precisa varios minutos para encontrar soluciones aceptables. En las sección V-F se presenta un estudio más detallado del tiempo de ejecución del algoritmo evolutivo.

V-F. Análisis de rendimiento del algoritmo paralelo

Para las mismas instancias de 1000, 800 y 600 contenedores se realizó un estudio del tiempo de ejecución del algoritmo variando la cantidad de núcleos de CPU que el algoritmo utiliza. Para ello se realizaron 30 ejecuciones de cada caso tomando como criterio de parada 100 generaciones. Los resultados pueden visualizarse en la figura 3 y en la tabla V-F con el valor promedio de tiempo de ejecución para cada caso. A su vez, se calculó el speedup algorítmico y la eficiencia computacional para analizar la mejora de desempeño del algoritmo paralelo. Observamos para las 3 instancias que la mayor eficiencia computacional se da al usar 39 núcleos. Esto muestra una buena escalabilidad del algoritmo para la cantidad de núcleos que se probaron. No pudimos hacer pruebas mayores a 39 debido a que clusterUY permite asignarle a un trabajo no más de 39 núcleos en un único nodo.

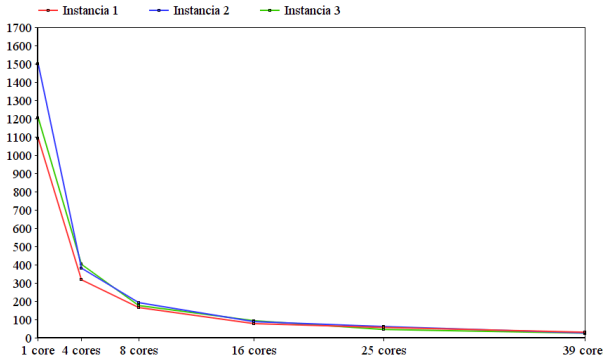

Figura 3. Tiempo de ejecución en función de la cantidad de núcleos utilizados

Inst	#cores	T Prom. (s)	SpeedUp Alg.	Ef. Comp.
1	1	1093		
1	4	318	3.44	0.86
1	8	165	6.24	0.78
1	16	77	14.19	0.89
1	25	56	19.52	0.78
1	39	30	36.43	0.93
2	1	1499		
2	4	382	3.92	0.98
2	8	192	7.81	0.98
2	16	88	17.03	1.06
2	25	61	24.57	0.98
2	39	26	57.65	1.48
3	1	1205		
3	4	403	2.99	0.75
3	8	176	6.85	0.86
3	16	93	12.96	0.81
3	25	45	26.78	1.07
3	39	25	48.2	1.24

VI. Conclusiones

A lo largo del proyecto se trabajó sobre el problema de asignación de rutas de un servicio de recolección de residuos. Se diseño e implementó un algoritmo evolutivo que resuelve el problema, buscando minimizar la distancia recorrida por los camiones respetando las restricciones del problema. Además se presentó un algoritmo ávido que sirvió como punto de partida del evolutivo y como referencia para evaluar las soluciones brindadas por el mismo.

Una vez diseñado el algoritmo evolutivo, se generaron instancias de prueba para realizar el ajuste paramétrico. De esta forma se ajustó el tamaño de la población, y las probabilidades de mutación y cruzamiento, buscando el mejor rendimiento posible del algoritmo.

De este ajuste paramétrico surgió una configuración definitiva, que fue utilizada para probar el algoritmo con instancias más grandes (más contenedores y camiones). A partir de estas evaluaciones se puede concluir que el algoritmo presenta resultados satisfactorios, ya que sus soluciones son ampliamente superiores a las brindadas por el greedy, con un porcentaje de mejora de hasta un **54 %**, como se puede ver en la figura 2. Esto debido a que el enfoque local del greedy no le permite tomar decisiones que, si bien no son óptimas localmente, sí lo son a nivel global. Aunque también es cierto que el tiempo requerido es mucho mayor en el caso del evolutivo.

También se observó un buen desempeño en la eficiencia del paralelismo del algoritmo. Mostrando su escalabilidad al disponer de decenas de núcleos. Si bien la función de evaluación es compleja y costosa, al hacerla paralela se obtuvo una aceleración significativa en el cálculo del fitness y permite al AE aprovechar aún más los recursos que dispone. Sin embargo, el tiempo de ejecución total sigue siendo considerablemente alto, lo cual dificultó la realización de las pruebas experimentales que requerían una gran cantidad de ejecuciones del algoritmo.

VI-A. Trabajos Futuros

Hay varios puntos sobre los cuales se puede trabajar para mejorar el proyecto. En cuanto a el cálculo de las distancias y tiempos entre contenedores, se podría evaluar el uso de una herramienta que brinde información en tiempo real de las rutas óptimas, aunque quizás esto cause un aumento considerablemente en el tiempo requerido por el algoritmo.

Además, para obtener mejores resultados, podrían refinarse aún más los parámetros utilizados, por ejemplo con una búsqueda mas exhaustiva (más combinaciones de parámetros). También podrían considerarse otros operadores genéticos, y evaluar si presentan mejores resultados que los utilizados. Otra modificación interesante sería introducir variantes al problema, por ejemplo ser más flexibles a la hora de permitir el desborde de algunos contenedores. Una posibilidad sería plantear un algoritmo multiobjetivo, que busque minimizar la cantidad de camiones utilizados a la vez que intenta minimizar la cantidad de contenedores desbordados. De esta forma, se podría buscar en el frente de Pareto la solución que el cliente considere se adecúa mejor a su realidad.

Si bien JMetal simplificó el proceso de implementación, muchas veces nos hubiera gustado tener un mayor control sobre nuestro algoritmo. Podría considerarse utilizar un lenguaje de programación más eficiente (como C o C++) con un mayor control sobre el manejo y comunicación entre hilos que nos permita aplicar otros modelos de PEA interesantes como es el de sub-poblaciones distribuidas. A su vez, tras la experiencia de utilizar ClusterUY, consideramos que también podría adaptarse el algoritmo evolutivo para que funcione mejor en un entorno de computación distribuido, como es un clúster, permitiendo utilizar varios nodos para su ejecución.

Referencias

[1] Goldberg D. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, New York, NY, 1953.

[2] División Limpieza de la Intendencia de Montevideo. Contenedores de residuos domiciliarios - ubicación, circuitos y frecuencia de recolección. *Catálogo de Datos Abiertos, AGESIC - Licencia de DAG de Uruguay*, Versión 5 de marzo de 2021.

[3] Intendencia de Montevideo. Más transparencia de la gestión, 2017.

[4] Toyos G. Vallcorba F. Montevideo garbage containers. 10.6084/m9.figshare.17126738, 2021.

[5] GraphHopper GmbH. Graphhopper routing engine. https://github.com/graphhopper/graphhopper, 2021.

[6] Klokan Technologies GmbH. Epsg.io: Find coordinate systems worldwide, 2021.

[7] A.J. Nebro J.J. Durillo. jmetal: a java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[8] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.
[9] Iturriaga S. Nesmachnow S. Cluster-uy: Collaborative scientific high performance computing in uruguay. *Communications in Computer and Information Science*, 1151, 2019.
[10] Schneider J. Schrimpf G. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 1999.
[11] Stefan Schröder. jsprit : java toolkit for rich vrps and tsps. https://github.com/graphhopper/jsprit, 2021.
[12] Vigo D. Toth P. The vehicle routing problem. *Monographs on Discrete Mathematics and Applications*, 2001.
[13] Vallcorba. V Toyos G. Ae2021 (solicitar acceso). https://gitlab.fing.edu.uy/guillermo.toyos/ae2021_practico/, 2021.

VII. Anexo

VII-A. Código Fuente del Proyecto

El código fuente, junto con las instancias de la evaluación experimental, se encuentran disponibles en el repositorio del proyecto [13]. A su vez, todo el código fue documentado utilizando la herramienta *javadoc*.

VII-B. Resultados del Ajuste Paramétrico

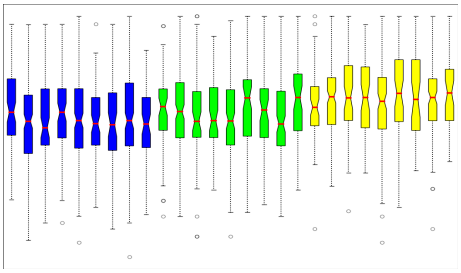

Figura 4. Box Plot de los resultados en la instancia 1 (ajuste paramétrico)

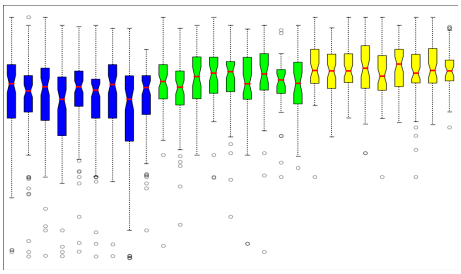

Figura 5. Box Plot de los resultados en la instancia 2 (ajuste paramétrico)

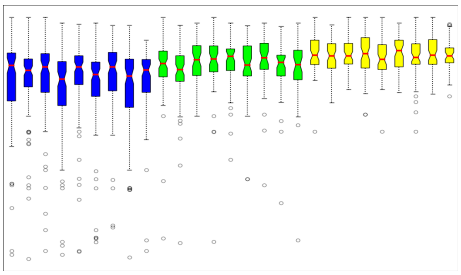

Figura 6. Box Plot de los resultados en la instancia 3 (ajuste paramétrico)

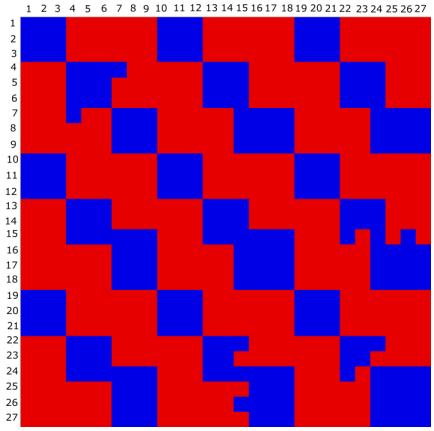

Figura 7. Matriz de p-valores para la instancia 1. Rojo indica que existe una diferencia estadística entre los algoritmos

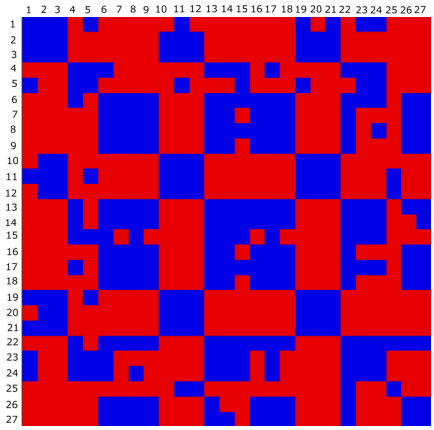

Figura 8. Matriz de p-valores para la instancia 2. Rojo indica que existe una diferencia estadística entre los algoritmos

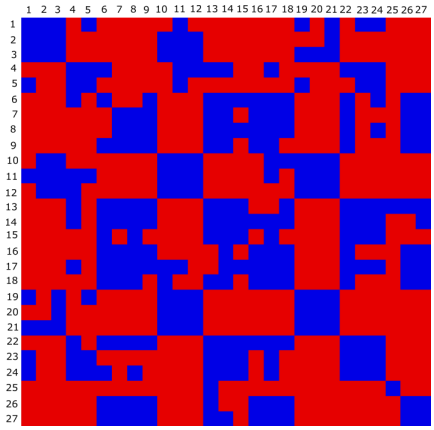

Figura 9. Matriz de p-valores para la instancia 3. Rojo indica que existe una diferencia estadística entre los algoritmos

Cuadro I
RESULTADOS DE LA CONFIGURACIÓN PARAMÉTRICA EN LA INSTANCIA 1 (FITNESS)

Población	Cruzamiento	Mutación	Mín	Mediana	Máx	Desvío
50	0.6	0.1	17790943378758	17851688046469	17852777389349	22528002695
50	0.6	0.001	17790848280214	17851594715286	17852776211565	22204565519
50	0.6	0.01	17791357811085	17851562710416	17852777389349	22646092518
50	0.75	0.1	17792008079524	17851659101046	17852777389349	23133290564
50	0.75	0.001	17788056708922	17851598841241	17852851433552	22857288289
50	0.75	0.01	17791347193956	17851429396590	17852777389349	23546709933
50	0.95	0.1	17777702062410	17851169982831	17852777389349	24746730899
50	0.95	0.001	17789419064448	17851594715286	17852851433552	23950818153
50	0.95	0.01	17791736093616	17851468316234	17852540595176	22954301924
100	0.6	0.1	17801216093588	17851890393657	17852762747472	12369593541
100	0.6	0.001	17798406639283	17851893214326	17852851433552	15239654075
100	0.6	0.01	17792502518541	17851818830767	17852851433552	17398706866
100	0.75	0.1	17800704843011	17851791782412	17852667601687	15811292078
100	0.75	0.001	17800625337097	17851811200939	17852809731701	12478907618
100	0.75	0.01	17800380363324	17851988427399	17852851433552	12590116524
100	0.95	0.1	17800808985179	17851782330351	17852851433552	16017531601
100	0.95	0.001	17793371933921	17851765880936	17852851433552	14730432118
100	0.95	0.01	17800596571027	17851996304863	17852851433552	14308931297
150	0.6	0.1	17801149959798	17852007476390	17852851433552	7081406993
150	0.6	0.001	17800823277833	17852108826317	17852851433552	9878841835
150	0.6	0.01	17800691694418	17852065168237	17852851433552	12663454896
150	0.75	0.1	17801720859107	17852103007053	17852776211565	8687529378
150	0.75	0.001	17800723120943	17852073225225	17852851433552	8322810866
150	0.75	0.01	17800997434365	17852110840440	17852851433552	11582622069
150	0.95	0.1	17851444934839	17852090720506	17852851433552	378233691
150	0.95	0.001	17800749833021	17852103007053	17852851433552	7090575971
150	0.95	0.01	17829789828687	17852131540571	17852851433552	5443600964

Cuadro II
RESULTADOS DE LA CONFIGURACIÓN PARAMÉTRICA EN LA INSTANCIA 2 (FITNESS)

Población	Cruzamiento	Mutación	Mín	Mediana	Máx	Desvío
50	0.6	0.004166667	937020814850	947836714047	948615806230	2398809549
50	0.6	0.001	937168005723	947694669465	948615806150	3325971531
50	0.6	0.01	935814641197	935833281317	948615805882	2840022705
50	0.75	0.004166667	934945299755	947610326865	948526378147	3217567643
50	0.75	0.001	935494240150	947797694624	948509984413	3245982706
50	0.75	0.01	934203767749	947685923392	948526377965	2997349515
50	0.95	0.004166667	936167503094	947697230510	948491649007	2814771346
50	0.95	0.001	934699895616	947636709111	948491648649	2964732581
50	0.95	0.01	937168005956	947694669443	948254960380	2606394440
100	0.6	0.004166667	938292678644	947888149630	948615805951	1160528657
100	0.6	0.001	945081814430	947793032814	948491648804	532678659
100	0.6	0.01	947070944873	947951359830	948526378095	297888758
100	0.75	0.004166667	937168006027	947952534675	948615805957	1597278504
100	0.75	0.001	946382447365	948005048115	948526377810	424664840
100	0.75	0.01	938610346348	947866591158	948482563620	1373633438
100	0.95	0.004166667	945987870194	947977345411	948526377910	463171466
100	0.95	0.001	945704444772	947906308487	948469205098	466849840
100	0.95	0.01	945122589312	947870695415	948526378022	577528238
150	0.6	0.004166667	946827635669	948018002497	948615806425	304805147
150	0.6	0.001	947273998441	948011042178	948495877638	279684476
150	0.6	0.01	947486193264	948011042209	948615806282	244775372
150	0.75	0.004166667	947090936313	948044137779	948615805963	337368535
150	0.75	0.001	946827635695	947952709808	948615806137	294057083
150	0.75	0.01	947437744619	948089681853	948526378119	274287239
150	0.95	0.004166667	946827635935	947985873391	948615806241	329873536
150	0.95	0.001	947412517859	948018002679	948615806183	274074670
150	0.95	0.01	947379308082	948011042367	948509984356	239959682

Cuadro III
RESULTADOS DE LA CONFIGURACIÓN PARAMÉTRICA EN LA INSTANCIA 3 (FITNESS)

Población	**Cruzamiento**	**Mutación**	**Mín**	**Mediana**	**Máx**	**Desvío**
50	0.6	0.004166667	937002174422	947818073924	948597165843	2398809558
50	0.6	0.001	924356718327	947676029153	948597165843	3948931165
50	0.6	0.01	935814641197	947768670478	948597165843	2840022712
50	0.75	0.004166667	924504853048	947590613237	948507737738	4771452367
50	0.75	0.001	916812134303	947779054449	948491344183	4330065891
50	0.75	0.01	934185127450	947667282955	948507737738	2997349551
50	0.95	0.004166667	922096883593	947677309631	948473008579	3699249118
50	0.95	0.001	916346855789	947606535002	948473008579	4143290296
50	0.95	0.01	937149365843	947676029153	948236320319	2606394446
100	0.6	0.004166667	938274038507	947869509321	948597165843	1160528638
100	0.6	0.001	945063174290	947774392700	948473008579	532678653
100	0.6	0.01	947052304569	947932719638	948507737738	297888726
100	0.75	0.004166667	937149365843	947933894325	948597165843	1597278508
100	0.75	0.001	946363807025	947986407892	948507737738	424664856
100	0.75	0.01	938591705773	947847951055	948463923312	1373633476
100	0.95	0.004166667	945969229931	947958705127	948507737738	463171442
100	0.95	0.001	945685804462	947887668289	948450564958	466849840
100	0.95	0.01	945103949232	947852055296	948507737738	577528220
150	0.6	0.004166667	946808995508	947999362351	948597165843	304805125
150	0.6	0.001	947255358208	947992402104	948477237584	279684496
150	0.6	0.01	947467553075	947992402104	948597165843	244775362
150	0.75	0.004166667	947072296130	948025497379	948597165843	337368491
150	0.75	0.001	946808995508	947934069611	948597165843	294057079
150	0.75	0.01	947393877806	948071041756	948507737738	274287241
150	0.95	0.004166667	946808995508	947967233217	948597165843	329873517
150	0.95	0.001	947393877806	947999362351	948597165843	274074649
150	0.95	0.01	947360667844	947992402104	948491344183	239959691