

CECAL

INSTITUTO DE COMPUTACIÓN, FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA
MONTEVIDEO, URUGUAY

PROYECTO DE GRADO INGENIERÍA EN COMPUTACIÓN

Optimización de viajes compartidos en taxis utilizando algoritmos evolutivos

Gabriel Fagúndez de los Reyes, Renzo Massobrio
gabrielf@fing.edu.uy, renzom@fing.edu.uy

Agosto de 2015

Tutor del Proyecto:
Sergio Nesmachnow, Universidad de la República.

Optimización de viajes compartidos en taxis utilizando algoritmos evolutivos
Fagúndez de los Reyes, Gabriel
Massobrio, Renzo
Proyecto de Grado
CECAL
Instituto de Computación - Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, Agosto de 2015

OPTIMIZACIÓN DE VIAJES COMPARTIDOS EN TAXIS UTILIZANDO ALGORITMOS EVOLUTIVOS

RESUMEN

Este proyecto estudia el problema de planificación de viajes compartidos en taxis para el caso de pasajeros viajando desde un mismo origen a múltiples destinos. Se estudian dos variantes del problema: una variante monoobjetivo donde se busca minimizar el costo total de un grupo de pasajeros y una variante multiobjetivo del problema, donde además de minimizar el costo, se plantea minimizar simultáneamente la demora percibida por cada usuario.

Se implementan cuatro algoritmos evolutivos diferentes: dos para cada una de las variantes del problema. Los algoritmos evolutivos implementados son evaluados utilizando un conjunto de instancias realistas del problema, generadas a partir de datos obtenidos de trazas de *GPS* instaladas en taxis de la ciudad de Beijing, China. Además, se generan y utilizan instancias de prueba específicas para la ciudad de Montevideo, Uruguay. Los resultados experimentales muestran que para la variante monoobjetivo, los algoritmos evolutivos implementados alcanzan mejoras de hasta un 41.0 % en costo, respecto a una estrategia ávida que resuelve el mismo problema. Para la variante multiobjetivo del problema, los resultados experimentales muestran que los algoritmos evolutivos implementados logran mejorar hasta un 105.2 % la demora alcanzada por un algoritmo ávido que busca optimizar el costo y hasta en un 75.1 % el costo alcanzado por un algoritmo ávido que busca minimizar la demora.

Finalmente se presenta el planificador de viajes compartidos en línea, un componente de software que incluye una aplicación web y una aplicación móvil, que permite a los usuarios resolver instancias realistas del problema de viajes compartidos en taxis de forma eficiente y amigable.

Palabras clave: viajes compartidos, algoritmos evolutivos, optimización, transporte urbano.

TAXI SHARING OPTIMIZATION USING EVOLUTIONARY ALGORITHMS

ABSTRACT

This project studies the taxi sharing optimization problem, where a group of passengers are travelling from the same origin to multiple destinations. Two different variants of the problem are studied: a single objective variant with the goal of minimizing the total cost spent by the group of passengers and a multiobjective variant, which proposes the simultaneous minimization of the total cost spent by the group of passengers and the delay experienced by each passenger.

Four different evolutionary algorithms are implemented: two for each one of the problem variants. The evolutionary algorithms implemented are evaluated using a set of real problem instances, generated using GPS data from taxis in the city of Beijing, China. Furthermore, a set of instances for the city of Montevideo, Uruguay, is generated and used in the experimental evaluation. The experimental results show that the evolutionary algorithms developed for the single objective variant of the problem are able to improve up to 41.0 % upon the cost computed using a greedy algorithm. Regarding the multiobjective variant of the problem, the experimental results show that the evolutionary algorithms are able to compute solutions that improve up to 105.2 % upon the delay computed by a greedy algorithm that optimizes the cost, and improve up to 75.1 % upon the cost computed by a greedy algorithm that minimizes the delay.

Finally, the online taxipooling scheduler is introduced, which is a software component including a web application and a mobile application, which allows end-users to solve real instances of the taxi sharing problem in an efficient and friendly manner.

Keywords: ridesharing, evolutionary algorithms, optimization, public transportation

Índice general

1	Introducción	11
2	Definición del problema	15
2.1	El problema de viajes compartidos en taxis	15
2.1.1	Introducción	15
2.1.2	Formulación matemática	16
2.1.3	Caso de ejemplo	17
2.2	Variante multiobjetivo del problema de viajes compartidos en taxis	18
2.2.1	Introducción y motivación	19
2.2.2	Formulación matemática	19
2.3	Complejidad del problema	20
3	Algoritmos evolutivos	23
3.1	Algoritmos evolutivos	23
3.1.1	Introducción	23
3.1.2	Representación de la solución	24
3.1.3	Función de fitness	25
3.1.4	Operadores evolutivos	26
3.2	Algoritmos evolutivos paralelos	27
3.2.1	El algoritmo evolutivo paralelo con micro-población: $p\mu EA$	28
3.3	AE para problemas multiobjetivo	28
3.3.1	Problemas de optimización multiobjetivo	28
3.3.2	Algoritmos evolutivos para optimización multiobjetivo	30
4	Trabajos relacionados	33
4.1	Car Pooling Problem	33
4.2	Dial-a-Ride Problem	34
4.3	Taxi Pooling Problem	36
4.4	Resumen	39
5	Implementación	41
5.1	Bibliotecas de desarrollo	41
5.1.1	Mallba	41
5.1.2	ECJ	42
5.2	AE para el PVCT (versión monoobjetivo)	43
5.2.1	Aspectos comunes a ambos algoritmos	43
5.2.2	Algoritmo evolutivo secuencial: $seqEA$	48
5.2.3	Micro algoritmo evolutivo paralelo: $p\mu EA$	48

5.3	AE para el PVCT (versión multiobjetivo)	49
5.3.1	Aspectos comunes a ambos algoritmos	49
5.3.2	Micro algoritmo evolutivo paralelo multiobjetivo por descomposición de dominio: $p\mu MOEA/D$	51
5.3.3	Algoritmo evolutivo multiobjetivo explícito: <i>NSGA-II</i>	51
6	Evaluación Experimental	53
6.1	Generación de instancias de prueba	53
6.1.1	Proceso de generación de instancias	53
6.1.2	Instancias de prueba para la variante monoobjetivo	56
6.1.3	Instancias de prueba para la variante multiobjetivo	57
6.2	Metodología	58
6.3	Variante monoobjetivo del PVCT	59
6.3.1	Entorno de ejecución	59
6.3.2	Configuración paramétrica	59
6.3.3	Algoritmo ávido	60
6.3.4	Resultados numéricos	61
6.4	Variante multiobjetivo del PVCT	68
6.4.1	Entorno de ejecución	68
6.4.2	Configuración paramétrica	68
6.4.3	Métricas multiobjetivo	69
6.4.4	Algoritmos ávidos	70
6.4.5	Resultados numéricos	71
7	Planificador de viajes compartidos en línea	83
7.1	Descripción general del planificador	83
7.2	Arquitectura del sistema y tecnologías utilizadas	84
7.3	Aplicación web	86
7.4	Aplicación móvil	88
7.4.1	Prototipo basado en PhoneGap	88
7.4.2	Desarrollo Nativo	89
8	Conclusiones y trabajo futuro	91
8.1	Conclusiones	91
8.2	Trabajo futuro	93
Bibliografía		95
A Resultados experimentales		99
B Publicaciones		111

Índice de figuras

2.1 Caso de ejemplo con 1 origen y 11 destinos.	17
3.1 Codificación basada en permutaciones de enteros para el TSP.	25
3.2 Función de fitness aplicada sobre fenotipos.	25
3.3 Operador de cruzamiento basado en la posición, PBX.	26
3.4 Operador de mutación por intercambio, EM.	27
3.5 Estructura del algoritmo $p\mu EA$.	28
3.6 Soluciones dominadas, no dominadas y frente de Pareto en un MOP.	29
3.7 Propósitos de un AE para problemas de optimización multiobjetivo.	30
3.8 Esquema del algoritmo $p\mu MOEA/D$.	31
3.9 Mecanismo de asignacion de fitness en $NSGA-II$.	31
5.1 Ejemplo de representación de una solución.	44
5.2 Ejemplo de aplicación del cruzamiento PBX y de la función correctiva.	47
5.3 Ejemplo de la mutación EM y de la función correctiva.	47
6.1 Consultas a través de la API de Taxi Fare Finder.	55
6.2 Visualización de una instancia de prueba en Google Maps.	56
6.3 $seqEA$ —mejoras logradas sobre el algoritmo ávido.	64
6.4 $p\mu EA$ —mejoras logradas sobre el algoritmo ávido.	65
6.5 $seqEA$ y $p\mu EA$ —evolución del costo a lo largo de una ejecución.	68
6.6 $p\mu MOEA/D$ —frentes de Pareto por generación.	72
6.7 $p\mu MOEA/D$ —mejoras sobre los algoritmos ávidos en instancias chicas.	73
6.8 $p\mu MOEA/D$ —mejoras sobre los algoritmos ávidos en instancias medianas.	73
6.9 $p\mu MOEA/D$ —mejoras sobre los algoritmos ávidos en instancias grandes.	74
6.10 $p\mu MOEA/D$ —mejoras sobre los algoritmos ávidos en instancias Montevideo.	74
6.11 $NSGA-II$ —frentes de Pareto por generación.	76
6.12 $NSGA-II$ —mejoras sobre los algoritmos ávidos en instancias chicas.	77
6.13 $NSGA-II$ —mejoras sobre los algoritmos ávidos en instancias medianas.	77
6.14 $NSGA-II$ —mejoras sobre los algoritmos ávidos en instancias grandes.	78
6.15 $NSGA-II$ —mejoras sobre los algoritmos ávidos en instancias Montevideo.	78
6.16 $p\mu MOEA/D$ y $NSGA-II$ —mejora sobre algoritmos ávidos vs. tiempo	80
6.17 $p\mu MOEA/D$ vs. $NSGA-II$ —frentes de Pareto.	81
7.1 Ingreso de origen y destinos en el planificador	83
7.2 Visualización de resultados en el planificador de viajes compartidos en línea.	84
7.3 Arquitectura del planificador de viajes compartidos en línea.	85
7.4 Interfaz de la aplicación web del planificador	87
7.5 Interfaz de la aplicación móvil del planificador	89

Índice de tablas

2.1	Matriz de costos en pesos uruguayos para el caso de ejemplo. Valores correspondientes a julio de 2015.	18
4.1	Trabajos relacionados al problema de viajes compartidos en taxis.	39
6.1	Ejemplo de coordenadas para 10 pasajeros.	54
6.2	Tabla comparativa: <i>seqEA</i> con inicialización aleatoria vs. <i>seqEA</i> con inicialización ávida	62
6.3	Tabla comparativa: <i>pμEA</i> con inicialización aleatoria vs. <i>pμEA</i> con inicialización ávida.	62
6.4	Tabla comparativa <i>seqEA</i> vs. <i>pμEA</i>	66
6.5	Comparativa: mejoras y tiempos de ejecución entre <i>seqEA</i> y <i>pμEA</i>	67
6.6	Métricas multiobjetivo para <i>pμMOEA/D</i>	71
6.7	Métricas multiobjetivo para <i>NSGA-II</i>	75
6.8	Tabla comparativa: <i>pμMOEA/D</i> vs. <i>NSGA-II</i>	79
A.1	Comparativa: mejoras y tiempos de ejecución entre <i>seqEA</i> y <i>pμEA</i> para instancias chicas.	99
A.2	Comparativa: mejoras y tiempos de ejecución entre <i>seqEA</i> y <i>pμEA</i> para instancias medianas.	100
A.3	Comparativa: mejoras y tiempos de ejecución entre <i>seqEA</i> y <i>pμEA</i> para instancias grandes.	101
A.4	Comparativa: mejoras y tiempos de ejecución entre <i>seqEA</i> y <i>pμEA</i> para instancias de Montevideo.	102
A.5	Métricas multiobjetivo <i>pμMOEA/D</i> – instancias chicas.	102
A.6	Métricas multiobjetivo <i>pμMOEA/D</i> – instancias medianas.	103
A.7	Métricas multiobjetivo <i>pμMOEA/D</i> – instancias grandes.	103
A.8	Métricas multiobjetivo <i>pμMOEA/D</i> – instancias Montevideo.	104
A.9	Métricas multiobjetivo <i>NSGA-II</i> – instancias chicas.	104
A.10	Métricas multiobjetivo <i>NSGA-II</i> – instancias medianas.	105
A.11	Métricas multiobjetivo <i>NSGA-II</i> – instancias grandes.	105
A.12	Métricas multiobjetivo <i>NSGA-II</i> – instancias Montevideo.	106
A.13	Tabla comparativa: <i>pμMOEA/D</i> vs. <i>NSGA-II</i> – Instancias chicas.	107
A.14	Tabla comparativa: <i>pμMOEA/D</i> vs. <i>NSGA-II</i> – Instancias medianas.	108
A.15	Tabla comparativa: <i>pμMOEA/D</i> vs. <i>NSGA-II</i> – Instancias grandes.	109
A.16	Tabla comparativa: <i>pμMOEA/D</i> vs. <i>NSGA-II</i> – Instancias Montevideo.	110

Lista de Algoritmos

1	Funcionamiento de un algoritmo evolutivo.	24
2	Esquema de un AE paralelo de subpoblaciones distribuidas.	27
3	Esquema del algoritmo <i>NSGA-II</i>	32
4	Inicialización aleatoria de la población.	45
5	Inicialización ávida de la población.	45
6	Procedimiento de corrección de soluciones.	46
7	Cruzamiento basado en la posición (<i>PBX</i>).	46
8	Mutación por intercambio (<i>EM</i>).	47
9	Algoritmo ávido para minimizar costo en el PVCT monoobjetivo.	61
10	Algoritmo ávido para minimizar demora en el PVCT multiobjetivo.	71

Capítulo 1

Introducción

El concepto de ciudad inteligente (*smart city*) propone la utilización de tecnologías de la información y la comunicación con el fin de mejorar la calidad y el desempeño de los servicios urbanos, de forma de reducir costos, aumentar la eficiencia en el uso de recursos y permitir una participación más activa de los ciudadanos [16]. Este tipo de prácticas suelen aplicarse a los sistemas de transporte, debido al rol central que cumplen en el funcionamiento de las ciudades.

Bajo este paradigma, los viajes compartidos en automóvil (conocidos por el término anglosajón *car pooling*) han captado gran interés por parte del público en los últimos años. Compartir vehículos con personas que se dirigen hacia destinos cercanos genera beneficios tanto en el plano económico como en el ecológico, a niveles individuales y colectivos [22]. El principal beneficio destacado a nivel individual es el ahorro económico, producto de compartir los gastos con otros pasajeros. Sin embargo, los usuarios de sistemas de viajes compartidos destacan beneficios adicionales, tales como la oportunidad para socializar con otras personas, una mayor sensación de seguridad en el trayecto y una mayor puntualidad debido a compartir viajes con otros pasajeros [55]. Los beneficios de los viajes compartidos han dado lugar a diferentes iniciativas para atender el gran interés del público, entre las que se destacan los carriles exclusivos para viajes compartidos presentes en muchas ciudades [25], las campañas para compartir los viajes hacia y desde el lugar del trabajo [19] y una gran variedad de aplicaciones en línea desarrolladas para encontrar compañeros de viaje [7, 8].

Los taxis constituyen un medio de transporte rápido y confiable, especialmente en aquellas ciudades donde el sistema de transporte público es poco eficiente. Sin embargo, los taxis raramente viajan a capacidad completa, por lo que su impacto en la congestión del tráfico y en la contaminación de las ciudades suele ser importante. Asimismo, las tarifas pueden alcanzar valores considerables, que suelen desalentar la utilización de este medio de transporte por el público en general. Por estos motivos, resulta interesante estudiar la posibilidad de aplicar las ideas del car pooling a los viajes en taxis, surgiendo de esta manera el concepto de *taxis pooling*. En la actualidad existen algunas aplicaciones que permiten encontrar compañeros con los que compartir viajes en taxis [9, 51]. Sin embargo, la mayoría de las aplicaciones se limita a actuar como una simple cartelera donde los usuarios pueden buscar manualmente con quien compartir viaje o, en el mejor de los casos, utilizan algoritmos muy simples para sugerir compañeros de viaje en base a la cercanía de los orígenes y destinos.

En la ciudad de Montevideo, Uruguay, se encuentran registrados más de 3000 taxis [39], los cuales constituyen una parte central del sistema de transporte público. Sin embargo, los usuarios de taxis suelen quejarse de los altos costos que deben pagar por un viaje [21]. La posibilidad de reducir costos, compartiendo los viajes con otros pasajeros, puede estimular a los usuarios a considerar el uso del taxi como medio de transporte con mayor frecuencia. Esto resulta especialmente importante al definir políticas que buscan evitar la conducción bajo los efectos del alcohol. Según un informe de la Unidad Nacional de Seguridad Vial (UNASEV) [59], 6 % de los accidentes de tránsito con lesionados ocurridos en Uruguay en el año 2015 involucraron a un conductor bajo los efectos del alcohol. Este porcentaje aumenta al 7 % en el caso de los siniestros graves y al 15 % en el caso de los siniestros con resultados fatales. Que los usuarios consideren a los taxis como una alternativa dentro de sus posibilidades económicas, gracias a la capacidad de compartir sus viajes con otros pasajeros, puede resultar muy útil a la hora de desalentar a los conductores a utilizar sus vehículos particulares cuando planean consumir alcohol.

En los últimos años han surgido aplicaciones que permiten realizar viajes compartidos con otros pasajeros con el fin de reducir costos [40, 58]. Sin embargo, usualmente estos sistemas violan las normativas de las ciudades donde se instalan, ya que utilizan vehículos y conductores no habilitados para ofrecer como taxis. Esto supone un peligro para los pasajeros y para el tránsito en general, lo que ha dado lugar a numerosas manifestaciones en contra de este tipo de prácticas, que han llevado incluso a prohibir el uso de aplicaciones en línea para compartir vehículos particulares [56, 57]. En el marco de este proyecto se presenta una solución al problema de viajes compartidos basada en utilizar *la infraestructura de taxis existente en las ciudades*, de forma de conseguir un uso más eficiente del transporte, y *siempre respetando las normativas vigentes* respecto al transporte de pasajeros.

El proyecto que se describe en este informe tiene dos objetivos principales: por una parte, se busca estudiar el problema de planificación de viajes compartidos en taxis y diseñar algoritmos que lo resuelvan eficientemente; por otra parte, se busca implementar una solución que permita a usuarios finales resolver instancias reales del problema de una forma fácil e intuitiva. Siguiendo la línea marcada por estos objetivos, las principales contribuciones del trabajo son las siguientes:

1. el análisis de la literatura relacionada al problema de viajes compartidos, específicamente a los viajes realizados utilizando taxis como medio de transporte.
2. la definición y formulación de dos variantes del problema de viajes compartidos en taxis: i) una variante monoobjetivo del problema, donde se busca minimizar el costo total del grupo de pasajeros; ii) una variante multiobjetivo del problema, donde además de minimizar el costo, se plantea minimizar simultáneamente la demora percibida por cada usuario debido a compartir su viaje con otros pasajeros.
3. la implementación de **cuatro** algoritmos evolutivos que resuelven el problema de viajes compartidos en taxis (dos para cada una de las variantes del problema abordadas).
4. la generación de un conjunto de instancias realistas de prueba para el problema de viajes compartidos en taxis utilizando información real de bases de datos y repositorios públicos: un conjunto de **22** instancias para la variante monoobjetivo y un conjunto de **88** instancias para la variante multiobjetivo del problema, incluyendo instancias realistas de la ciudad de Montevideo.

-
5. la evaluación experimental de los algoritmos propuestos, comparándolos entre sí y contra heurísticas ávidas diseñadas en base a las ideas presentadas en trabajos de la literatura relacionada y siguiendo una estrategia intuitiva para resolver el problema de viajes compartidos en taxis.
 6. la implementación de un sitio web y de una aplicación móvil que permiten a los usuarios finales resolver instancias reales del problema de forma amigable.
 7. la publicación de cuatro artículos científicos en conferencias internacionales, enfocados en la descripción y evaluación de cada uno de los algoritmos evolutivos implementados.

El resto del documento se estructura del modo que se describe a continuación. En el Capítulo 2 se define el problema de viajes compartidos en taxis, en su variante monoobjetivo y en su variante multiobjetivo. Se presenta la formulación matemática de cada variante, un caso de ejemplo y por último se discute la complejidad del problema. El Capítulo 3 introduce la técnica utilizada para resolver el problema: los algoritmos evolutivos. Se presentan los conceptos generales asociados a esta técnica, así como los modelos paralelos y los detalles específicos al aplicarlos para resolver problemas de optimización multiobjetivo. El Capítulo 4 presenta una reseña de los principales trabajos relacionados con el problema de viajes compartidos. Los algoritmos evolutivos implementados para resolver el problema de viajes compartidos en taxis, en sus diferentes versiones, se presentan en el Capítulo 5. En el Capítulo 6 se describe la evaluación experimental realizada sobre los algoritmos implementados y se discuten los resultados alcanzados. En el Capítulo 7 se presenta el *planificador de viajes compartidos en línea*, una aplicación web/móvil que permite a los usuarios finales resolver instancias reales del problema de viajes compartidos en taxis. Por último, las conclusiones del proyecto y las principales líneas de trabajo futuro se presentan en el Capítulo 8.

Capítulo 2

Definición del problema

Este capítulo presenta el problema abordado en el marco del proyecto, que consiste en la planificación de viajes de un número de personas ubicadas en un mismo lugar de origen, que se dirigen hacia destinos potencialmente distintos, utilizando taxis de forma compartida. En la Sección 2.1 se presenta el problema en su formulación monoobjetivo, considerando el costo total del viaje como único objetivo a optimizar. La Sección 2.2 describe una variante multiobjetivo del problema, donde además de minimizar el costo se considera minimizar simultáneamente la demora percibida por los pasajeros para llegar a su destino. Por último, en la Sección 2.3 se estudia la complejidad del problema y se sugiere el camino a seguir para su resolución.

2.1. El problema de viajes compartidos en taxis

En esta sección se presenta el problema de viajes compartidos en taxis en su variante monoobjetivo, su formulación matemática y una instancia realista a modo de ejemplo.

2.1.1. Introducción

El problema a resolver modela la siguiente situación: un determinado número de personas, ubicadas en un mismo lugar de origen, deciden viajar hacia diferentes destinos utilizando taxis de forma compartida. El problema de optimización consiste en determinar el número apropiado de taxis, la asignación de pasajeros a taxis y las rutas que cada taxi debe realizar, de forma de minimizar el costo total del grupo de pasajeros.

La distribución de pasajeros está restringida a la cantidad máxima de personas que pueden viajar en un mismo taxi. Esta constante es parametrizable, permitiendo aplicar el problema a realidades diferentes de acuerdo a la legislación de cada ciudad. El problema fue originalmente concebido para la ciudad de Montevideo, Uruguay, aunque ni la formulación matemática ni los algoritmos desarrollados para su resolución restringen su aplicabilidad a otros escenarios.

Los costos asociados a cada viaje se modelan con una función realista basada en las distancias recorridas por cada taxi y que utiliza las tarifas específicas de la ciudad en la que se define cada instancia del problema. No se contemplan los costos asociados al tiempo del viaje, lo que equivale a considerar escenarios con tráfico fluido, sin grandes embotellamientos. La incorporación de datos de tráfico en tiempo real está propuesta como una de las principales líneas de trabajo futuro.

Las siguientes consideraciones deben ser tenidas en cuenta en el modelo del problema:

- cada taxi puede trasladar a un número limitado de pasajeros, dependiendo de las reglamentaciones propias de cada ciudad.
- el número máximo de taxis para N pasajeros es N , en el caso particular de que cada pasajero viaje en un vehículo separado.
- el costo de un taxi está dado por la suma del costo inicial —conocido popularmente como “bajada de bandera”— más el costo determinado por la distancia recorrida desde el origen hasta el destino final, pasando por cada uno de los destinos intermedios. No se consideran otros posibles costos relacionados con eventos tales como esperas, propinas o cargos extra por equipaje.

2.1.2. Formulación matemática

A continuación se presenta la formulación matemática para el problema de viajes compartidos en taxis (*PVCT*).

Dados los siguientes elementos:

- un conjunto de pasajeros $P = \{p_1, p_2, \dots, p_N\}$; que parten desde un punto geográfico común O y que desean trasladarse hacia un conjunto de destinos potencialmente diferentes $D = \{d_1, d_2, \dots, d_N\}$. La función $dest : P \rightarrow D$ establece el destino de cada uno de los pasajeros.
- un conjunto de taxis $T = \{t_1, t_2, \dots, t_M\}$; con $M \leq N$; y una función $C : T \rightarrow \{0, 1, \dots, C_{MAX}\}$ asociada a cada taxi, que indica la cantidad de pasajeros que viajan en él. C_{MAX} es la capacidad máxima permitida en un mismo taxi de acuerdo a las regulaciones propias de cada ciudad. El valor de C_{MAX} es fijo para todos los vehículos en esta formulación del problema (pero se considera variable en la variante multiobjetivo del problema descrita en la Sección 2.2.2).
- una constante B que indica el costo inicial del taxi (conocido popularmente como “bajada de bandera”).
- una función de distancia, $dist : \{\{O\} \cup D\} \times D \rightarrow \mathbb{R}_0^+$.
- una función de costo asociado a la distancia recorrida por cada taxi, $cost : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$.

El problema consiste en hallar una planificación, es decir una función $f : P \rightarrow T \times \{1, \dots, C_{MAX}\}$ para transportar los N pasajeros en K taxis ($K \leq N$) que determine la asignación de pasajeros a taxis y el orden en que serán trasladados a los respectivos destinos, minimizando la función de costo total (*CT*) de la asignación, dada en la Ecación 2.1, donde $f^{-1}(t_i, j)$ indica el pasajero asignado a la posición j (orden de traslado) en el taxi t_i y $dest(f^{-1}(t_i, 0)) = O$, $\forall t_i$.

$$CT = \sum_{t_i, C(t_i) \neq 0} \left[B + \sum_{j=1}^{C(t_i)} cost \left(\underbrace{dist \left(dest(f^{-1}(t_i, j-1)), dest(f^{-1}(t_i, j)) \right)}_{\text{destinos consecutivos en el recorrido del taxi } t_i} \right) \right] \quad (2.1)$$

El objetivo del problema consiste en optimizar el costo total del grupo de pasajeros. La formulación presentada no propone un mecanismo para distribuir el costo asociado al viaje entre los pasajeros que comparten un mismo taxi y se asume que la decisión es delegada a los propios usuarios.

Diversas estrategias pueden aplicarse para calcular el costo por pasajero [6]. Algunas de ellas necesitan conocer previamente el monto final a pagar y otras no. Las estrategias más intuitivas son: i) dividir el monto a pagar de forma equitativa entre los pasajeros de un taxi, sin tomar en cuenta las distancias recorridas por cada uno; ii) repartir el costo de bajada de bandera equitativamente y dividir el costo asociado a la distancia recorrida en cada tramo entre los pasajeros que recorrieron dicho tramo y iii) dividir el costo de la bajada de bandera proporcionalmente a la distancia recorrida por cada pasajero y el costo asociado a la distancia recorrida en cada tramo entre los pasajeros que recorrieron dicho tramo.

2.1.3. Caso de ejemplo

A continuación se describe una instancia de ejemplo del problema de planificación de viajes compartidos utilizando taxis. Considérese el mapa de la Figura 2.1 en el cual se destaca el origen común a todos los pasajeros (indicado con la etiqueta *Origen*) y un conjunto de 11 destinos $\{\#1, \#2, \dots, \#11\}$.

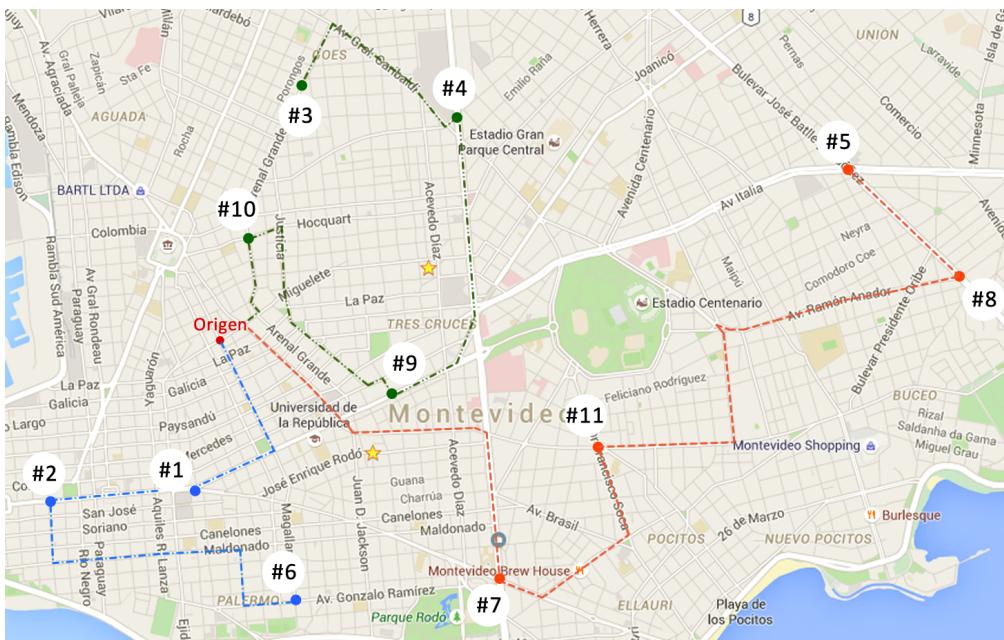


Figura 2.1: Caso de ejemplo con 1 origen y 11 destinos.

Los 11 pasajeros parten del origen común O y desean trasladarse hacia sus respectivos destinos utilizando taxis de forma compartida. Definiendo la máxima cantidad de pasajeros que pueden ser trasladados en un mismo taxi como 4 (lo usual para la ciudad de Montevideo), la solución al problema consiste en encontrar el número de taxis, la distribución de pasajeros a taxis y el recorrido de cada taxi, tal que se minimice el costo total.

Una posible distribución de pasajeros es la que se muestra en la Figura 2.1, donde se utilizan 3 taxis para trasladar a los 11 pasajeros. Un taxi traslada a los pasajeros #1,

#2 y #6 comenzando desde el origen y llevando a dichos pasajeros a sus respectivos destinos en el orden mencionado. Otro taxi traslada a los pasajeros #10, #9, #4 y #3 hacia sus destinos en ese orden y un último taxi traslada hacia sus respectivos destinos a los pasajeros #7, #11, #8 y #5 en dicho orden.

Dada la matriz de costos de la Tabla 2.1, y sabiendo que el costo de la *bajada de bandera* para la ciudad de Montevideo es de \$32.84 (valor correspondiente a julio de 2015), es posible calcular el costo total de la solución. El costo de cada taxi se detalla a continuación:

- **taxis rojos, línea con guiones:** (costo bandera) \$32.84 + (costo desde el origen al destino #7) \$59.2 + (costo desde el destino #7 al destino #11) \$30.6 + (costo desde el destino #11 al destino #8) \$59.2 + (costo desde el destino #8 al destino #5) \$22.9. **Total=\$204.74**
- **taxis azules, línea con guiones y puntos:** (costo bandera) \$32.84 + (costo desde el origen al destino #1) \$32.5 + (costo desde el destino #1 al destino #2) \$17.2 + (costo desde el destino #2 al destino #6) \$40.1. **Total=\$122.64**
- **taxis verdes, línea con guiones y dos puntos:** (costo bandera) \$32.84 + (costo desde el origen al destino #10) \$17.2 + (costo desde el destino #10 al destino #9) \$32.5 + (costo desde el destino #9 al destino #4) \$42.0 + (costo desde el destino #4 al destino #3) \$22.9. **Total=\$147.44**

El costo total de la solución es el resultado de la suma de los costos de cada uno de los taxis utilizados. En el caso ejemplo el costo total asciende a **\$474.82**.

	Origen	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11
Origen	0.0	32.5	45.8	38.2	59.2	110.8	43.9	59.2	112.7	28.7	17.2	63.0
#1	32.5	0.0	17.2	70.7	70.7	112.7	24.8	47.8	110.8	28.7	45.8	57.3
#2	45.8	17.2	0.0	76.4	95.5	128.0	40.1	66.9	126.0	43.9	51.6	80.2
#3	38.2	70.7	76.4	0.0	22.9	91.7	80.2	78.3	99.3	51.6	26.7	72.6
#4	59.2	70.7	95.6	22.9	0.0	70.7	86.0	57.3	78.3	42.0	38.2	57.3
#5	110.8	112.7	128.0	91.7	70.7	0.0	114.6	87.9	22.9	76.4	95.5	64.9
#6	43.9	24.8	40.1	80.2	86.0	114.6	0.0	40.1	116.5	38.2	51.6	53.5
#7	59.2	47.8	66.9	78.3	57.3	87.9	40.1	0.0	84.0	34.4	61.1	30.6
#8	112.7	110.8	126.1	99.3	78.3	22.9	116.5	84.0	0.0	78.3	107.0	59.2
#9	28.7	28.7	43.9	51.6	42.0	76.4	38.2	34.4	78.3	0.0	32.5	40.1
#10	17.2	45.8	51.6	26.7	38.2	95.5	51.6	61.1	107.0	32.5	0.0	74.5
#11	63.0	57.3	80.2	72.6	57.3	64.9	53.5	30.6	59.2	40.1	74.5	0.0

Tabla 2.1: Matriz de costos en pesos uruguayos para el caso de ejemplo. Valores correspondientes a julio de 2015.

2.2. Variante multiobjetivo del problema de viajes compartidos en taxis

Esta sección presenta el problema de viajes compartidos en taxis en su variante multiobjetivo, donde se busca minimizar simultáneamente el costo del grupo de pasajeros y la demora percibida por cada uno de ellos en llegar a su destino.

2.2.1. Introducción y motivación

Minimizar costos es un claro objetivo desde el punto de vista del usuario al compartir vehículos con otras personas. Sin embargo, la decisión de un usuario puede estar condicionada a la demora que debe experimentar por compartir el viaje, frente a la alternativa de viajar solo en un taxi. Por tal motivo, es de interés estudiar la variante multiobjetivo del problema de viajes compartidos en taxis, donde se propone minimizar simultáneamente el costo del grupo de usuarios y la demora percibida por cada uno de ellos para llegar a su destino.

La variante multiobjetivo del problema modela una realidad con las mismas características que las del problema en su formulación monoobjetivo, pero adicionando un “nivel de apuro” asociado a cada pasajero. El nivel de apuro denota la demora que está dispuesto a tolerar un usuario por compartir su viaje, respecto al tiempo que demoraría si optara por un viaje directo desde el origen hasta su destino (sin compartir el viaje con otros pasajeros). El problema de optimización consiste en determinar el número apropiado de taxis y la distribución adecuada de pasajeros, buscando minimizar dos objetivos simultáneamente: el *costo del viaje* y la *demora percibida* por los pasajeros. La distribución de pasajeros está restringida a la cantidad máxima de personas que pueden viajar en un mismo taxi. En esta variante del problema se contemplan vehículos de distintas capacidades, aportando mayor realismo a la formulación. Al tomar en cuenta las demoras percibidas por cada pasajero, puede ser útil contar con vehículos de distinta capacidad al momento de definir el número de taxis a utilizar y los pasajeros que viajarán en cada taxi.

Además de las consideraciones mencionadas en la Sección 2.1.1, para la variante multiobjetivo se establece que:

- cada taxi puede trasladar a un número limitado de pasajeros dependiendo de su capacidad. La cantidad de taxis disponibles de cada capacidad es dada en la instancia del problema a resolver. Se asume que existen suficientes vehículos disponibles para trasladar al grupo de pasajeros.
- el número máximo de taxis a utilizar para un escenario con N pasajeros es N , en el caso particular de que cada pasajero viaje en un vehículo distinto.
- cada pasajero tiene asociada una tolerancia que indica el tiempo que está dispuesto a demorar por sobre el tiempo que le insumiría un viaje directo desde el origen hasta su destino. La tolerancia de cada pasajero también es parte de los datos de entrada de la instancia del problema a resolver.

2.2.2. Formulación matemática

La formulación matemática del problema de viajes compartidos en taxis en su variante multiobjetivo se presenta a continuación.

Dados los siguientes elementos:

- un conjunto de pasajeros $P = \{p_1, p_2, \dots, p_N\}$; que parten desde un punto geográfico común O y que desean trasladarse hacia un conjunto de destinos potencialmente diferentes $D = \{d_1, d_2, \dots, d_N\}$. La función $dest : P \rightarrow D$ establece el destino de cada uno de los pasajeros.

- un conjunto de taxis $T = \{t_1, t_2, \dots, t_M\}$; con $M \leq N$, con capacidades (pasajeros que pueden transportar) $K = \{K_1, K_2, \dots, K_M\}$.
- una función $C : T \rightarrow \mathbb{N}_0$ que determina cuántos pasajeros hacen uso de un taxi en un determinado viaje, siendo $C(t_i) \leq K_i$.
- una constante B que indica el costo inicial del taxi (conocido popularmente como “bajada de bandera”).
- una función de distancia, $dist : \{\{O\} \cup D\} \times D \rightarrow \mathbb{R}_0^+$.
- una función de costo asociado a la distancia recorrida por cada taxi, $cost : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$.
- una función de tiempo de recorrido, $time : \{\{O\} \cup D\} \times D \rightarrow \mathbb{R}_0^+$.
- una función $tol : P \rightarrow \mathbb{R}_0^+$ que establece el tiempo adicional que está dispuesto a tolerar cada pasajero, por encima del tiempo que demora en viajar directamente desde el origen O a su destino.

El problema consiste en hallar una *planificación de viajes* para transportar los N pasajeros en L taxis ($L \leq N$), es decir una función $f : P \rightarrow T \times \mathbb{N}$, donde $\forall (t_i, h)/f(p_j) = (t_i, h)$ se cumple $h \leq C(t_i)$. La función f asigna pasajeros a taxis y determina el orden en que serán trasladados a sus respectivos destinos, minimizando simultáneamente el costo total dado por la función CT (Ecación 2.2) y la demora total dada por la función DT (Ecación 2.3), donde $f^{-1}(t_i, j)$ indica el pasajero asignado a la posición j (orden de traslado) en el taxi t_i y $dest(f^{-1}(t_i, 0)) = O$, $\forall t_i$.

$$CT = \sum_{t_i, C(t_i) \neq 0} \left[B + \sum_{j=1}^{C(t_i)} \underbrace{cost \left(dist \left(\overbrace{dest(f^{-1}(t_i, j-1)), dest(f^{-1}(t_i, j))}^{\text{destinos consecutivos en el recorrido del taxi } t_i} \right) \right)}_{\text{destinos consecutivos en el recorrido del taxi } t_i} \right] \quad (2.2)$$

$$DT = \sum_{t_i} \left[\sum_{j=1}^{C(t_i)} \left[\underbrace{\sum_{h=1}^j time \left(dest(f^{-1}(t_i, h-1)), dest(f^{-1}(t_i, h)) \right)}_{\text{tiempo efectivo de traslado del pasajero en la posición } j \text{ del taxi } t_i} \right. \right. \\ \left. \left. - \underbrace{tol(f^{-1}(t_i, j)) + time(O, dest(f^{-1}(t_i, j)))}_{\text{tiempo tolerado por el pasajero en la posición } j \text{ del taxi } t_i} \right] \right] \quad (2.3)$$

2.3. Complejidad del problema

El problema de planificación de viajes compartidos en taxis (y por extensión, su variante multiobjetivo) tiene varios puntos en común con dos conocidos problemas de la literatura relacionada: el problema genérico de viajes compartidos en autos (*Car Pooling Problem, CPP*) y el problema de rutas de vehículos (*Vehicle Routing Problem, VRP*).

Baldacci et al. [4] estudiaron una variante del *CPP* que modela la realidad de una empresa que busca motivar a sus empleados para compartir vehículos hacia y desde el

lugar de trabajo. Se estudia el caso del *CPP hacia el trabajo*, es decir desde muchos orígenes hacia un mismo destino. En la variante estudiada se conocen de antemano los vehículos disponibles, los empleados que van a actuar como conductores y los que van a actuar como pasajeros. Para la resolución del problema se propone un método exacto que combina una relajación lagrangeana y el método de generación de columnas.

El *CPP* estudiado por Baldacci et al. es un caso particular del *VRP* con demanda unitaria estudiado por Letchford et al. [37]. En este problema se cuenta con un depósito central desde donde parten todos los vehículos, los cuales tienen una capacidad máxima asociada. La formulación del problema propone encontrar las rutas que minimicen el costo total al visitar a un determinado número de clientes. Letchford et al. probaron que esta variante del *VRP* es \mathcal{NP} -difícil.

A pesar de estudiar la versión donde los vehículos parten desde muchos orígenes y viajan hacia un mismo destino, Baldacci et al. mencionan que la misma asignación puede ser utilizada para el problema inverso: desde un origen común (el lugar de trabajo) hacia varios destinos. Esta variante del problema tiene grandes similitudes con el problema de viajes compartidos en taxis estudiado en este proyecto:

- los empleados que parten desde su lugar de trabajo en el *CPP* se corresponden con los pasajeros que parten desde un origen común en la formulación de nuestro problema.
- los vehículos que utilizan los empleados para dirigirse hacia sus hogares en el *CPP* se corresponden con los taxis que utilizan los pasajeros para dirigirse hacia sus destinos.
- una solución al *CPP* debe cumplir que el último pasajero de un determinado vehículo sea su dueño. Esta restricción no aplica en el caso de los taxis, por lo que el espacio de soluciones de nuestro problema es aún mayor que el del *CPP*.
- el *CPP* establece tiempos máximos de llegada al destino para cada pasajero. Estos tiempos son equivalentes a la tolerancia por pasajero introducida en la variante multiobjetivo del problema de viajes compartidos en taxis.
- el *CPP* admite vehículos de diferentes capacidades, lo cual es incorporado en la formulación de la variante multiobjetivo del problema de viajes compartidos en taxis.

De acuerdo a las analogías presentadas previamente, es posible afirmar que el problema de viajes compartidos en taxis, tanto en su formulación monoobjetivo como en su variante multiobjetivo, tiene una complejidad similar al *CPP* estudiado por Baldacci et al.

Tomando en cuenta la complejidad del problema, cuando se utilizan instancias de tamaños realistas, los algoritmos exactos tradicionales no resultan útiles para una planificación eficiente. En particular, no son aplicables para la planificación en línea, que es necesaria para proveer una buena experiencia de usuario en aplicaciones web o móviles, como la que se presenta en el Capítulo 7. En estos casos es necesario utilizar heurísticas y metaheurísticas que permitan calcular soluciones de calidad aceptable en tiempos razonables [47].

Capítulo 3

Algoritmos evolutivos

El presente capítulo describe la técnica utilizada para resolver el problema de viajes compartidos en taxis, tanto en su formulación monoobjetivo como en su variante multiobjetivo. En la Sección 3.1 se introducen los algoritmos evolutivos como técnicas para resolver problemas de optimización y se describen los principales conceptos asociados a estos métodos. La Sección 3.2 presenta la utilización de técnicas de computación paralela en el diseño de algoritmos evolutivos. Por último, la Sección 3.3 detalla las particularidades de los algoritmos evolutivos aplicados a problemas de optimización multiobjetivo.

3.1. Algoritmos evolutivos

En esta sección se presentan los algoritmos evolutivos como técnica para resolver problemas de optimización y se detallan los conceptos fundamentales de esta técnica, a los cuáles se hará referencia frecuentemente a lo largo del documento.

3.1.1. Introducción

Al resolver problemas de optimización \mathcal{NP} -difíciles (como el que se aborda en este proyecto) puede que no exista un algoritmo que garantice encontrar la solución óptima en un tiempo aceptable. Los métodos de resolución exactos (e.g., programación lineal, programación dinámica) no son capaces de resolver eficientemente instancias de dimensiones grandes para este tipo de problemas, debido a que su complejidad algorítmica crece de forma superpolinomial con respecto al tamaño del problema, demandando una enorme capacidad de cómputo y un largo tiempo de ejecución. Esto ha llevado a la aplicación de técnicas heurísticas y metaheurísticas, que si bien no garantizan encontrar la solución óptima al problema a resolver, suelen ser capaces de encontrar soluciones de buena calidad en tiempos de ejecución aceptables [26].

Los *algoritmos evolutivos* (AE) son técnicas estocásticas que emulan el proceso de evolución natural de las especies para resolver problemas de optimización, búsqueda y aprendizaje [3]. La primer propuesta de un algoritmo basado en el proceso de evolución natural se remonta a 1975, cuando Holland propuso un procedimiento de búsqueda genética y sentó las bases del área de la computación evolutiva [29]. Los AE se han popularizado en los últimos 30 años, siendo exitosamente aplicados para resolver problemas de optimización subyacentes a problemas complejos del mundo real en múltiples áreas de aplicación [46].

Un AE es una técnica iterativa (cada iteración se denomina *generación*) que aplica operadores estocásticos sobre un conjunto de individuos (la *población*). Inicialmente, la población se genera a través de un procedimiento aleatorio o utilizando una heurística específica para el problema a resolver. Cada individuo en la población codifica una solución tentativa al problema y tiene un valor de *fitness*, dado por una función de evaluación que determina su adecuación para resolver el problema. El propósito del AE es mejorar el fitness de los individuos en la población. Este propósito se logra mediante la aplicación iterativa de *operadores evolutivos*, tales como la *recombinación* de partes de dos individuos y la *mutación* aleatoria de su codificación. Estos operadores son aplicados a individuos seleccionados según su fitness, guiando al AE hacia soluciones tentativas de mayor calidad. El funcionamiento básico de un AE se presenta en el Algoritmo 1.

Algoritmo 1 Funcionamiento de un algoritmo evolutivo.

```

1: inicializar( $P(0)$ )
2:  $t \leftarrow 0$  {contador de generación}
3: mientras no se cumpla el criterio de parada hacer
4:   evaluar( $P(t)$ )
5:   padres  $\leftarrow$  selección( $P(t)$ )
6:   hijos  $\leftarrow$  operadores evolutivos(padres)
7:   nueva población  $\leftarrow$  reemplazo(hijos,  $P(t)$ )
8:    $t++$ 
9:    $P(t) \leftarrow$  nueva población
10: fin mientras
11: retornar mejor individuo hallado
  
```

El criterio de parada de la iteración usualmente involucra un número determinado de generaciones, una cota de calidad sobre el mejor valor de fitness hallado, o la detección de una situación de convergencia. Políticas específicas se utilizan para seleccionar el grupo de individuos para participar en la recombinación (la *selección*) y para determinar cuáles nuevos individuos serán insertados en la población en cada nueva generación (el *reemplazo*). Finalmente, el AE retorna la mejor solución hallada en el proceso iterativo, tomando en cuenta la función de fitness considerada para el problema.

3.1.2. Representación de la solución

Al momento de trabajar sobre un determinado problema, los AE hacen uso de una abstracción de las soluciones tentativas denominada *cromosoma*. Un cromosoma es un vector de *genes* que representan características o variables en la solución codificada. Los genes tienen valores asignados denominados *alelos*. Un *genotipo* denota al conjunto de cromosomas que definen las características de un individuo, pero dado que se suele utilizar un único cromosoma por individuo, los términos genotipo, cromosoma e individuo se suelen utilizar indistintamente. Por otra parte, el *fenotipo* representa un punto del espacio de soluciones del problema (i.e., la expresión del genotipo en el dominio del problema).

Es necesario definir un proceso de codificación y decodificación que permita transformar fenotipos en genotipos, es decir, transformar las soluciones del dominio del problema en individuos del AE, y viceversa. El mecanismo de codificación de individuos es un punto clave al momento de desarrollar un AE, ya que distintas codificaciones se pueden

adaptar con mayor o menor facilidad a la realidad propia del problema que se busca resolver. A modo de ejemplo, se presenta a continuación el esquema de codificación basado en *permutaciones de enteros*, el cual es útil para problemas de optimización que buscan hallar ordenamientos óptimos como los problemas de planificación de viajes compartidos que se resuelven en este proyecto. La Figura 3.1 muestra un ejemplo para el reconocido Problema del Agente Viajero (*Traveling Salesman Problem, TSP*), donde se muestra el proceso de codificación y decodificación de soluciones.

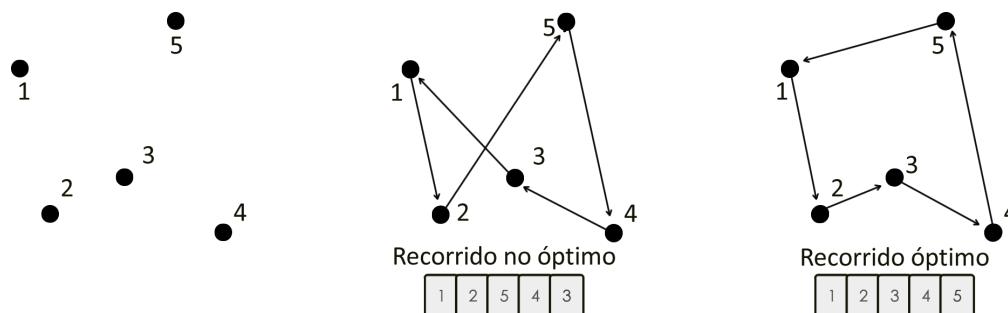


Figura 3.1: Codificación basada en permutaciones de enteros para el TSP.

Esta codificación utiliza vectores de enteros para representar a los individuos, donde los enteros representan las ciudades que debe visitar el agente viajero. El orden de los enteros en la codificación es relevante pues determina el recorrido a realizar. Además, las restricciones del problema establecen que cada ciudad debe ser visitada una única vez, por lo que cada valor entero debe aparecer una única vez en el genotipo.

3.1.3. Función de fitness

Todo individuo en un AE tiene asociado un valor de fitness que indica su grado de aptitud para resolver el problema. El valor de fitness es calculado por la función de fitness, la cual asigna un valor a cada uno de los individuos de un problema evaluando su fenotipo, tal como se muestra en la Figura 3.2.

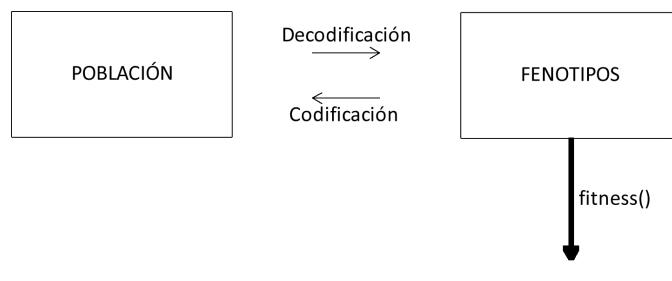


Figura 3.2: Función de fitness aplicada sobre fenotipos.

La función de fitness actúa como el *entorno* en el proceso evolutivo, indicando cuáles son los mejores individuos de la población y por lo tanto es la principal herramienta para guiar al AE hacia soluciones de calidad. La función de fitness está fuertemente ligada a la función objetivo del problema de optimización que se desea resolver. Si bien los AE pueden ser utilizados tanto para problemas que busquen maximizar como minimizar la función objetivo, se suele aplicar una transformación de forma que el valor de fitness siempre sea maximizado (para reflejar aptitud en los individuos y seguir con la analogía

de la evolución natural). Debido al proceso de codificación, es posible que un determinado genotipo tenga un fenotipo asociado que no represente una solución factible en el dominio del problema. En estos casos existen tres posibles caminos a seguir ante la aparición de individuos con estas características: aplicar un mecanismo de corrección que los transforme en una solución válida, asignarles un valor de fitness adecuado de forma de reducir la posibilidad de que sobrevivan al proceso evolutivo, o descartarlos.

3.1.4. Operadores evolutivos

En su variante más elemental, un AE cuenta con operadores de *selección, recombinación y mutación*.

Selección: es el operador encargado de asignar mayor oportunidad de sobrevivir a los buenos individuos a lo largo del proceso evolutivo. Generalmente, los individuos con mayor valor de fitness (i.e., los más aptos) tienen mayor probabilidad de ser seleccionados, aunque esto depende de la metodología de selección utilizada. Múltiples estrategias de selección se han propuesto en la literatura [3]. Se presentan a continuación los dos mecanismos de selección empleados en los AE implementados en este proyecto a modo de ejemplo:

- *selección proporcional*: que asigna a cada individuo de la población una probabilidad de selección dada por el cociente entre su valor de fitness y la suma del fitness del total de los individuos en la población.
- *selección por torneo*: donde se sortean m individuos de la población y se eligen los k individuos con mayor valor de fitness (usualmente $k = 1$).

Recombinación: es un operador que se aplica sobre dos cromosomas para generar descendientes con las características de ambos cromosomas padres. Este operador es análogo a la reproducción entre individuos de una especie en el proceso evolutivo biológico. Se suelen utilizar distintos operadores de recombinación de acuerdo a la codificación de las soluciones y a características propias del problema a resolver. A modo de ejemplo, se presenta en la Figura 3.3 el cruzamiento basado en la posición (*Position Based Crossover, PBX*), utilizado para representaciones de permutaciones de enteros, que será utilizado en la resolución del problema de viajes compartidos abordado en este proyecto. Este operador selecciona algunos genes del primer parente, los cuales se copian directamente al hijo y se marcan como “usados” en el segundo parente. Luego, se recorre el segundo parente de izquierda a derecha y se copian los genes no marcados hacia las posiciones libres en el hijo.

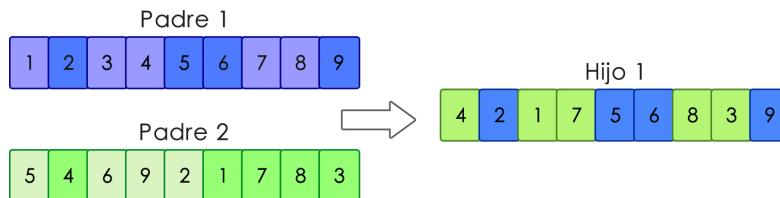


Figura 3.3: Operador de cruzamiento basado en la posición, PBX.

Mutación: este operador modifica al azar parte del cromosoma de los individuos, introduciendo diversidad en el proceso evolutivo. De esta forma evita que el AE se estanque en un sector del espacio de soluciones y permite que la búsqueda alcance zonas del espacio de soluciones que no estaban cubiertas por los individuos de la población actual. En la Figura 3.4 se presenta a modo de ejemplo el operador de mutación por intercambio (*Exchange Mutation, EM*), el cual sortea dos posiciones del cromosoma e intercambia los valores en ellas. Este operador será utilizado en la resolución del problema de viajes compartidos en taxis.

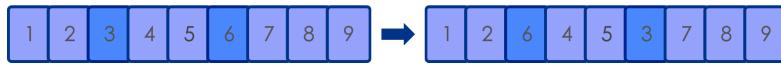


Figura 3.4: Operador de mutación por intercambio, EM.

3.2. Algoritmos evolutivos paralelos

Las implementaciones paralelas se han popularizado como un mecanismo para mejorar el desempeño de los AE. Dividiendo la población o el cálculo de fitness entre varios elementos de procesamiento, los AE paralelos permiten abordar problemas de grandes dimensiones y difíciles de resolver, hallando soluciones de calidad en tiempos razonables.

Entre los modelos paralelos de AE se destaca el de subpoblaciones distribuidas [2], donde se divide la población original en varias subpoblaciones (islas). Cada isla ejecuta un AE secuencial, donde los individuos solamente son capaces de interactuar con otros en su misma isla. Se define un operador evolutivo adicional llamado *migración*, que permite el intercambio ocasional de individuos entre islas, introduciendo una nueva fuente de diversidad. El Algoritmo 2 presenta el esquema de un AE paralelo de subpoblaciones distribuidas, donde *emigrantes* denota al conjunto de individuos a intercambiar con otra isla, seleccionados de acuerdo a una política determinada por *seleccMigracion*. El operador de migración intercambia los individuos entre islas de acuerdo a una topología de interconexión, que generalmente es un anillo unidireccional. La *CondicionMigracion* determina cuándo se lleva a cabo el intercambio de individuos.

Algoritmo 2 Esquema de un AE paralelo de subpoblaciones distribuidas.

```

1: inicializar( $P(0)$ )
2:  $t \leftarrow 0$  {contador de generación}
3: evaluar( $P(0)$ )
4: mientras no se cumple CriterioParada hacer
5:   padres  $\leftarrow$  selección( $P(t)$ )
6:   hijos  $\leftarrow$  operadores evolutivos(padres)
7:   nueva población  $\leftarrow$  reemplazo(hijos,  $P(t)$ )
8:   evaluar( $P(t)$ )
9:    $t++$ 
10:   $P(t) \leftarrow$  nueva población
11:  si CondicionMigración entonces
12:    emigrantes  $\leftarrow$  seleccMigración( $P(t)$ )
13:    inmigrantes  $\leftarrow$  migración(emigrantes)
14:    insertar(inmigrantes,  $P(t)$ )
15:  fin si
16: fin mientras
17: retornar mejor solución hallada

```

3.2.1. El algoritmo evolutivo paralelo con micro-población: $p\mu EA$

Los AE paralelos mejoran su eficiencia computacional al trabajar con subpoblaciones que limitan las interacciones entre individuos. Sin embargo, estos algoritmos suelen perder diversidad, convergiendo prematuramente a soluciones sub-óptimas del problema.

Varias alternativas se han propuesto para mitigar el efecto de la pérdida de diversidad en los AE. Una estrategia difundida en los algoritmos evolutivos paralelos es la de los micro algoritmos evolutivos paralelos ($p\mu EA$), que intenta mitigar la pérdida de diversidad haciendo uso de poblaciones pequeñas e incluyendo un operador específico de diversidad. La idea de un micro algoritmo genético fue presentada por Coello y Pulido [13], basándose en los estudios teóricos de Goldberg sobre la convergencia al óptimo de un problema de optimización cuando se utiliza un AE con tan solo tres individuos en la población, en caso de contar con un operador apropiado para introducir diversidad ante una situación de convergencia prematura. La Figura 3.5 muestra el esquema utilizado por el $p\mu EA$, donde el operador de migración de individuos entre las micro-poblaciones actúa como operador de diversidad. El algoritmo $p\mu EA$ se utiliza en este trabajo para resolver la formulación monoobjetivo del problema de viajes compartidos en taxis.

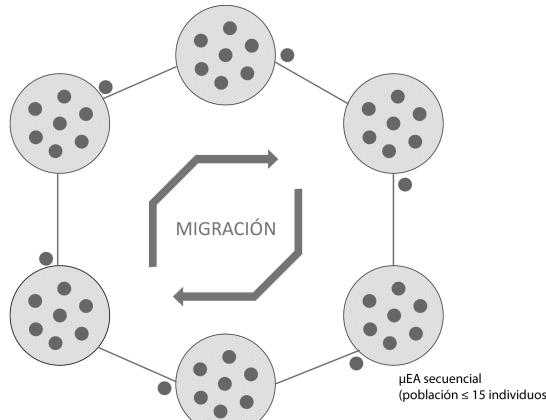


Figura 3.5: Estructura del algoritmo $p\mu EA$.

3.3. Algoritmos evolutivos para problemas multiobjetivo

En esta sección se presentan los algoritmos evolutivos aplicados a problemas de optimización multiobjetivo. Inicialmente se formaliza la noción de problema multiobjetivo y luego se presentan los dos algoritmos evolutivos empleados en este trabajo para resolver la variante multiobjetivo del problema de viajes compartidos en taxis.

3.3.1. Problemas de optimización multiobjetivo

Como su nombre indica, los problemas de optimización multiobjetivo (*Multiobjective optimization problem, MOP*) son aquellos que tratan con más de una función objetivo, habitualmente en conflicto entre sí. A diferencia de la optimización monoobjetivo, donde existe una única solución al problema, en optimización multiobjetivo se busca hallar un conjunto de soluciones que plantearán diferentes compromisos entre los valores de las funciones a optimizar. La mayoría de problemas de toma de decisiones suelen ser multiobjetivo. En su forma más general, un problema multiobjetivo tiene un cierto número de

funciones objetivo que se desean minimizar o maximizar y un conjunto de restricciones que cualquier solución debe cumplir [17]. La Ecuación 3.1 presenta la formulación matemática para un problema de optimización multiobjetivo genérico, donde una solución x es un vector de n variables de decisión $x = (x_1, x_2, \dots, x_n)^T$.

$$\begin{aligned} & \text{minimizar/maximizar} && f_m(x), m = 1, 2, \dots, M; \\ & \text{sujeto a} && g_j(x) \geq 0, j = 1, 2, \dots, J; \\ & && h_k(x) = 0, k = 1, 2, \dots, K; \\ & && x_i^{(L)} \leq x_i \leq x_i^{(U)}, i = 1, 2, \dots, n. \end{aligned} \quad (3.1)$$

La mayoría de los algoritmos para optimización multiobjetivo utilizan el concepto de *dominancia*. Se dice que una solución x^1 domina a otra solución x^2 si se cumplen las condiciones: a) x^1 no es peor que x^2 en todos los objetivos, y b) x^1 es estrictamente mejor que x^2 en al menos un objetivo.

Es intuitivo que si una solución x^1 domina a otra solución x^2 , entonces x^1 es mejor solución que x^2 para el problema multiobjetivo que se está resolviendo. Por esta razón, muchos algoritmos multiobjetivo utilizan el concepto de dominancia como método para comparar soluciones en busca de soluciones no dominadas de buena calidad. Si al momento de comparar dos soluciones la condición a) no se satisface en ninguno de los dos sentidos (i.e., x^1 supera a x^2 en el objetivo m , pero x^2 supera a x^1 en el objetivo n), no se puede establecer una relación de dominancia entre dichas soluciones.

Para un conjunto finito de soluciones es posible realizar todas las comparaciones entre ellas, de forma de establecer cuál solución domina a cuál y cuáles no son dominadas entre sí. Finalmente, es posible obtener un conjunto de soluciones en el cual no existe un par de soluciones donde una domine a la otra. Este conjunto se denomina *conjunto de soluciones no dominadas* y cumple que, para toda solución que no pertenezca al mismo, existe una solución “mejor” (i.e., que la domina) perteneciente al conjunto. El conjunto de soluciones no dominadas del espacio de soluciones entero se denomina *conjunto óptimo de Pareto*. La región de puntos definida por el conjunto óptimo de Pareto en el espacio de valores de las funciones objetivo se conoce como *frente de Pareto*. La Figura 3.6 ilustra los conceptos de soluciones no dominadas, soluciones dominadas y frente de Pareto, para un problema de optimización donde se busca minimizar dos funciones objetivo.

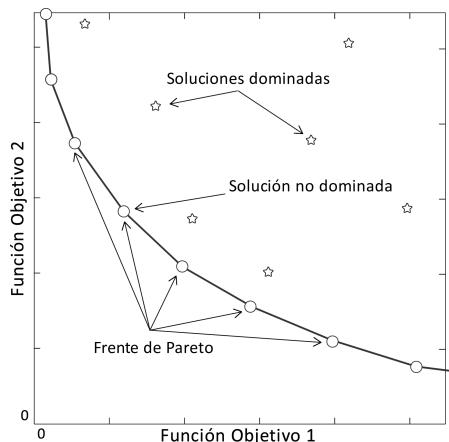


Figura 3.6: Soluciones dominadas, no dominadas y frente de Pareto en un MOP.

3.3.2. Algoritmos evolutivos para optimización multiobjetivo

A diferencia de los métodos tradicionales para resolver problemas de optimización, que trabajan con una única solución, los AE consideran un conjunto de soluciones. Esto hace que los AE sean útiles para problemas multiobjetivo, hallando en cada ejecución un conjunto de soluciones que aproximen el frente de Pareto del problema. Además, mediante la aplicación de operadores evolutivos estocásticos, los AE suelen ser menos sensibles que otros métodos de búsqueda a la forma del frente de Pareto del problema [17].

Un algoritmo evolutivo para optimización multiobjetivo (*Multiobjective evolutionary algorithm, MOEA*) debe lograr dos propósitos simultáneamente: acercarse al frente de Pareto del problema (*convergencia*) y muestrear adecuadamente el frente de soluciones sin converger a una solución única o a una sección específica del frente de Pareto (*diversidad*). La Figura 3.7 muestra gráficamente estos dos propósitos que deben ser tenidos en cuenta en el diseño de un MOEA. La convergencia se logra gracias al proceso evolutivo del MOEA, mientras que para mantener la diversidad es necesario agregar técnicas específicas, utilizadas también en la optimización multimodal (e.g., *crowding, sharing*).

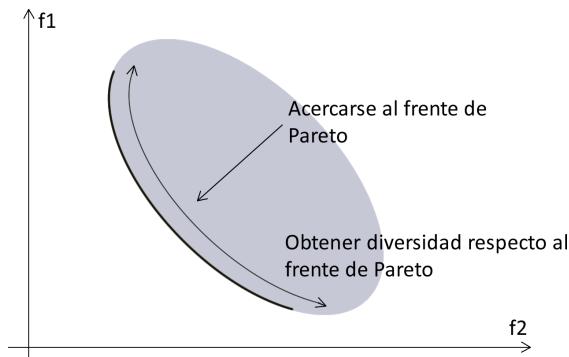


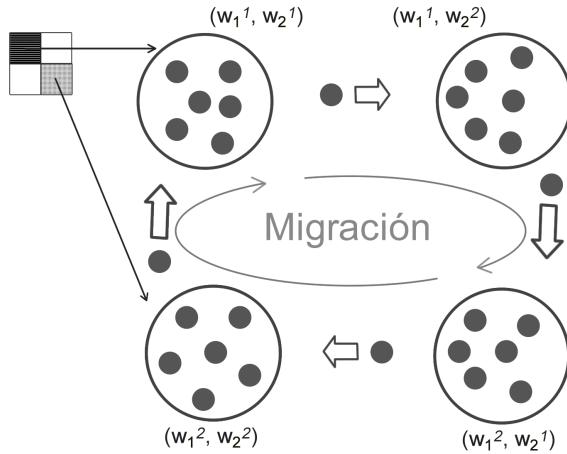
Figura 3.7: Propósitos de un AE para problemas de optimización multiobjetivo.

Los MOEA utilizados en este proyecto para la resolución de la variante multiobjetivo del problema de viajes compartidos en taxis se describen a continuación.

El algoritmo $p\mu MOEA/D$

El algoritmo $p\mu MOEA/D$ combina dos características de diseño: i) un modelo paralelo de subpoblaciones distribuidas utilizando micropoblaciones y un operador de migración para proveer diversidad a la búsqueda y ii) un enfoque de descomposición de dominio similar al aplicado en *MOEA/D* [62] para resolver un problema de optimización con múltiples objetivos. Cada subpoblación en $p\mu MOEA/D$ se enfoca en resolver un problema de optimización específico, aplicando un esquema de agregación lineal para los objetivos del problema, pero utilizando diferentes pesos para ponderar las funciones objetivo.

En el caso de un problema donde se buscan optimizar dos funciones objetivo f_1 y f_2 , la función de fitness queda definida por la suma ponderada de las funciones objetivo del problema, discretizando los pesos en una grilla de dos dimensiones: $F_{i,j} = w_1^i \times f_1 + w_2^j \times f_2$, donde $i \in \{1 \dots M\}$, $j \in \{1 \dots N\}$. Si se utiliza una grilla cuadrada y se distribuyen los pesos uniformemente entre las islas, el algoritmo utiliza $M = N$ islas. La Figura 3.8 muestra el esquema del algoritmo $p\mu MOEA/D$ para un problema de dos objetivos, donde se utiliza una grilla cuadrada para definir los pesos.

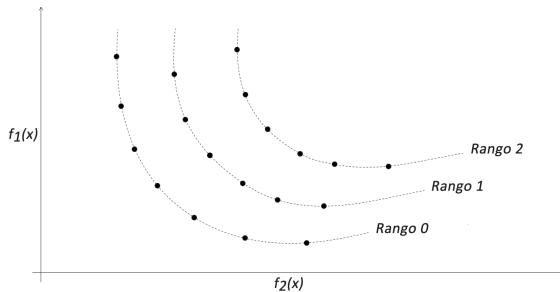
Figura 3.8: Esquema del algoritmo *pμMOEA/D*.

El enfoque de agregación lineal y descomposición de dominio aplicado en *pμMOEA/D* permite muestrear el frente de Pareto del problema de optimización, a pesar de que el algoritmo no utilice un esquema explícito de asignación de fitness basado en dominancia de Pareto. El enfoque ha sido aplicado de forma exitosa al algoritmo CHC [48], y se aplica concretamente a un AE genérico en el marco de este proyecto.

El algoritmo *NSGA-II*

Uno de los algoritmos multiobjetivo más populares es el llamado *Non-dominated Sorting Genetic Algorithm, versión II (NSGA-II)* propuesto por Deb et al. [18]. *NSGA-II* ofrece una búsqueda evolutiva mejorada respecto a su predecesor *NSGA*, basándose en tres aspectos: i) un ordenamiento no-dominado, elitista, que disminuye la complejidad asociada al chequeo de dominancia; ii) una técnica de *crowding* para preservar la diversidad de la población que, a diferencia de la técnica de *sharing* usada en su predecesor, no requiere parámetros adicionales; iii) un método de asignación de fitness que considera los valores de distancia de *crowding* para evaluar la diversidad de las soluciones.

NSGA-II clasifica a los individuos de la población en distintos niveles (llamados *rangos*), formados por las soluciones no dominadas, siendo 0 el mejor rango (definido por el conjunto de soluciones no dominadas de la población). Dentro de cada rango, las soluciones son ordenadas de acuerdo a su valor de *distancia de crowding*, dándole mayor prioridad de selección a aquellas soluciones con menor densidad de soluciones vecinas. La Figura 3.9 muestra el mecanismo de asignación de fitness mediante rangos para un problema de ejemplo donde se busca minimizar dos objetivos.

Figura 3.9: Mecanismo de asignacion de fitness en *NSGA-II*.

El esquema de funcionamiento de *NSGA-II* se describe en el Algoritmo 3. *NSGA-II* ha sido aplicado para la resolución de problemas de optimización en diversas áreas, obteniendo muy buenos resultados en la práctica.

Algoritmo 3 Esquema del algoritmo *NSGA-II*.

```

1: inicializar(P(0))
2: generación = 0
3: Evaluar (P(0))
4: mientras no CriterioParada hacer
5:    $R = Padres \cup Hijos$ 
6:   Frentes = Ordenamiento no Dominado(R)
7:   NuevaPop =  $\emptyset$ 
8:   i = 1
9:   mientras  $|NuevaPop| + |Frentes(i)| \leq sizepop$  hacer
10:    Calcular Distancia de Crowding (Frentes(i))
11:    NuevaPop = NuevaPop  $\cup$  Frentes(i)
12:    i++
13:  fin mientras
14:  Ordenamiento por Distancia(Frentes(i))
15:  NuevaPop = NuevaPop  $\cup$  Frentes(i)[1:(sizepop - |NuevaPop|)]
16:  Hijos = Selección y Reproducción(NuevaPop)
17:  generación++
18:  P(generación) = NuevaPop
19: fin mientras
20: retornar mejor solución

```

Capítulo 4

Trabajos relacionados

En este capítulo se describen los principales trabajos relacionados al problema abordado en este proyecto. El problema de viajes compartidos en taxis tiene varios puntos en común con el problema de ruteo de vehículos (*Vehicle Routing Problem*), el cual ha sido ampliamente estudiado en la literatura. Sin embargo, la principal diferencia radica en que el *VRP* trata sobre el ruteo de vehículos que transportan carga genérica, mientras que el problema estudiado en este proyecto se enfoca en resolver el ruteo de taxis que transportan pasajeros. El hecho de transportar pasajeros hace que las decisiones sobre costos de traslados, demoras percibidas y calidad de servicio afecten en un plano más personal que al planificar el transporte de carga. Por este motivo, se decidió centrar el estudio de trabajos relacionados en aquellos que se enfoquen en el transporte de pasajeros. La Sección 4.1 presenta trabajos que abordan el problema de viajes compartidos en autos (*Car Pooling Problem*). La Sección 4.2 presenta los principales trabajos que tratan el problema conocido como *Dial-a-Ride Problem (DARP)*. La Sección 4.3 describe los principales estudios que tratan con distintas variantes del problema de viajes compartidos en taxis (*Taxi Pooling Problem, TPP*). Finalmente, la Sección 4.4 presenta un breve resumen del análisis de la literatura relacionada al problema abordado en este proyecto.

4.1. Car Pooling Problem

El problema de viajes compartidos en taxis abordado en este proyecto tiene muchos puntos en común con el conocido problema de viajes compartidos en autos (*Car Pooling Problem*).

Hartman et al. [28] estudian el *CPP* como un problema teórico de grafos, distinguiendo dos posibles escenarios y presentando heurísticas ávidas para cada uno de ellos: i) el escenario donde se conoce de antemano el conjunto de conductores; ii) el escenario donde no se conoce quiénes van a actuar como conductores y quiénes como pasajeros. El primer escenario es equivalente al problema de asignación en grafos bipartitos, para el cual existen algoritmos polinomiales que encuentran la solución óptima [30]. Sin embargo, en instancias del problema de grandes dimensiones los tiempos de ejecución de los algoritmos exactos se tornan prohibitivos. Por esta razón, se propone una heurística ávida para la resolución del problema, que se evalúa utilizando un conjunto de 1000 grafos con 500 vértices cada uno, generados aleatoriamente. El algoritmo ávido alcanza en promedio una solución del 96 % respecto a la solución óptima. Posteriormente se presenta una variante del problema que consiste en recalcular una solución cuando el grafo

es perturbado (i.e., una cierta cantidad de aristas aparecen o desaparecen), partiendo de la solución calculada antes de la perturbación. Inicialmente se plantea la resolución de esta variante aplicando el algoritmo Kuhn-Munkres [36] y luego se propone su resolución mediante un algoritmo de proyección de gradientes que resuelve una formulación de programación lineal del problema. La resolución utilizando el algoritmo de proyección de gradientes es trivialmente paralelizable, logrando un mejor desempeño computacional (especialmente en instancias grandes del problema). El segundo escenario del *CPP* estudiado es de complejidad \mathcal{NP} -difícil, tal como fue probado en un trabajo previo de los mismos autores [34]. Para resolverlo se presentan dos heurísticas ávidas, las cuales son evaluadas entre sí utilizando datos generados a partir de simulaciones de movimiento en la región de Flanders, Bélgica. El análisis experimental reporta los resultados comparativos entre ambas heurísticas ávidas, tanto en calidad de soluciones como en desempeño computacional. Sin embargo, no se reporta la dimensión ni la cantidad de instancias de prueba utilizadas para la evaluación experimental y tampoco se comparan las soluciones halladas contra las de otros algoritmos de la literatura que resuelvan el problema.

Yan et al. [61] presentan una formulación para una variante del *CPP* que utiliza la información de soluciones pasadas (*CPP with pre-matching information, CPPPMI*). La información utilizada en el *CPPPMI* incluye las asignaciones entre pasajeros y vehículos, las rutas elegidas y los horarios de cada vehículo en asignaciones pasadas. La incorporación de información pasada permite mantener una cierta estabilidad en las asignaciones de pasajeros a vehículos a lo largo del tiempo, evitando exponer a los participantes a cambios frecuentes de compañeros de viaje y de rutas, que puedan desmotivarlos de realizar viajes compartidos. El problema es modelado como un caso particular de un problema de flujo de costo mínimo para múltiples bienes (*Multi-commodity Network Flow Problem*), el cual es \mathcal{NP} -difícil [24]. Cada pasajero proporciona su lugar de origen y destino, el horario más temprano en el que está dispuesto a partir de su origen y el horario más tardío al que está dispuesto a llegar a su destino, si dispone o no de vehículo propio (y su capacidad en caso afirmativo) y una serie de preferencias respecto a sus potenciales compañeros de viajes (hombres/mujeres, fumadores/no fumadores). El problema busca minimizar los costos para el grupo de pasajeros, cumpliendo las restricciones indicadas por cada uno de ellos y penalizando aquellas soluciones que dejen algún pedido sin servir. La solución desarrollada combina una relajación lagrangeana con una heurística para encontrar la cota superior al problema. La evaluación experimental fue realizada sobre un conjunto de 30 instancias de prueba (de entre 600 y 1400 pasajeros), generadas a partir de información del tráfico en la ciudad de Taiwan y fijando las preferencias de cada pasajero de forma aleatoria. Los resultados muestran que el algoritmo propuesto converge a menos de un 3 % de distancia de la solución óptima en todas las instancias de prueba. Estos resultados son alcanzados en tiempos de ejecución del entorno de una hora, lo que parece razonable para una planificación fuera de línea.

4.2. Dial-a-Ride Problem

Un problema muy relacionado con el *CPP* es el conocido como *Dial-a-Ride Problem*, donde la principal diferencia con el *CPP* radica en que los dueños de los vehículos no forman parte del grupo de pasajeros que desean compartir viajes.

Madsen et al. [42] estudian el *DARP* con ventanas de tiempo (*DARP with Time Windows, DARPTW*). El problema estudiado responde a las necesidades del departamento

de bomberos de la ciudad de Copenhague, Dinamarca, el cual ofrece un servicio de traslado para personas de la tercera edad y discapacitadas. El problema estudiado es estático, por lo que se conoce de antemano la información de cada pedido al momento de realizar la planificación. Cada pedido contiene la información del origen y destino del viaje, la ventana de tiempo deseada por el pasajero para ser recogido en su origen o para llegar a su destino, sus requerimientos específicos de transporte (e.g., asiento común, asiento para niño, silla de ruedas), el tiempo que le toma ascender y descender del vehículo, y un nivel de prioridad asociado al pasajero. Además, se conoce para cada vehículo de la flotilla su velocidad promedio, los tiempos en los cuales se encuentra operativo, las capacidades asociadas a cada requerimiento especial y su costo operacional. Los tiempos de recorrido entre cada punto son conocidos, así como los intervalos de descanso de los conductores y de servicio o reparación de los vehículos. El problema consiste en la optimización de una serie de objetivos: minimizar el tiempo total de manejo, minimizar la cantidad de vehículos utilizados, minimizar el tiempo de espera de los pasajeros, minimizar la desviación respecto al servicio prometido a los pasajeros y minimizar el costo operacional. La optimización multiobjetivo se reduce a un problema monoobjetivo especificando una serie de pesos para cada uno de los objetivos, de acuerdo a los criterios deseados por el planificador. Se diseña una heurística de inserción para resolver el problema, que comienza insertando aquellos pedidos que a priori presentan más dificultades para su inserción (e.g., ventanas de tiempo muy reducidas, requerimientos de traslado específicos que pocos vehículos pueden satisfacer). Luego, se itera sobre los pedidos no insertados en la planificación y se busca incorporarlos en el lugar que consiga el mejor valor para la función objetivo. La evaluación experimental se realiza sobre un escenario con 300 pedidos y 24 vehículos, utilizando información realista del tráfico de la ciudad de Copenhague y con patrones de pedidos típicos provistos por el departamento de bomberos de dicha ciudad. El algoritmo es capaz de encontrar una planificación en unos 10 segundos de ejecución, logrando mejorar el tiempo de ejecución entre 6 y 10 veces respecto a soluciones previamente desarrolladas en la literatura relacionada [33].

Cordeau et al. [15] resuelven el *DARP* estático para un conjunto de vehículos que parten desde un mismo depósito y deben atender pedidos de un grupo de clientes. En la variante estudiada se plantea encontrar las rutas que minimicen el costo operacional del total de los vehículos, conociendo las ventanas de tiempo deseadas por cada pasajero para partir de su origen y para llegar a su destino. Se considera además una cota para el tiempo máximo que un pasajero puede permanecer dentro de un vehículo, como una restricción dura al problema. El problema se modela a través de un grafo y se resuelve mediante una búsqueda tabú, la cual parte de una solución inicial y busca en cada iteración la mejor solución en el vecindario de la solución actual. El vecindario de una solución se define como el conjunto de soluciones a las que es posible llegar desde la solución actual realizando un único cambio en el grafo (e.g., removiendo una arista). Para evitar que la búsqueda se estanke en óptimos locales, se propone un mecanismo de relajación que permite explorar soluciones no factibles durante la búsqueda. La solución inicial a partir de la cual se inicia el proceso de búsqueda es generada de manera aleatoria. La evaluación experimental se realizó sobre un conjunto de 20 instancias de prueba (de entre 24 y 144 pedidos) generadas aleatoriamente utilizando información realista de la ciudad de Montreal, Canadá, para las ventanas de tiempo, capacidad de los vehículos, duración de las rutas y tiempos máximos de recorridos. Además, se utilizaron 6 instancias reales correspondientes a un empresa de servicios de transporte en Dinamarca. Los resultados

muestran que el algoritmo implementado es capaz de encontrar soluciones a menos de 1.5 % de la solución óptima, en un tiempo de ejecución que varía (dependiendo de la instancia) entre 2 y 90 minutos. Estos resultados sugieren la aplicabilidad del algoritmo a resolver problemas de planificación diarios en una empresa que ofrezca servicios de transporte de pasajeros.

Chevrier et al. [11] presentan una formulación multiobjetivo para el *DARP*, donde se busca minimizar la cantidad de vehículos utilizados, minimizar la duración de los viajes y minimizar las demoras percibidas por los pasajeros. Este problema se relaciona con la variante multiobjetivo del problema de viajes compartidos en taxis estudiado en este proyecto, donde se busca minimizar el costo total del grupo de pasajeros y la demora percibida por cada uno de ellos. El problema se resuelve utilizando tres *MOEA* de la literatura: *NSGA-II*, *SPEA-II*, e *IBEA*. Se utiliza una representación bidimensional para los individuos de la población que contiene la ruta asociada a cada vehículo de la solución. La población se inicializa de forma aleatoria y se utiliza un operador de cruzamiento ad hoc para el problema. El operador de mutación se define de la siguiente manera: el 90 % de las veces que es aplicado intercambia el orden de dos destinos en la solución; el 10 % restante de las veces aplica una búsqueda iterativa local (*Iterated Local Search*, *ILS*). La búsqueda *ILS* aplica un operador de la literatura relacionada al *VRP* conocido como *2-OPT-SWAP*, el cual selecciona una porción de la ruta de un vehículo y revierte su orden. No resulta clara la razón detrás de implementar *ILS* como parte de la mutación y no como un operador independiente, en particular cuando la configuración paramétrica muestra que los mejores resultados se alcanzan con probabilidades de mutación sumamente altas (0.5), por lo cual el algoritmo se desvía notoriamente del concepto de AE, transformándose en una búsqueda aleatoria guiada por el operador *ILS*. Inicialmente se realiza una evaluación experimental que busca comparar tanto el desempeño computacional como la calidad de las soluciones alcanzadas por cada uno de los tres *MOEA* implementados, utilizando el *indicador epsilon* y el *hipervolumen* como métricas para optimización multiobjetivo. Los resultados para un conjunto de 30 instancias de prueba de entre 100 y 1000 pasajeros generadas aleatoriamente, sugieren que el algoritmo *IBEA* es capaz de alcanzar mejores soluciones que sus dos competidores, en un menor tiempo de ejecución. Adicionalmente, se comparan los algoritmos con y sin hibridación (i.e., con y sin *ILS*), y se observa que la hibridación permite alcanzar mejores resultados sobre el conjunto de instancias de prueba estudiadas.

4.3. Taxi Pooling Problem

Diversos trabajos han estudiado el problema de viajes compartidos en taxis (*Taxi Pooling Problem*), tanto desde el punto de vista de la empresa de taxis que busca optimizar sus costos operacionales como desde el punto de vista de los pasajeros que buscan ahorrar al compartir sus viajes con otros pasajeros.

Hosni et al. [31] proponen un sistema centralizado para recibir pedidos de clientes y asignarlos a una flotilla de taxis. El sistema está compuesto por dos aplicaciones móviles (una para los pasajeros y otra para los taxistas) y un servidor central que ejecuta el algoritmo de planificación. La aplicación de los taxistas utiliza la información del *GPS* para enviar la información de la posición actual del taxi al servidor central. Al momento de solicitar un viaje desde la aplicación móvil, el usuario debe ingresar su destino, el cual es enviado junto a su posición actual (extraída del *GPS*) hacia el servidor central. El

servidor ejecuta un algoritmo de planificación que devuelve el taxi encargado de satisfacer el pedido del cliente y envía dicha información al conductor y al cliente, junto al tiempo aproximado de espera y la información que identifica a ambos. El algoritmo de planificación implementado simplifica en exceso el problema, ya que define una región predeterminada para cada taxi y solamente asigna a un taxi pedidos que se originen en su región. Adicionalmente, solo se considera la posibilidad de agregar un único pasajero a la ruta ya establecida para un determinado taxi, lo que reduce significativamente el espacio de búsqueda. Para la resolución del problema se utiliza un algoritmo de programación entera que toma en cuenta la demanda de taxis, las preferencias de los conductores y los pedidos de los clientes. La evaluación experimental se realiza a través de una única simulación utilizando el simulador de tráfico de código libre *SUMO (Simulation of Urban Mobility)*. Los resultados muestran que el modelo propuesto lleva a una distribución de ingresos justa entre los conductores y que el tiempo de ejecución del algoritmo permite integrarlo en una aplicación en línea. El artículo se centra en describir el diseño de la aplicación y los resultados experimentales no son estadísticamente significativos, ya que se realizan sobre una única instancia de prueba. En el marco de nuestro proyecto se presenta el diseño de una aplicación web y una aplicación móvil que resulta más completa y comprehensiva que la presentada por Hosni et al., y que resuelve un problema significativamente más complejo y realista, considerando menos restricciones y suposiciones.

Lin et al. [38] estudian la optimización de rutas para taxis compartidos, considerando tanto los intereses de los conductores como los de los pasajeros. El problema a resolver consiste en minimizar el costo operacional de los vehículos (de forma de aumentar las ganancias de los conductores) y maximizar la satisfacción de los pasajeros (medida a través del tiempo de viaje y del tiempo de espera). El problema se reduce a una formulación monoobjetivo, definiendo distintos pesos para cada objetivo que se busca optimizar. Asimismo, el problema que se resuelve es estático, por lo que se conocen de antemano todos los datos de los pedidos, incluyendo las coordenadas de origen y destino y las ventanas de tiempo de cada pasajero. Se considera que los taxis viajan a una velocidad constante y toman las rutas de menor distancia entre cada par de puntos. Se propone un algoritmo de recocido simulado (*Simulated Annealing*) para resolver el problema luego de realizar un preprocesamiento con el fin de reducir el espacio de búsqueda, eliminando soluciones no factibles debido a restricciones en las ventanas de tiempo de cada pedido. Al igual que en el trabajo de Chevrier et al. [11], se utilizan dos operadores para guiar la búsqueda del algoritmo: el intercambio de dos destinos dentro de una ruta y el operador *2-OPT-SWAP*. La evaluación experimental se realiza sobre un escenario con 9 puntos geográficos que actúan tanto como orígenes y destinos en un total de 29 pedidos. Se utilizan taxis con capacidad fija de cuatro pasajeros, que viajan a una velocidad constante de 30km/h. Para compartir su viaje, se considera para cada pasajero una tolerancia del 50 % del tiempo que toma un viaje directo desde su origen hasta su destino, un valor que resulta muy holgado y no se adapta a la realidad propia de cada uno de los pasajeros. Los resultados alcanzados muestran que 10 taxis son capaces de satisfacer todos los pedidos, en lugar de 29 si se utiliza un taxi diferente para atender cada pedido, reduciendo además un 19 % la distancia total recorrida. Sin embargo, la evaluación experimental resulta inapropiada, ya que considera una única instancia de prueba (de tamaño reducido) y no presenta resultados comparativos frente a otras técnicas que validen el enfoque elegido.

Ma et al. [41] estudian el *TPP* en su variante dinámica, donde se busca planificar los pedidos de los usuarios en tiempo real, a medida que van siendo generados. Dado un conjunto fijo de taxis y un flujo de pedidos de usuarios, el problema consiste en definir el taxi que atiende cada pedido, buscando minimizar la distancia adicional que debe ser recorrida por atender ese nuevo pedido. La estrategia propuesta es ávida, en el sentido que busca minimizar la distancia adicional recorrida a medida que surge cada nuevo pedido, lo cual no garantiza que se minimice la distancia total recorrida por el conjunto de taxis en el total de la planificación. Debido a la característica dinámica del problema, se desea encontrar soluciones en tiempos de ejecución reducidos, para lo cual se describen estrategias que buscan minimizar el cómputo necesario al momento de incluir un nuevo pedido en la planificación. De forma de evitar calcular las distancias entre el origen de un nuevo pedido y cada uno de los taxis del sistema, se divide el mapa en una grilla y se precisan las distancias entre cada una de las celdas. Este procedimiento para el cálculo de las distancias ofrece una mejora en la eficiencia computacional pero reduce la precisión de los resultados. A continuación, se genera un conjunto de taxis candidatos a satisfacer el nuevo pedido, en función de la distancia al origen y al destino y de la ventana de tiempo del pedido. Luego, se itera sobre el conjunto de taxis capaces de satisfacer el pedido, insertando el viaje en la planificación del taxi que minimice la distancia adicional recorrida por satisfacer el pedido, sin modificar el orden relativo de los pedidos ya incluidos en la planificación actual del taxi. La evaluación experimental se realizó utilizando datos de taxis de la ciudad de Beijing, China, compuestos por trazas de *GPS* de más de 33.000 taxis, recolectadas durante un período de 87 días. Esta base de datos se utilizó para generar instancias realistas para la evaluación experimental de los algoritmos que se proponen para resolver el problema de viajes compartidos en taxis abordado en nuestro proyecto, tal como se detalla en la Sección 6.1. Los resultados en un escenario con una relación de 6 pedidos por taxi, muestran que el algoritmo propuesto permite atender un 25 % más de pedidos gracias a la utilización de viajes compartidos, logrando minimizar un 13 % la distancia recorrida por el conjunto de taxis.

Tao et al. [52] presentan dos heurísticas ávidas para resolver el *TPP* en su variante de un origen común a muchos destinos (*one-to-many*) y en la variante inversa de muchos orígenes a un mismo destino (*many-to-one*). El algoritmo agrupa a los pasajeros de acuerdo a las ventanas de tiempo de cada uno y considerando las preferencias que cada uno de ellos indica sobre sus compañeros de viaje. Cada pasajero puede establecer la cantidad máxima de personas con las que está dispuesto a compartir viaje y si acepta compartir viaje con hombres o mujeres por igual. De esta forma, el algoritmo forma los grupos de pasajeros asignados a cada taxi, considerando una capacidad máxima fija de cuatro pasajeros por taxi. La optimización en los recorridos de los taxis, con el fin de minimizar la distancia recorrida, se realiza luego de que los pasajeros del taxi ya están definidos, intercambiando el orden relativo en el que el taxi visitará sus orígenes y destinos. Este enfoque resulta menos elaborado que los propuestos en otros artículos de la literatura, ya que la cantidad de rutas posibles en el espacio de soluciones resulta muy reducida. La evaluación experimental se realizó a través de una aplicación real del algoritmo en un sistema de viajes compartidos en taxi en la ciudad de Taipei, Taiwan, conformado por 10 taxis y 798 pasajeros. Los resultados reportados muestran un ahorro en la distancia recorrida gracias a compartir los viajes, pero se presentan las mejoras en términos absolutos (cantidad de kilómetros de recorrido ahorrados gracias a compartir viajes), por lo que no es posible establecer la eficacia numérica real del algoritmo diseñado.

4.4. Resumen

La Tabla 4.1 presenta un resumen de los trabajos relacionados, siguiendo el orden en que fueron mencionados anteriormente. Se indica el autor, el año y una breve descripción con la principal contribución del trabajo.

Tabla 4.1: Trabajos relacionados al problema de viajes compartidos en taxis.

<i>autores</i>	<i>año</i>	<i>comentario</i>
Hartman et al. [28]	2014	Modela el <i>CPP</i> como un problema de grafos y lo resuelve con heurísticas ávidas, según se conozca o no de antemano quiénes van a actuar como conductores y quiénes como pasajeros.
Yan et al. [61]	2011	Resuelve una variante del <i>CPP</i> que toma en cuenta información acerca de las asignaciones pasadas.
Madsen et al. [42]	1995	Resuelve el <i>DARP</i> estático con ventanas de tiempo y requerimientos especiales por parte de los pasajeros.
Cordeau et al. [15]	2003	Resuelve el <i>DARP</i> estático utilizando una búsqueda tabú para minimizar el costo operacional de la flotilla de vehículos.
Chevrier et al. [11]	2012	Presenta una formulación multiobjetivo del <i>DARP</i> la cual es resuelta utilizando distintos <i>MOEA</i> .
Hosni et al. [31]	2012	Diseña un sistema centralizado para asignar pedidos de clientes a taxis en línea.
Lin et al. [38]	2012	Resuelve el <i>TPP</i> con un algoritmo de recocido simulado, tomando en cuenta los intereses de los conductores y de los pasajeros.
Ma et al. [41]	2013	Resuelve una variante dinámica del <i>TPP</i> con una estrategia ávida. La evaluación experimental se realiza sobre datos realistas que es utilizado para generar instancias en este proyecto.
Tao et al. [52]	2007	Resuelve el <i>TPP</i> en sus variantes <i>one-to-many</i> y <i>many-to-one</i> utilizando una heurística ávida.

El relevamiento de trabajos relacionados muestra que existe un gran interés en estudiar distintas variantes de problemas de planificación para viajes compartidos. Sin embargo, la mayoría de las propuestas estudiadas consideran los intereses de las empresas dueñas de la flotilla de vehículos. Pocos trabajos resuelven problemas enfocados en optimizar los recursos del grupo de pasajeros en un sistema de viajes compartidos.

Algunos trabajos utilizan técnicas ávidas para resolver el problema de viajes compartidos en taxis, utilizando criterios intuitivos para agrupar los pasajeros al momento de realizar la planificación. En el Capítulo 6 se describen dos algoritmos ávidos basados en las ideas presentadas en los trabajos relacionados, los cuales son utilizados en la evaluación experimental de los AE propuestos en nuestro proyecto. Adicionalmente, muchos de los trabajos estudiados resuelven únicamente problemas de optimización de un único objetivo, o en su defecto, de un conjunto de objetivos ponderados. La optimización multiobjetivo resulta un camino relevante a explorar, dado que los diversos intereses de un grupo de usuarios difícilmente puedan ser modelados en un único objetivo. El análisis de la literatura relacionada permite concluir que existe lugar para contribuir en soluciones enfocadas en los intereses de los usuarios finales, especialmente aplicando metaheurísticas eficaces como los AE y siguiendo un enfoque de optimización multiobjetivo.

Capítulo 5

Implementación

El presente capítulo describe los algoritmos evolutivos implementados para resolver el problema de viajes compartidos en taxis en sus diferentes variantes. La Sección 5.1 presenta las herramientas y bibliotecas utilizadas para la implementación de los distintos algoritmos. La Sección 5.2 presenta los detalles de implementación de un algoritmo evolutivo secuencial (*seqEA*) y de un micro algoritmo evolutivo paralelo (*pμEA*), para resolver el problema de viajes compartidos en taxis en su formulación monoobjetivo. Por último, la Sección 5.3 presenta los detalles de implementación de un algoritmo paralelo multiobjetivo por descomposición de dominio (*pμMOEA/D*) y de un algoritmo secuencial multiobjetivo explícito (*NSGA-II*) para resolver la variante multiobjetivo del problema de viajes compartidos en taxis.

5.1. Bibliotecas de desarrollo

La siguiente sección presenta las bibliotecas utilizadas para la implementación de los distintos algoritmos evolutivos que resuelven el problema de viajes compartidos en taxis, tanto en su formulación monoobjetivo como en su variante multiobjetivo.

5.1.1. Mallba

Mallba es una biblioteca para optimización combinatoria desarrollada en el lenguaje C++, que incluye métodos exactos, técnicas heurísticas y metaheurísticas. El proyecto Mallba surge de un esfuerzo en conjunto entre las universidades de Málaga, La Laguna y Barcelona en España. La biblioteca es de código libre y se encuentra disponible en el sitio web del proyecto [1].

Los algoritmos en Mallba son implementados a partir de *esqueletos* provistos por la biblioteca. Un *esqueleto* implementa una plantilla de un algoritmo de optimización, el cuál debe ser completado para adaptarse a un problema en concreto. El diseño de un esqueleto se basa en la separación de dos conceptos: las *características del problema* (incluyendo la función a optimizar y características de las soluciones tentativas) que deben ser proporcionadas por el usuario y las *técnicas de resolución* que son provistas de modo abstracto por la biblioteca.

Los esqueletos se implementan como un conjunto de clases requeridas y clases provistas:

- Las *clases requeridas* contienen los aspectos específicos del problema que se quiere resolver y deben ser implementadas por el usuario de la biblioteca en los archivos con extensión *req.cc*. Cada esqueleto incluye las clases requeridas *Problem* (con la definición de la instancia del problema que se desea resolver), *Solution* (que determina las funciones de codificación y decodificación de soluciones), *Crossover* y *Mutation* (correspondientes a las definiciones de los operadores de cruzamiento y mutación, respectivamente).
- Las *clases provistas* implementan aspectos internos de los algoritmos, de modo independiente a los problemas a resolver. Se encuentran en los archivos con extensión *pro.cc*. Las clases provistas por todo esqueleto son *Solver* (que abstrae el método de resolución específico) y *SetUpParams* (que permite fijar los parámetros necesarios para la ejecución). Otras clases pueden ser requeridas dependiendo del método de resolución empleado.

La biblioteca facilita el pasaje de implementaciones secuenciales a paralelas, sin ser necesarios grandes cambios en la implementación por parte del usuario. Para proveer soporte a implementaciones paralelas, Mallba utiliza la biblioteca libre MPICH [44]. MPICH es una implementación portable y de código libre del estándar de pasaje de mensajes MPI (*Message Passing Interface*) [43]. De esta manera, Mallba soporta tanto ejecuciones paralelas (utilizando múltiples procesadores en un mismo equipo) como ejecuciones distribuidas (utilizando diferentes máquinas conectadas a través de una infraestructura de red).

En el marco de este proyecto se desarrolló una variante de la biblioteca Mallba denominada *Malva*. El motivo principal detrás del desarrollo de Malva fue el de contar con un repositorio centralizado para la biblioteca, que permita a los usuarios de la misma desarrollar mejoras, obtener versiones actualizadas y reportar errores. Adicionalmente, se actualizaron bibliotecas obsoletas utilizadas en el código original, se mejoraron los archivos de compilación y se corrigieron errores en la implementación del manejo de objetos al hacer cálculos distribuidos. Además, se actualizó la documentación de la biblioteca y se creó un sitio web para el proyecto, disponible públicamente en la dirección www.themalvaproject.github.io.

La biblioteca Malva se utilizó para la implementación de los algoritmos *seqEA* y *pμEA* (descritos en la Sección 5.2) que resuelven el problema de viajes compartidos en taxis en su formulación monoobjetivo, y para la implementación del algoritmo *pμMOEA/D* (descrito en la Sección 5.3.2) que resuelve el problema de viajes compartidos en taxis en su variante multiobjetivo aplicando una estrategia de descomposición de dominio.

5.1.2. ECJ

ECJ es una biblioteca de computación evolutiva desarrollada en la Universidad George Mason de Fairfax, Estados Unidos [20]. Se encuentra desarrollada en Java y está especialmente diseñada para soportar problemas grandes y complejos. A pesar de orientarse fuertemente hacia la programación genética, ECJ provee soporte para una gran variedad de algoritmos evolutivos e incluso algoritmos específicos para problemas multiobjetivo. Ofrece también la posibilidad de realizar cómputo paralelo y/o distribuido, soportando

el modelo maestro/esclavo y el modelo de islas. Además, ofrece herramientas para facilitar la impresión de estadísticas de las ejecuciones realizadas, facilidades para retomar ejecuciones pausadas (*checkpointing*) y un generador de números pseudoaleatorios, entre otras herramientas.

Gracias a su arquitectura y a la separación en clases, la biblioteca permite soportar diversos métodos alternativos para operadores, tales como la inicialización de la población, la estrategia de selección de individuos y la aplicación de operadores evolutivos, sin implicar un esfuerzo adicional por parte del usuario. ECJ cuenta con soporte para múltiples codificaciones de los individuos e incluye implementaciones para varios de los operadores evolutivos conocidos en la literatura relacionada. Además, provee una interfaz accesible para el programador, logrando que la implementación de nuevas características específicas para un problema sea sencilla.

La biblioteca ECJ fue utilizada para la implementación del algoritmo *NSGA-II* (descrito en la Sección 5.3.3) correspondiente a la variante multiobjetivo del problema de viajes compartidos en taxis. La elección de dicha biblioteca se basó en la posibilidad de contar con soporte para algoritmos multiobjetivos, ya que Mallba no lo ofrece de manera explícita. Además, la biblioteca se encuentra bien mantenida y documentada, contando con actualizaciones recientes al momento de escribir este informe.

5.2. Algoritmos evolutivos para el problema de viajes compartidos en taxis (versión monoobjetivo)

En la siguiente sección se describen los detalles de dos algoritmos evolutivos implementados para resolver el problema de viajes compartidos en taxis en su formulación monoobjetivo (descrita en la Sección 2.1). Se plantean primero los aspectos comunes a las dos implementaciones y luego los detalles particulares de cada algoritmo.

5.2.1. Aspectos comunes a ambos algoritmos

En esta sección se presentan detalles de implementación que son comunes a los dos algoritmos evolutivos implementados para resolver el problema de viajes compartidos en taxis en su formulación monoobjetivo.

Datos de entrada

Se definió una nomenclatura común a los dos algoritmos evolutivos implementados para los datos de entrada asociados a la instancia del problema que se desea resolver. De esta manera, todos los datos correspondientes a la instancia del problema quedan definidos en un único archivo de texto que contiene la siguiente información:

- cantidad de pasajeros que desean compartir su viaje,
- costo de la “bajada de bandera”,
- matriz de costos entre cada uno de los puntos de la instancia (origen y destinos),
- solución calculada por un algoritmo ávido (descrito en la Sección 6.3.3) que resuelve el problema, la cual es utilizada en una de las estrategias de inicialización de la población.

Codificación de las soluciones

Las soluciones del problema (individuos en el algoritmo evolutivo) son representadas como tuplas de largo $2N - 1$ donde N es el número de pasajeros de la instancia del problema a resolver. Las tuplas contienen los enteros del 1 al N que representan a cada uno de los pasajeros y $N - 1$ ceros que actúan como separadores, para distinguir grupos de pasajeros asignados a diferentes taxis. Cada grupo de dígitos distintos de cero representan a un taxi que transportará hacia su destino a cada uno de los pasajeros, respetando el orden en que aparecen en la tupla, de izquierda a derecha. La Figura 5.1 muestra la codificación de una posible solución para una instancia con cinco pasajeros ($N = 5$).



Figura 5.1: Ejemplo de representación de una solución.

La solución mostrada como ejemplo hace uso de tres taxis para transportar a los cinco pasajeros. Un primer taxi parte del origen (común a todos los pasajeros) y se dirige hacia el destino del pasajero #3 y luego al destino del pasajero #1. Otro taxi, con un único pasajero, parte desde el origen y se dirige hacia el destino del pasajero #5. Finalmente, un tercer taxi parte desde el origen y viaja hacia el destino del pasajero #2 y luego continúa su recorrido hasta el destino del pasajero #4.

Dada la representación elegida surgen las siguientes restricciones para cada tupla:

- el número de dígitos consecutivos mayores que cero está limitado por la máxima cantidad de pasajeros permitidos por taxi (C_{MAX}),
- cada número mayor que 0 debe aparecer una y sólo una vez, para asegurar que cada pasajero esté asignado a un único taxi,
- finalmente, se observa que dos o más ceros consecutivos cumplen el mismo rol que un único cero, y que los ceros al comienzo y al final de la tupla no cumplen ninguna función.

Generación de la población inicial

Para la generación de la población inicial se proponen dos métodos alternativos. El primer método consiste en una inicialización completamente aleatoria y el segundo propone una inicialización basada en una estrategia ávida (descrita en la Sección 6.3.3). Los resultados obtenidos utilizando ambas inicializaciones son comparados en el análisis experimental descrito en la Sección 6.3.4.

Inicialización aleatoria: esta estrategia de inicialización de la población coloca los pasajeros del 1 al N en posiciones aleatorias de una tupla con todos sus elementos inicializados con el valor 0. El proceso se describe en el Algoritmo 4.

Algoritmo 4 Inicialización aleatoria de la población.

```

desde  $i = 1$  a tamaño(población) hacer
    individuo = población[i];
    desde  $j = 1$  a  $2N - 1$  hacer
        individuo[j] = 0; {inicializar solución con ceros}
    fin desde
    desde pasajero = 1 a  $N$  hacer
        repetir
            posición = random(1,  $2N - 1$ );
            hasta (individuo[posición] == 0) {sortear una posición libre}
            individuo[posición] = pasajero;
        fin desde
    fin desde

```

Inicialización ávida: este método utiliza la solución obtenida por el algoritmo ávido que se describe en la Sección 6.3.3 para generar la población inicial. El Algoritmo 5 describe el funcionamiento de esta estrategia de inicialización. Un individuo de la población se inicializa copiando directamente la solución obtenida por el algoritmo ávido. El resto de la población se inicializa realizando un número aleatorio de perturbaciones (i.e., intercambio de dos valores) sobre la solución del algoritmo ávido. La cantidad máxima de perturbaciones a realizar se fijó en un cuarto del largo total del individuo.

Algoritmo 5 Inicialización ávida de la población.

```

desde  $m = 1$  a  $2N - 1$  hacer
    población[1][m] = solución_ávida[m]; {primer individuo (copia directa)}
fin desde
desde  $i = 2$  a tamaño(población) hacer
    individuo = población[i];
    desde  $j = 1$  a  $2N - 1$  hacer
        individuo[j] = solución_ávida[j];
    fin desde
    cantidad_perturbaciones = aleatorio(1,  $\frac{2N-1}{4}$ );
    desde  $k = 1$  a cantidad_perturbaciones hacer
        posición1 = random(1,  $2N - 1$ );
        posición2 = random(1,  $2N - 1$ );
        mientras posición2 == posición1 hacer
            posición2 = random(1,  $2N - 1$ );
        fin mientras
        individuo.intercambiar(posición1, posición2);
    fin desde
fin desde

```

Función correctiva

Luego de inicializar la población, alguno de los individuos generados puede violar las restricciones impuestas a la codificación de soluciones. Por este motivo, se diseñó un procedimiento de corrección de soluciones que se describe en el Algoritmo 6. La función implementada corrige las soluciones inválidas desplazando ceros consecutivos de forma de romper, en un punto aleatorio, las secuencias de dígitos mayores que cero que superan el largo máximo permitido (valor de C_{MAX}).

Algoritmo 6 Procedimiento de corrección de soluciones.

```

mientras no terminar hacer
    largo_secuencia = 0;
    posición = 0;
    mientras largo_secuencia<=  $C_{MAX}$  y posición <  $2N - 1$  hacer
        si individuo[posición] ≠ 0 entonces
            largo_secuencia++;
        en otro caso
            largo_secuencia = 0;
        fin si
        posición++;
    fin mientras
    si largo_secuencia >  $C_{MAX}$  entonces
        punto_de_corte = random(posición- $C_{MAX} + 1$ , posición-1);
        cero = encontrarPrimerCeroConsecutivo();
        individuo.intercambiar(cero, punto_de_corte);
    en otro caso
        terminar
    fin si
fin mientras

```

Recombinación

Para recombinar individuos se utilizó una versión modificada del operador de cruceamiento basado en la posición (*PBX*), que se describe en el Algoritmo 7.

Algoritmo 7 Cruzamiento basado en la posición (*PBX*).

- 1: seleccionar aleatoriamente varias posiciones del padre 1.
 - 2: generar parcialmente el hijo 1, copiando los valores (*alelos*) de las posiciones elegidas del padre 1 directamente al hijo 1.
 - 3: marcar los alelos del padre 2 que ya fueron seleccionados en el padre 1.
 - 4: desde el inicio del padre 2, seleccionar secuencialmente el siguiente alelo que no haya sido marcado, y copiarlo a la primer posición libre del hijo 1, desde el comienzo.
-

De forma de evitar que los hijos resultantes del cruzamiento PBX violen las restricciones de codificación de soluciones, se aplica la función correctiva definida en el Algoritmo 6 al completar el cruzamiento. La Figura 5.2 muestra un ejemplo del cruzamiento PBX de dos individuos y de la aplicación de la función correctiva, en una instancia del problema con cinco pasajeros y un máximo permitido de cuatro pasajeros por taxi ($C_{MAX} = 4$).

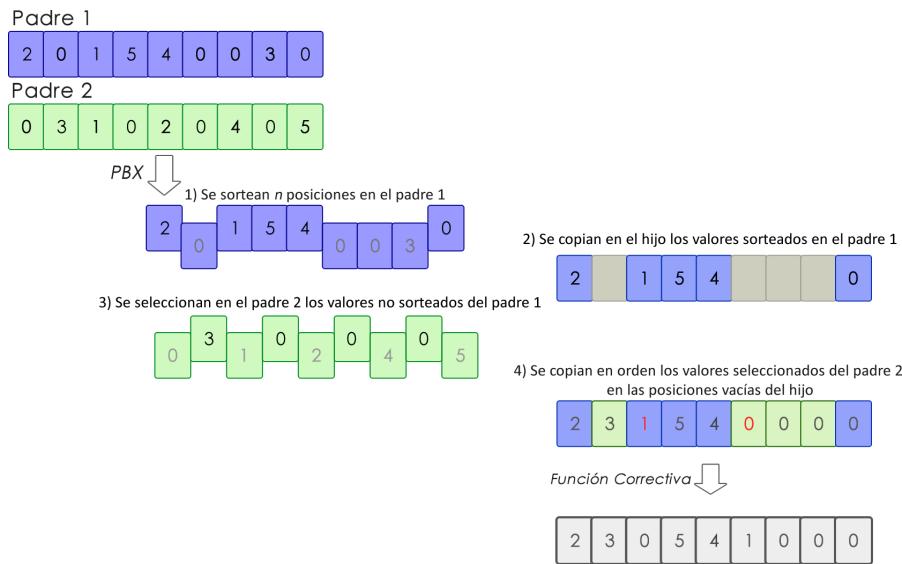


Figura 5.2: Ejemplo de aplicación del cruzamiento PBX y de la función correctiva.

Mutación

Se utilizó el operador de mutación por intercambio (*EM*), el cual selecciona dos posiciones en la solución a mutar de forma aleatoria y las intercambia de lugar. El procedimiento se describe en el Algoritmo 8.

Algoritmo 8 Mutación por intercambio (*EM*).

```

1: posición_1 = random(0, 2N - 1)
2: posición_2 = random(0, 2N - 1)
3: mientras posición_2 == posición_1 hacer
4:   posición_2 = random(0, 2N - 1)
5: fin mientras
6: individuo.intercambiar(posición_1, posición_2)

```

El individuo mutado puede no cumplir con las restricciones impuestas en la codificación de soluciones, por lo que se aplica nuevamente la función correctiva definida en el Algoritmo 6 luego de la mutación. La Figura 5.3 muestra un ejemplo de mutación de un individuo y la aplicación de la función correctiva, en una instancia del problema con cinco pasajeros y un máximo de cuatro pasajeros permitidos por taxi ($C_{MAX} = 4$).

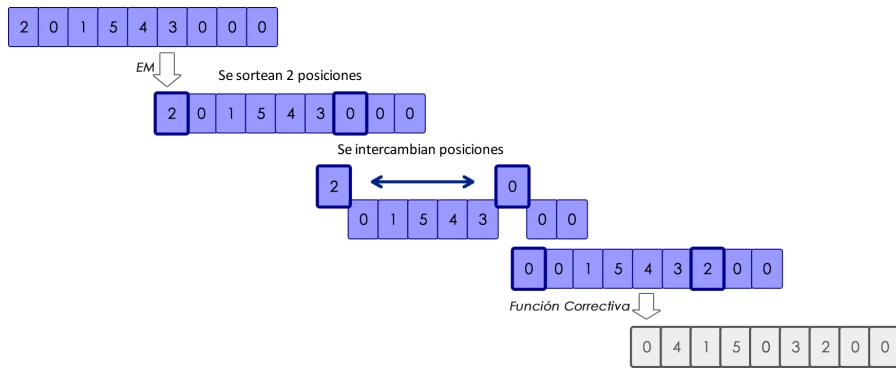


Figura 5.3: Ejemplo de la mutación EM y de la función correctiva.

Función de fitness

La función objetivo del problema a minimizar corresponde al costo total de los pasajeros (definido en la Ecuación 2.1). Para transformar el problema de minimización de costo en un problema de maximización de fitness, se utiliza como función de fitness el opuesto de la función de costo. Esta transformación se maneja automáticamente en la biblioteca Malva, retornando el valor “minimize” en el método *direction* de la clase *Problem*.

5.2.2. Algoritmo evolutivo secuencial: *seqEA*

El algoritmo *seqEA* es un AE secuencial implementado para resolver el PVCT en su formulación monoobjetivo. El algoritmo fue implementado en C++ utilizando la biblioteca Malva. En el Apéndice B se ofrece el artículo “Online taxi sharing optimization using evolutionary algorithms”, Fagúndez, G.; Massobrio, R.; Nesmachnow, S., XL Latin American Computing Conference, 2014, que reporta el diseño, implementación y evaluación experimental de *seqEA*.

Se implementaron dos variantes de *seqEA*: una utilizando la inicialización aleatoria y otra utilizando la inicialización ávida (descritas en la Sección 5.2.1). Se utilizó el método de *selección proporcional*, donde a cada individuo se le asigna una probabilidad de selección dada por el cociente entre su valor de fitness y la suma del fitness del total de los individuos en la población (Ecuación 5.1). Este método asigna una mayor probabilidad de selección a los mejores individuos de la población, pero mantiene la posibilidad de elegir individuos con peor valor de fitness, para preservar la diversidad de la población.

$$P_i = \frac{fitness(i)}{\sum_{j=1}^N fitness(j)} \quad (5.1)$$

5.2.3. Micro algoritmo evolutivo paralelo: *pμEA*

El algoritmo *pμEA* es un AE paralelo con micro-población, que utiliza el modelo de subpoblaciones distribuidas descrito en la Sección 3.2.1. *pμEA* fue implementado utilizando la biblioteca Malva, haciendo uso de las facilidades que ofrece para el soporte de paralelismo. En el Apéndice B se presenta el artículo “A parallel micro evolutionary algorithm for taxi sharing optimization”, Massobrio, R.; Fagúndez, G.; Nesmachnow, S., VIII ALIO/EURO Workshop on Applied Combinatorial Optimization, 2014, que reporta los detalles de diseño e implementación del algoritmo *pμEA* y su evaluación experimental.

Se implementaron dos variantes del algoritmo utilizando cada una de las estrategias de inicialización de la población descritas en la Sección 5.2.1. El método de selección utilizado fue el de *selección por torneo*, ya que experimentos iniciales mostraron que la selección proporcional no era capaz de proveer suficiente diversidad cuando se trabaja con micro-poblaciones. En la selección por torneo se sortean m individuos de la población y sobrevive aquel o aquellos con mayor(es) valor(es) de fitness.

Al utilizar un modelo de islas distribuidas es necesario introducir un nuevo operador evolutivo: la migración de soluciones entre subpoblaciones. Se utilizó una migración asíncrona, que considera a las subpoblaciones conectadas siguiendo una topología de anillo unidireccional. El operador de migración selecciona individuos adaptados de cada isla (mediante una selección por torneo) para ser enviados hacia la isla vecina, donde reemplazan a los peores individuos.

5.3. Algoritmos evolutivos para el problema de viajes compartidos en taxis (versión multiobjetivo)

En la siguiente sección se describen los detalles de los dos algoritmos evolutivos implementados para resolver el problema de viajes compartidos en taxis en su variante multiobjetivo (descrita en la Sección 2.2). Se describen los aspectos comunes a los dos algoritmos y luego los detalles particulares de cada uno de ellos.

5.3.1. Aspectos comunes a ambos algoritmos

En la siguiente sección se presentan los detalles de implementación que son comunes a los dos algoritmos evolutivos implementados para resolver el problema de viajes compartidos en taxis en su variante multiobjetivo.

Datos de entrada

Se definió una nomenclatura, común a los dos algoritmos evolutivos implementados, para los datos de entrada. De esta forma, todos los datos correspondientes a la instancia del problema que se desea resolver quedan definidos en un único archivo de texto que contiene la siguiente información:

- cantidad de pasajeros que desean compartir su viaje,
- vector con el “nivel de apuro” de cada pasajero,
- vector con la cantidad de taxis disponibles de cada capacidad,
- costo de la “bajada de bandera”,
- matriz de costos entre cada uno de los puntos de la instancia (origen y destinos),
- matriz de tiempos de recorrido entre cada uno de los puntos de la instancia (origen y destinos),
- solución calculada por el algoritmo ávido en costo (descrito en la Sección 6.4.4),
- solución calculada por el algoritmo ávido en demora (descrito en la Sección 6.4.4).

Codificación de las soluciones

Se utilizó la misma representación para soluciones tentativas del problema que la utilizada en los algoritmos que resuelven la formulación monoobjetivo del problema. Sin embargo, al incorporar vehículos de distintas capacidades en la variante multiobjetivo del problema, la primer restricción sobre las soluciones mencionada en la Sección 5.2.1 varía. En la variante multiobjetivo del problema es necesario verificar que las secuencias de dígitos distintos de cero sean de largo menor o igual a la máxima capacidad disponible hasta el momento, de acuerdo al vector de capacidades de la flotilla de vehículos dado como dato de entrada. De esta forma, es posible asegurar que existen taxis disponibles para trasladar a los pasajeros siguiendo la asignación representada por la solución.

Generación de la población inicial

Debido a que la estrategia de inicialización ávida alcanzó mejores resultados que la estrategia de inicialización aleatoria en la evaluación experimental de los algoritmos que resuelven el problema en su formulación monoobjetivo (los resultados numéricos se reportan en la Sección 6.3.4), se decidió utilizar dicha estrategia para la inicialización de la población en los algoritmos que resuelven el problema en su variante multiobjetivo.

Para la inicialización se utilizan las soluciones obtenidas por dos algoritmos ávidos, uno enfocado en el costo y otro en la demora, descritos en la Sección 6.4.4. Un individuo de la población se inicializa copiando directamente la solución alcanzada por el algoritmo ávido en costo y otro copiando la solución alcanzada por el algoritmo ávido en demora. El resto de la población se inicializa de la siguiente forma: una mitad como perturbaciones (i.e., intercambio de dos valores) sobre la solución del algoritmo ávido en costo y la otra mitad como perturbaciones sobre la solución del algoritmo ávido en demora.

Función correctiva

Al igual que en los algoritmos implementados para resolver la variante monoobjetivo del problema, es necesario aplicar un proceso de corrección de soluciones para garantizar que los individuos cumplen con todas las restricciones asociadas a la codificación de soluciones. Para la variante multiobjetivo del problema se implementó una función correctiva, similar a la implementada en los algoritmos que resuelven la variante monoobjetivo, pero que toma en cuenta la disponibilidad de vehículos de distinta capacidad.

El algoritmo de corrección busca secuencias de dígitos distintos de cero de largo mayor a la máxima capacidad disponible hasta el momento. Al encontrar una secuencia con dicha característica, se busca la primer pareja de ceros consecutivos y se mueve el primer cero hacia una posición aleatoria dentro de la secuencia encontrada, de forma de romper con el grupo inválido de dígitos. Este proceso continúa hasta que se recorre la solución por completo, garantizando que la misma cumple con todas las restricciones y representa una solución factible para el problema.

Selección

Se utilizó el operador de *selección por torneo* (definido en la Sección 5.2.3) para otorgar una apropiada presión selectiva. Experimentos iniciales mostraron que la selección proporcional no era capaz de ofrecer suficiente diversidad, conduciendo a convergencia prematura.

Recombinación y mutación

Para el cruzamiento y la mutación de individuos se utilizaron los mismos operadores empleados en los algoritmos que resuelven la variante monoobjetivo del problema: cruzamiento basado en la posición (PBX) y mutación por intercambio (EM) (ver Sección 5.2.1). Nuevamente, luego de aplicar cada operador evolutivo es necesario corregir los individuos resultantes utilizando la función correctiva, en su versión multiobjetivo.

5.3.2. Micro algoritmo evolutivo paralelo multiobjetivo por descomposición de dominio: $p\mu MOEA/D$

El algoritmo $p\mu MOEA/D$ (presentado en la Sección 3.3.2) es un AE paralelo con micro-poblaciones que utiliza el modelo de subpoblaciones distribuidas y sigue una estrategia de descomposición de dominio. $p\mu MOEA/D$ fue implementado utilizando la biblioteca Malva. En el Apéndice B se presenta el artículo “Planificación multiobjetivo de viajes compartidos en taxis utilizando un micro algoritmo evolutivo paralelo” Massobrio, R.; Fagúndez, G.; Nesmachnow, S., X Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, 2015, que reporta el diseño, implementación y evaluación experimental del algoritmo $p\mu MOEA/D$.

Función de fitness

Tal como fue explicado en la Sección 3.3.2, cada isla del algoritmo $p\mu MOEA/D$ se enfoca en resolver el problema de optimización utilizando una función de fitness específica. Se aplica un esquema de agregación lineal para los objetivos del problema utilizando diferentes pares de pesos para ponderar las funciones objetivo.

La función de fitness para cada isla queda definida por $F = w_C \times CT + w_D \times DT$, donde $w_C = [0 : \frac{1}{\#islas} : 1]$, $w_D = 1 - w_C$. CT y DT son los opuestos de las funciones objetivo definidas en la Sección 2.2.2, de forma de transformar el problema de minimización de las funciones objetivos en un problema de maximización de fitness.

Migración

El operador de migración es idéntico al utilizado en el algoritmo $p\mu EA$ (descrito en la Sección 5.2.3) pero en $p\mu MOEA/D$ cumple un propósito diferente. En el caso del algoritmo $p\mu EA$ la migración tiene como propósito intercambiar soluciones de buena calidad entre las distintas islas. Sin embargo, en el $p\mu MOEA/D$ cada isla resuelve un problema con una función de fitness diferente, por lo que una solución de buena calidad en una isla puede no ser buena en la isla a la que es enviada. La migración en el $p\mu MOEA/D$ cumple el rol de incorporar diversidad en las micro-poblaciones, fomentando la explotación e intentando evitar inconvenientes relacionados con la convergencia prematura al utilizar micro-poblaciones.

5.3.3. Algoritmo evolutivo multiobjetivo explícito: *NSGA-II*

El algoritmo *NSGA-II* (descrito en la Sección 3.3.2) es un AE que resuelve el problema con un enfoque multiobjetivo explícito. *NSGA-II* fue implementado utilizando la biblioteca ECJ. En el Apéndice B se ofrece el artículo “Multiobjective taxi sharing optimization using the NSGA-II evolutionary algorithm” Massobrio, R.; Nesmachnow, S.; Fagúndez, G., 11th Metaheuristics International Conference, 2015, que reporta los detalles de diseño e implementación y la evaluación experimental del algoritmo *NSGA-II*.

Asignación de fitness

El algoritmo *NSGA-II* es multiobjetivo de forma explícita, por lo que se utilizan los opuestos de las dos funciones objetivo definidas en la Sección 2.2.2 al momento de asignar el fitness. El propio algoritmo es el encargado de clasificar a las soluciones según dominancia, de acuerdo a lo explicado en la Sección 3.3.2.

Capítulo 6

Evaluación Experimental

El presente capítulo detalla la evaluación experimental de los AE implementados para resolver el problema de viajes compartidos en taxis. La Sección 6.1 describe el proceso de generación de instancias realistas del problema utilizadas para la evaluación experimental. La Sección 6.2 describe la metodología utilizada durante la evaluación experimental de los AE implementados. La Sección 6.3 presenta las pruebas realizadas y los resultados numéricos obtenidos con los algoritmos *seqEA* y *p μ EA* para el problema de viajes compartidos en taxis en su formulación monoobjetivo. La Sección 6.4 presenta los resultados experimentales correspondientes a los algoritmos *p μ MOEA/D* y *NSGA-II* que resuelven la variante multiobjetivo del problema de viajes compartidos en taxis.

6.1. Generación de instancias de prueba

Con el fin de realizar la evaluación experimental de los AE implementados, se generó un conjunto de instancias realistas del problema. La Sección 6.1.1 describe el proceso de generación de instancias. Las Secciones 6.1.2 y 6.1.3 detallan las instancias utilizadas en la evaluación experimental de los AE que resuelven el problema de viajes compartidos en taxis en su formulación monoobjetivo y en su variante multiobjetivo, respectivamente.

6.1.1. Proceso de generación de instancias

Esta sección describe el enfoque metodológico utilizado en el proceso de generación de instancias realistas, las fuentes de datos consultadas y las herramientas utilizadas.

Generación de puntos realistas en el mapa

Para generar instancias realistas del problema se utilizó una herramienta presentada en el trabajo de Ma et al. [41], llamada Generador de Pedidos de Taxis (*Taxi Query Generator*, *TQG*). *TQG* genera pedidos de taxis realistas a partir de información de una base de datos obtenida mediante dispositivos GPS instalados en taxis de la ciudad de Beijing, China. Los datos corresponden a la red vial de dicha ciudad, la cual está compuesta por 141.380 segmentos de ruta y 106.579 intersecciones. Se incluyen registros de más de 33.000 taxis, los cuales fueron recolectados durante un período de 87 días, entre los meses de marzo y mayo del año 2011. La distancia total registrada por el conjunto de taxis supera los 400 millones de kilómetros, lo que equivale a más de 790 millones de registros en la base de datos.

TQG se encarga de generar pedidos de taxis realistas utilizando la información histórica de la base de datos, mediante el proceso que se describe a continuación. Se discretiza el día en intervalos de tiempo f_j de duración α (por defecto $\alpha = 30$ minutos). Sea r_i la variable que denota el segmento de calle i . Se calcula la cantidad de viajes originados en un mismo segmento de calle durante el mismo intervalo de tiempo: c_i^j representa la cantidad de viajes originados en el segmento r_i en el intervalo de tiempo f_j . Con esta información, se generan pedidos de taxis originados en cada segmento r_i , que siguen una distribución de Poisson de parámetro $\lambda_i^j = c_i^j/\alpha$. Para generar los destinos de los viajes, cada c_i^j se descompone en un subconjunto $\{c_{i1}^j, c_{i2}^j, \dots, c_{im}^j\}$, donde c_{ik}^j representa la cantidad de pedidos originados en el segmento de calle r_i con destino al segmento r_k durante el intervalo de tiempo f_j . De esta forma, la probabilidad de transición del segmento r_i al segmento r_k durante el intervalo f_j queda definida por: $p_{ik}^j = c_{ik}^j/c_i^j$. La salida ofrecida por TQG consiste en un archivo de texto plano, donde cada línea corresponde a un viaje en taxi. Para cada viaje se indica la fecha y hora de comienzo, las coordenadas del origen y las coordenadas del destino.

Para obtener instancias aplicables al PVCT, fue necesario implementar un script que transforme la salida del TQG en instancias con un único origen y múltiples destinos, agrupando viajes que comienzan en orígenes cercanos. La distancia máxima permitida entre los puntos de origen de dos viajes al momento de agruparlos es configurable en el script. Esta distancia se estableció en 500 metros para el conjunto de instancias utilizadas en la evaluación experimental. La salida del script consiste en una lista de coordenadas con la ubicación del origen (común a todos los pasajeros) y de los diferentes destinos de cada pasajero. La Tabla 6.1 muestra una salida de ejemplo para 10 pasajeros.

punto	latitud	longitud	punto	latitud	longitud
Origen	39.8519013611111	116.4032090277778			
Destino #1	39.8667409722222	116.433686611111	Destino #6	39.9296867222222	116.4681216666667
Destino #2	39.8563128333333	116.369006333333	Destino #7	39.8293008611111	116.2884061666667
Destino #3	39.8546568888889	116.363095611111	Destino #8	39.8519013611111	116.4032090277778
Destino #4	39.9017958888889	116.420744777778	Destino #9	39.8817845277778	116.30666666666667
Destino #5	39.8583990277778	116.451695972222	Destino #10	39.8964006111111	116.4276463333333

Tabla 6.1: Ejemplo de coordenadas para 10 pasajeros.

Instancias en Montevideo

Ante comentarios de un revisor del artículo “*Online taxi sharing optimization using evolutionary algorithms*” presentado en la *XL Conferencia Latinoamericana en Informática, 2014* (ver Apéndice B), se generaron instancias del problema ubicadas geográficamente en la ciudad de Montevideo, Uruguay. No se tiene conocimiento de la existencia de registros disponibles de forma pública sobre los recorridos de los taxis en la ciudad de Montevideo. Por este motivo, se realizaron múltiples reuniones con la Intendencia de Montevideo con el fin de conseguir acceso a datos en poder de dicho organismo, pero las mismas no han dado resultados positivos hasta el momento de escribir este informe. Con el fin de contar con un conjunto de instancias ubicadas en la ciudad de Montevideo, se generaron instancias del problema de forma manual, seleccionando puntos en diferentes zonas de la ciudad. Los orígenes de las instancias generadas se establecieron en puntos de interés de la ciudad (facultades, centros comerciales, escuelas, etc.) y los destinos fueron seleccionados de forma de cubrir los principales barrios de la ciudad.

Generación de la matriz de costos

Luego de obtener las coordenadas del origen y de los destinos correspondientes a una instancia del problema, es necesario generar una matriz de costos de traslado entre dichos puntos. De forma de contar con costos realistas, se utilizó una herramienta denominada *Taxi Fare Finder (TFF)* [54].

TFF es un sitio web con información de tarifas de taxis realistas de más de 1000 localidades alrededor del mundo. El cálculo de las tarifas no se basa únicamente en la distancia y la duración del viaje, sino que también incorpora otras fuentes de información tales como patrones de tráfico, velocidades de manejo, densidad urbana y tiempos de espera. Las tarifas son actualizadas en base a la información de las compañías de taxis y de los propios usuarios del sitio. TFF provee una interfaz de programación de aplicaciones (*Application programming interface, API*) que permite a los desarrolladores acceder al servicio de cálculo de tarifas de forma programática. A través de una consulta a la *API*, es posible calcular el costo de traslado entre dos puntos geográficos. La consulta devuelve el costo inicial (“bajada de bandera”), el costo asociado a la distancia recorrida y, en caso de aplicarse, otros costos tales como propinas, cargos extra, peajes, etc.

La *API* fue utilizada en la generación de las instancias de prueba ubicadas en la ciudad de Beijing para obtener:

- el costo inicial (“bajada de bandera”),
- el costo asociado a la distancia recorrida entre cada par de puntos (el origen común y los destinos de cada uno de los pasajeros), para formar la matriz de costos de la instancia,
- el tiempo de recorrido entre cada par de puntos, para formar la matriz de tiempos de la instancia (únicamente en las instancias de prueba correspondientes a la variante multiobjetivo del problema)

Cada par de coordenadas de una determinada instancia del problema es enviado a la *API* para obtener el costo asociado, tal como indica la Figura 6.1.

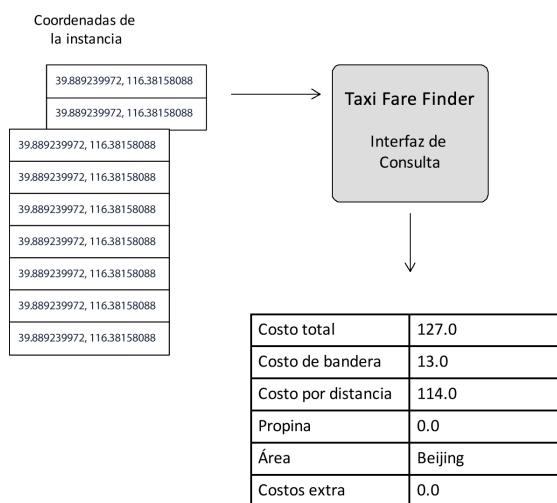


Figura 6.1: Consultas a través de la API de Taxi Fare Finder.

Desafortunadamente, TFF no cuenta con información de las tarifas de los taxis de la ciudad de Montevideo. Por esta razón, los costos para las instancias ubicadas en

Montevideo se calculan manualmente a partir de las tarifas publicadas por el Ministerio de Economía y Finanzas, vigentes al momento de escribir este informe (publicadas en la resolución 476/014 de agosto del 2014 [5], que detalla el costo inicial de “bajada de bandera” y el costo cada 100 metros de recorrido). Los costos asociados a cada tramo del viaje se obtienen en base a la distancia recorrida y al costo cada 100 metros. No se toman en cuenta demoras causadas por el tráfico, peajes, ni otros posibles gastos.

Visualización de instancias en el mapa

Luego de generar las instancias de prueba, es útil contar con una representación visual de las mismas. Para ello, se utilizaron las siguientes herramientas:

- *QGIS Desktop*, un sistema de información geográfica gratuito y de código libre, utilizado para crear, editar, visualizar y publicar material geoespacial [49]. QGIS se utilizó para transformar la lista de coordenadas asociadas a cada instancia en un archivo KML (*Keyhole Markup Language*). Este formato permite ser visualizado utilizando la herramienta Google Maps [27], de forma de tener una representación visual del origen y destinos de la instancia en un mapa.
- *Google Maps*, un servicio de mapas en línea de Google. Ofrece imágenes de mapas desplazables, así como imágenes satelitales. Permite visualizar en el mapa los archivos KML generados por *QGIS Desktop*, tal como se puede apreciar en la Figura 6.2.

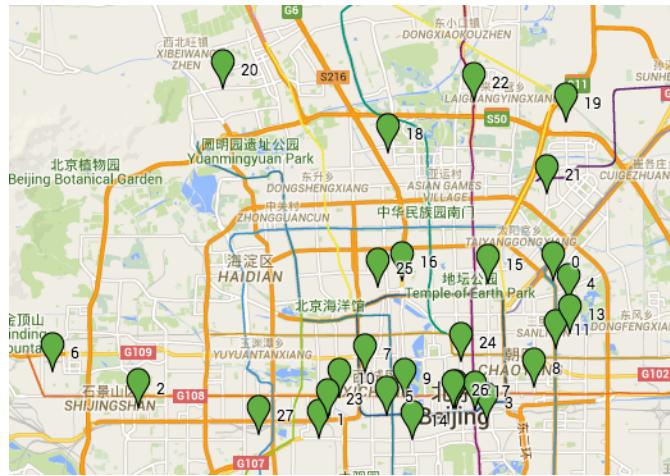


Figura 6.2: Visualización de una instancia de prueba en Google Maps.

6.1.2. Instancias de prueba para la variante monoobjetivo

Para la evaluación experimental de los algoritmos que resuelven el problema de viajes compartidos en taxis en su formulación monoobjetivo, se generaron un total de **22** instancias de prueba, agrupadas en cuatro familias:

- **6 instancias chicas:** entre 10 y 15 pasajeros, en la ciudad de Beijing.
- **6 instancias medianas:** entre 15 y 25 pasajeros, en la ciudad de Beijing.
- **6 instancias grandes:** entre 25 y 45 pasajeros, en la ciudad de Beijing.
- **4 instancias de Montevideo:** entre 8 y 17 pasajeros, en la ciudad de Montevideo.

Las instancias de prueba para el problema en su formulación monoobjetivo están identificadas siguiendo la nomenclatura $\langle\text{familia}\rangle\#X$. A modo de ejemplo, la instancia mediana #4 indica la instancia número 4 dentro de la familia de instancias medianas.

6.1.3. Instancias de prueba para la variante multiobjetivo

La variante multiobjetivo del problema de viajes compartidos en taxis, tal como se describe en la Sección 2.2.2, requiere dos datos de entrada adicionales con respecto a los que requiere el problema en su formulación monoobjetivo: la demora percibida por los pasajeros y la cantidad de vehículos disponibles de cada capacidad.

A partir de cada una de las instancias de prueba generadas para el problema en su formulación monoobjetivo, se generaron cuatro variantes con el mismo origen y los mismos destinos, pero estableciendo diferentes valores para la cantidad de taxis disponibles de cada capacidad y para el nivel de apuro de cada uno de los pasajeros. Todas las instancias generadas consideran una cantidad ilimitada de taxis con capacidad para 4 pasajeros, de forma de asegurar que hayan suficientes vehículos para trasladar a la totalidad de los pasajeros. Además, se consideran taxis con capacidades mayores, que permiten trasladar entre 5 y 10 pasajeros. La cantidad de taxis disponibles de cada capacidad fue establecida aleatoriamente, con un máximo de 10 taxis disponibles para una determinada capacidad. Para establecer el nivel de apuro de cada pasajero se consideraron cuatro valores posibles: 0%, 5%, 10% y 20% de tiempo extra tolerado por sobre el tiempo asociado a un viaje directo desde el origen hasta el destino del pasajero. El nivel de apuro de cada pasajero se seleccionó aleatoriamente entre estos valores.

De esta manera, se generaron un total de **88** instancias de prueba para la evaluación experimental de los algoritmos que resuelven la variante multiobjetivo del problema. Las mismas se corresponden a cuatro variantes de cada una de las instancias generadas para la formulación monoobjetivo del problema y están distribuidas de la siguiente forma:

- **24 instancias chicas:** entre 10 y 15 pasajeros, en la ciudad de Beijing.
- **24 instancias medianas:** entre 15 y 25 pasajeros, en la ciudad de Beijing.
- **24 instancias grandes:** entre 25 y 45 pasajeros, en la ciudad de Beijing.
- **16 instancias de Montevideo:** entre 8 y 17 pasajeros, ubicadas en Montevideo.

Las instancias de prueba para el problema en su variante multiobjetivo están identificadas siguiendo la nomenclatura $\langle\text{familia}\rangle\#XvY$, donde:

- $\text{familia} \in \{\text{chica, mediana, grande, Montevideo}\}$ indica el grupo al que pertenece la instancia de prueba.
- $X \in \begin{cases} \{1, \dots, 6\}, & \text{si } \text{familia} \in \{\text{chica, mediana, grande}\} \\ \{1, \dots, 4\}, & \text{si } \text{familia} = \text{Montevideo} \end{cases}$ indica el número de instancia dentro de la familia. Se corresponde con el número de la instancia de prueba generada para el problema en su formulación monoobjetivo, a partir de la cual se generó la instancia para el problema multiobjetivo.
- $Y \in \{1, \dots, 4\}$ indica la variante de la instancia del problema multiobjetivo.

A modo de ejemplo, la instancia chica #2v3 corresponde a la segunda instancia de la familia de instancias chicas, en su variante número 3. Dos instancias con igual *familia* e igual cardinal *X*, pero distinto número de variante *Y*, tienen el mismo conjunto de puntos (origen y destinos) pero presentan distintos valores para la cantidad de taxis disponibles de cada capacidad y para el nivel de apuro de los pasajeros.

6.2. Metodología

En esta sección se presentan los lineamientos generales seguidos al momento de realizar la evaluación experimental de los algoritmos implementados.

Entorno de ejecución. La evaluación experimental de los AE implementados fue realizada utilizando la infraestructura de cómputo de alto desempeño *Cluster FING* [12], de la Facultad de Ingeniería, Universidad de la República. El principal objetivo de *Cluster FING* es el de proveer soporte para la resolución de problemas complejos que demanden un gran poder de cómputo [45]. Las ejecuciones de todas las instancias de prueba para cada algoritmo fueron realizadas sobre la misma infraestructura, la cual se utilizó de modo exclusivo (i.e., no fue compartida con otros usuarios durante el período en que se realizaron las ejecuciones). De esta forma, se busca que los valores reportados de desempeño computacional no presenten interferencias debido a procesos ajenos.

Ejecuciones independientes. Debido a la cualidad estocástica de los algoritmos evolutivos, distintas ejecuciones sobre una misma instancia de prueba pueden alcanzar diferentes resultados. Con el fin de obtener resultados estadísticamente significativos, se realizaron 30 ejecuciones independientes de cada algoritmo sobre cada instancia de prueba. Las ejecuciones independientes se logran inicializando el generador de números pseudoaleatorios del AE con una semilla diferente en cada ejecución. Como consecuencia, al reportar los resultados obtenidos por un algoritmo en una determinada instancia de prueba, se indica el mejor valor, el valor promedio y la desviación estándar de los valores alcanzados en las 30 ejecuciones independientes realizadas.

Comparación de resultados. La comparación de resultados entre los AE implementados abarca tanto la calidad de las soluciones como la eficiencia computacional. La calidad de las soluciones se define de acuerdo a los valores alcanzados en las funciones objetivo y, para la variante multiobjetivo del problema, a través de métricas específicas para MOP. Para evaluar la eficiencia computacional, se reportan los tiempos de ejecución necesarios para mejorar las soluciones alcanzadas por algoritmos ávidos implementados para resolver el problema y los tiempos de ejecución totales.

Para comparar los resultados obtenidos por dos algoritmos (o por dos variantes de un mismo algoritmo) se utilizan tests estadísticos sobre las distribuciones de resultados obtenidos en las ejecuciones independientes de cada instancia. Primeramente, se aplica el test de Shapiro–Wilk [50] sobre cada muestra a comparar, para establecer si dicha muestra sigue o no una distribución normal. El test de Shapiro–Wilk plantea como hipótesis nula que la muestra estudiada proviene de una distribución normal. Si el *p*–valor obtenido es menor al nivel de confianza, se puede rechazar la hipótesis nula y concluir que la muestra estudiada no sigue una distribución normal. A lo largo del trabajo se utiliza un nivel de confianza del 95 % ($\alpha = 0,05$) para el test de Shapiro–Wilk.

En caso que el test de Shapiro–Wilk permita concluir que los resultados obtenidos no siguen una distribución normal, se debe utilizar un test no paramétrico para comparar los resultados de los dos algoritmos (o variantes de algoritmos). Con este propósito, se utiliza el test no paramétrico de Kruskal–Wallis [35], ya que no asume normalidad en las muestras a comparar. El test de Kruskal–Wallis plantea como hipótesis nula que las muestras que se comparan provienen de la misma distribución. Un p –valor menor al nivel de confianza permite rechazar la hipótesis nula y concluir que una muestra domina a la otra. En este trabajo se utiliza un nivel de confianza del 95 % ($\alpha = 0,05$) para evaluar los resultados del test de Kruskal–Wallis. Dado que el test no ofrece información acerca de cuál es la muestra que domina, se utiliza el valor promedio alcanzado en las 30 ejecuciones independientes para tomar la decisión.

6.3. Variante monoobjetivo del PVCT

Esta sección detalla el análisis experimental de los algoritmos *seqEA* y *pμEA*, que resuelven el problema de viajes compartidos en taxis en su formulación monoobjetivo.

6.3.1. Entorno de ejecución

La evaluación experimental del algoritmo *seqEA* fue realizada en un servidor Dell Power Edge 2950, utilizando un único núcleo de un procesador Intel Xeon E5430 2.66GHz con 8GB de memoria RAM disponibles. Las ejecuciones del algoritmo *pμEA* fueron realizadas utilizando un total de 24 núcleos de procesador de un servidor HP Proliant DL585 conformado por procesadores AMD Opteron 6272 2.09GHz con 48GB de memoria RAM.

6.3.2. Configuración paramétrica

Debido a la cualidad estocástica de los AE, es necesario ajustar sus parámetros previamente al análisis experimental [46]. Los resultados detallados obtenidos en la etapa de configuración paramétrica se pueden consultar en los artículos correspondientes a cada algoritmo en el Apéndice B. A continuación, se brinda un breve resumen de las configuraciones evaluadas y se mencionan aquellas que alcanzaron los mejores resultados.

seqEA

La configuración paramétrica del algoritmo *seqEA* fue realizada sobre tres instancias del problema de diferentes dimensiones, distintas a las utilizadas en la evaluación experimental del AE para evitar resultados sesgados. El número máximo de pasajeros permitidos en un mismo taxi (valor de C_{MAX}) se fijó en 4 para las tres instancias. El análisis se concentró en los siguientes parámetros: el tamaño de la población ($\#P$), la probabilidad de recombinación (p_C) y la probabilidad de mutación (p_M). Para cada parámetro se estudiaron los siguientes valores candidatos: $\#P \in \{150, 200, 250\}$; $p_C \in \{0,6, 0,75, 0,95\}$; y $p_M \in \{0,001, 0,01, 0,1\}$. Se realizaron 20 ejecuciones independientes (de 2000 generaciones cada una) para cada instancia, utilizando cada una de las posibles combinaciones de valores para los parámetros. Los mejores resultados fueron obtenidos con el máximo tamaño de población (250), el mínimo valor para p_C (0.6) y el máximo valor para p_M (0.1), para las tres instancias del problema estudiadas. Estos parámetros fueron los utilizados por *seqEA* para alcanzar los resultados numéricos que se reportan en la Sección 6.3.4.

p μ EA

La configuración paramétrica del algoritmo *p μ EA* fue realizada sobre un conjunto de cinco instancias del problema de dimensión media, distintas a las utilizadas en la evaluación experimental con el fin de evitar resultados sesgados. Mediante experimentos iniciales, el tamaño de la micro-población de cada isla fue establecido en 15 individuos. El operador de migración fue configurado para seleccionar dos individuos de cada isla utilizando selección por torneo, sorteando dos individuos y eligiendo el mejor (i.e., $m = 2$ y $k = 1$ según lo explicado en la Sección 3.1.4). Los individuos seleccionados son enviados hacia la isla vecina, donde reemplazan a los peores individuos. La frecuencia del operador de migración se estableció en 500 generaciones.

La configuración paramétrica se enfocó en los siguientes aspectos del AE: la probabilidad de cruzamiento (p_C) y la probabilidad de mutación (p_M). Se estudiaron los siguientes valores candidatos: $p_C \in \{0,6, 0,75, 0,95\}$ y $p_M \in \{0,001, 0,01, 0,1\}$. Para cada una de las nueve combinaciones de parámetros, se realizaron 20 ejecuciones independientes de 100.000 generaciones cada una. Los resultados del análisis paramétrico indican que los mejores resultados para las instancias estudiadas se obtienen utilizando $p_C=0.75$ y $p_M=0.1$. Estos parámetros fueron los utilizados para las ejecuciones de *p μ EA* para obtener los resultados numéricos que se detallan en la Sección 6.3.4.

6.3.3. Algoritmo ávido

Un algoritmo ávido (o *greedy*) es un método de construcción de soluciones que toma una decisión localmente óptima en cada paso, con la idea de llegar a un óptimo global del problema. Es relevante comparar el desempeño de *seqEA* y de *p μ EA* frente a una estrategia ávida, tanto en términos de calidad de las soluciones alcanzadas como de la eficiencia computacional. Con este propósito, se implementó un algoritmo ávido que busca minimizar el costo total del grupo de pasajeros tomando decisiones localmente óptimas. El algoritmo ávido utiliza las ideas presentadas en los trabajos relacionados de Ma et al. [41] y Tao et al. [52] para agrupar los pasajeros de acuerdo a sus destinos y asignarlos a diferentes taxis. Este algoritmo emula una estrategia intuitiva para resolver el problema, siguiendo un enfoque de agregación de decisiones localmente óptimas, aproximándose a lo que un grupo de usuarios humanos puede idear de manera simple para resolver el problema. El funcionamiento del algoritmo ávido se detalla en el Algoritmo 9.

El algoritmo ávido implementado comienza con un taxi inicialmente vacío y agrega en la primera posición de dicho taxi (primer destino a ser visitado) al pasajero cuyo destino se encuentra más cerca del origen (común a todos los pasajeros). Luego, continúa agregando pasajeros al taxi, eligiendo en cada paso al pasajero cuyo destino es el más cercano al destino del pasajero agregado en el paso anterior. El algoritmo continúa agregando pasajeros hasta alcanzar la capacidad del taxi actual (en cuyo caso forma un nuevo taxi) o hasta que todos los pasajeros hayan sido asignados en algún taxi. Una excepción ocurre cuando el costo de añadir un pasajero al taxi actual es mayor al costo de asignar un nuevo taxi que transporte únicamente a ese pasajero directamente desde el origen hasta su destino. En ese caso, se “cierra” el taxi actual, se forma un nuevo taxi para transportar a dicho pasajero y se continúa agregando pasajeros aún no asignados a este nuevo taxi.

Algoritmo 9 Algoritmo ávido para minimizar costo en el PVCT monoobjetivo.

```

taxi_actual = 1;
contador_pasajeros = 0;
mientras pasajerosSinAsignar() hacer
    si (contador_pasajeros == 0) entonces
        pasajero1 = destinoMasCercanoDesdeOrigen();
        agregarPasajero(pasajero1, taxi_actual, solución);
        contador_pasajeros++;
    en otro caso si (contador_pasajeros ≥ CMAX) entonces
        contador_pasajeros = 0; {El taxi actual está completo}
        taxi_actual++;
    en otro caso
        pasajero2 = vecinoMasCercano(pasajero1);
        si (costo(pasajero1,pasajero2) ≤ costo(origen, pasajero2) + B) entonces
            agregarPasajero(pasajero2, taxi_actual, solución);
            pasajero1 = pasajero2;
            contador_pasajeros++;
    en otro caso
        taxi_actual++; {Comenzar con un nuevo taxi}
        agregarPasajero(pasajero2, taxi_actual, solución);
        pasajero1 = pasajero2;
        contador_pasajeros = 1;
    fin si
    fin si
fin mientras
retornar solución;
```

6.3.4. Resultados numéricos

A continuación se presentan los resultados numéricos del análisis experimental de los algoritmos *seqEA* y *pμEA*, para la formulación monoobjetivo del problema. Las instancias utilizadas fueron generadas siguiendo la metodología descrita en la Sección 6.1. Para cada instancia se realizaron 30 ejecuciones independientes de 10.000 generaciones cada una.

Comparativa de métodos de inicialización

En la Sección 5.2.1 se presentaron dos métodos alternativos para la inicialización de los algoritmos *seqEA* y *pμEA*: inicialización aleatoria e inicialización basada en el algoritmo ávido descrito en la Sección 6.3.3. Las Tablas 6.2 y 6.3 muestran los resultados alcanzados por *seqEA* y por *pμEA* respectivamente, utilizando cada una de las estrategias de inicialización. Se reporta el mejor valor de costo (*min(c)*) y el promedio de los mejores valores de costo junto a su desviación estándar ($\bar{c} \pm std$), alcanzados en las 30 ejecuciones independientes. Se reporta además el *p*-valor resultante del test de Shapiro–Wilk (*pvSW*) sobre los valores de costo obtenidos utilizando cada método de inicialización, de forma de contrastar normalidad en los resultados obtenidos [50].

Ciertos resultados del test de Shapiro–Wilk indican que, para un subconjunto de las instancias de prueba estudiadas, es posible rechazar la hipótesis nula de que los resultados obtenidos siguen una distribución normal. Por esta razón, se utilizó el test no paramétrico de Kruskal–Wallis para comparar los resultados obtenidos por ambas inicializaciones.

inst	inicialización ávida			inicialización aleatoria			$pvK-W$	
	$min(c)$	$\bar{c} \pm std$	$pvS-W$	$min(c)$	$\bar{c} \pm std$	$pvS-W$		
ch	1	164.4	165.6 ± 2.0	3.5×10^{-8}	164.4	164.9 ± 1.3	1.8×10^{-10}	0.1
	2	220.7	225.7 ± 5.0	3.2×10^{-7}	220.7	224.9 ± 6.0	4.7×10^{-6}	0.4
	3	160.4	160.4 ± 0.0	1.0	160.4	160.4 ± 0	1.0	1.0
	4	181.3	181.3 ± 0.1	4.4×10^{-11}	181.3	182.0 ± 2.2	1.9×10^{-10}	0.3
	5	152.1	155.6 ± 4.5	6.0×10^{-6}	152.1	154.5 ± 4.1	1.0×10^{-7}	0.2
	6	118.4	119.6 ± 2.5	9.4×10^{-9}	118.4	118.8 ± 1.0	9.0×10^{-10}	0.4
me	1	211.9	216.0 ± 4.2	5.0×10^{-5}	211.9	213.5 ± 2.7	3.1×10^{-8}	0.0
	2	428.6	444.1 ± 11.7	0.0	427.3	444.5 ± 14.3	0.0	0.8
	3	361.7	378.7 ± 6.5	0.6	364.8	387.5 ± 14.1	0.1	0.0
	4	267.5	279.8 ± 5.5	2.2×10^{-5}	266.7	277.8 ± 10.9	0.0	0.3
	5	479.3	487.1 ± 6.5	0.0	481.7	518.4 ± 23.8	0.5	2.1×10^{-7}
	6	306.0	321.2 ± 7.7	0.5	306.0	329.7 ± 15.2	0.0	0.0
gr	1	421.9	435.1 ± 5.0	0.1	433.6	528.0 ± 65.3	0.0	1.0×10^{-9}
	2	479.3	489.9 ± 4.3	0.6	501.2	536.0 ± 22.3	0.4	2.9×10^{-11}
	3	332.8	349.7 ± 7.7	0.1	330.4	359.4 ± 14.2	0.7	0.0
	4	351.1	390.7 ± 26.3	0.1	370.7	397.2 ± 17.7	0.2	0.1
	5	395.9	429.6 ± 16.2	0.7	474.4	672.6 ± 55.3	0.0	2.9×10^{-11}
	6	360.8	382.4 ± 8.1	0.1	388.1	460.9 ± 69.6	0.0	4.4×10^{-10}
Mo	1	168.4	168.4 ± 0.0	1.0	168.4	168.6 ± 0.6	7.8×10^{-12}	0.3
	2	319.3	331.2 ± 3.8	4.2×10^{-5}	319.3	325.5 ± 5.8	0.0	7.3×10^{-5}
	3	266.7	269.1 ± 2.3	0.0	268.2	273.1 ± 3.3	0.3	1.4×10^{-5}
	4	303.2	304.7 ± 0.5	6.7×10^{-7}	303.2	311.7 ± 7.9	0.0	0.0

Tabla 6.2: Tabla comparativa: $seqEA$ con inicialización aleatoria vs. $seqEA$ con inicialización ávida

inst	inicialización ávida			inicialización aleatoria			$pvK-W$	
	$min(c)$	$\bar{c} \pm std$	$pvS-W$	$min(c)$	$\bar{c} \pm std$	$pvS-W$		
ch	1	164.4	164.4 ± 0.0	1.0	164.4	166.4 ± 2.7	8.2×10^{-7}	1.0
	2	220.7	220.7 ± 0.0	1.0	220.7	232.7 ± 11.8	0.0	1.0
	3	160.4	160.4 ± 0.0	1.0	160.4	162.0 ± 2.9	5.4×10^{-8}	1.0
	4	181.3	182.4 ± 1.9	1.0×10^{-8}	181.3	186.1 ± 7.0	7.9×10^{-7}	5.3×10^{-3}
	5	152.1	152.1 ± 0.0	1.0	152.1	162.9 ± 6.6	0.0	1.0
	6	118.4	118.4 ± 0.0	1.0	118.4	120.9 ± 3.2	1.2×10^{-5}	1.0
me	1	211.9	211.9 ± 0.0	1.0	211.9	218.4 ± 7.3	0.0	1.0
	2	427.9	429.4 ± 1.6	3.0×10^{-5}	434.3	447.9 ± 14.7	3.1×10^{-5}	1.4×10^{-3}
	3	364.5	370.4 ± 4.5	1.5×10^{-6}	375.7	390.9 ± 10.0	0.0	9.7×10^{-9}
	4	266.8	266.8 ± 0.0	1.8×10^{-7}	266.8	287.2 ± 13.6	0.1	0.1
	5	479.6	479.8 ± 0.2	1.0×10^{-7}	476.7	514.0 ± 21.6	0.2	5.5×10^{-6}
	6	306.0	307.7 ± 3.4	9.5×10^{-9}	306.0	340.6 ± 15.5	0.1	0.9
gr	1	425.9	437.7 ± 3.2	6.6×10^{-11}	418.0	451.1 ± 19.5	0.2	1.5×10^{-11}
	2	477.0	481.1 ± 2.3	0.0	484.4	521.8 ± 21.9	0.2	1.8×10^{-11}
	3	326.3	331.7 ± 4.0	0.0	323.8	353.2 ± 19.4	0.0	1.8×10^{-4}
	4	338.4	344.8 ± 6.1	9.4×10^{-5}	349.3	379.4 ± 14.0	0.2	2.4×10^{-4}
	5	370.2	380.0 ± 4.4	0.0	393.6	418.7 ± 14.9	0.5	2.6×10^{-11}
	6	343.8	350.6 ± 3.8	0.0	353.5	375.9 ± 13.7	0.2	0.2
Mo	1	168.4	168.4 ± 0.0	1.0	168.4	169.8 ± 2.0	9.7×10^{-8}	1.0
	2	324.9	328.6 ± 3.2	1.2×10^{-6}	320.4	335.6 ± 11.7	0.0	7.0×10^{-12}
	3	266.7	266.7 ± 0.0	1.0	266.7	280.6 ± 12.5	0.0	0.2
	4	304.1	304.5 ± 0.4	1.4×10^{-7}	303.2	318.4 ± 12.9	0.0	3.9×10^{-13}

Tabla 6.3: Tabla comparativa: $p\mu EA$ con inicialización aleatoria vs. $p\mu EA$ con inicialización ávida.

En la última columna de las Tablas 6.2 y 6.3 se reporta el p -valor del test de Kruskal–Wallis ($pvK-W$) sobre los valores de costo obtenidos con cada estrategia de inicialización. En aquellas instancias en las que el test de Kruskal–Wallis permite afirmar (con un nivel de confianza del 95 %) que una de las inicializaciones obtuvo mejores resultados que la otra (i.e., $pvK-W \leq 0,05$), se destaca en negrita la que alcanzó el menor costo promedio.

Los resultados del test de Kruskal–Wallis sugieren que la inicialización ávida permite alcanzar mejores resultados que la inicialización aleatoria en el conjunto de instancias de prueba estudiadas. En el caso del algoritmo *seqEA*, la inicialización ávida alcanzó mejores resultados que la inicialización aleatoria en **10** instancias de prueba, mientras que la inicialización aleatoria alcanzó mejores resultados en tan solo **2** instancias. Para el algoritmo *pμEA* los resultados son similares: la inicialización ávida alcanzó mejores resultados que la inicialización aleatoria en **11** instancias de prueba, mientras que no hubo instancias en las que se pueda afirmar que la inicialización aleatoria haya alcanzado mejores soluciones. En base a los resultados numéricos obtenidos, se decidió utilizar la inicialización ávida para el resto de la evaluación experimental de ambos algoritmos.

seqEA

Una vez definido el método de inicialización a utilizar, se reportan los resultados del algoritmo *seqEA* sobre el conjunto de instancias de prueba. La Figura 6.3 muestra el tiempo requerido por *seqEA* para lograr mejorar las soluciones encontradas por el algoritmo ávido en cada instancia de prueba. Se muestran los tiempos necesarios para alcanzar mejoras con un paso de 5% y, sobre cada barra, se indica la mejora final alcanzada por *seqEA* al completar las 10.000 generaciones. Los valores reportados corresponden a aquellas ejecuciones que alcanzaron la máxima mejora final.

El algoritmo *seqEA* fue capaz de alcanzar mejores valores de costo que los alcanzados por el algoritmo ávido en **todas** las instancias de prueba estudiadas. En el mejor caso (instancia grande #4), *seqEA* fue capaz de mejorar el costo alcanzado por el algoritmo ávido en un **35.9 %**. Además, se observa que *seqEA* fue capaz de obtener mejoras sobre el algoritmo ávido al cabo de unos pocos segundos de comenzada la ejecución, logrando alcanzar mejoras superiores en la mayoría de instancias de prueba al dejarlo ejecutando por un tiempo más prolongado.

En el caso de instancias con un número reducido de pasajeros (e.g., instancias chica #4 y Montevideo #1), *seqEA* logró pequeñas mejoras respecto a la solución alcanzada por el algoritmo ávido, mientras que en instancias de mayor dimensión alcanzó mejoras más amplias. Sin embargo, las ejecuciones de estas instancias de mayores dimensiones registraron tiempos más elevados, en el entorno de unos 15 segundos. Los resultados sugieren que existe la posibilidad de mejorar el algoritmo, tanto en la calidad de las soluciones alcanzadas, como en el desempeño computacional.

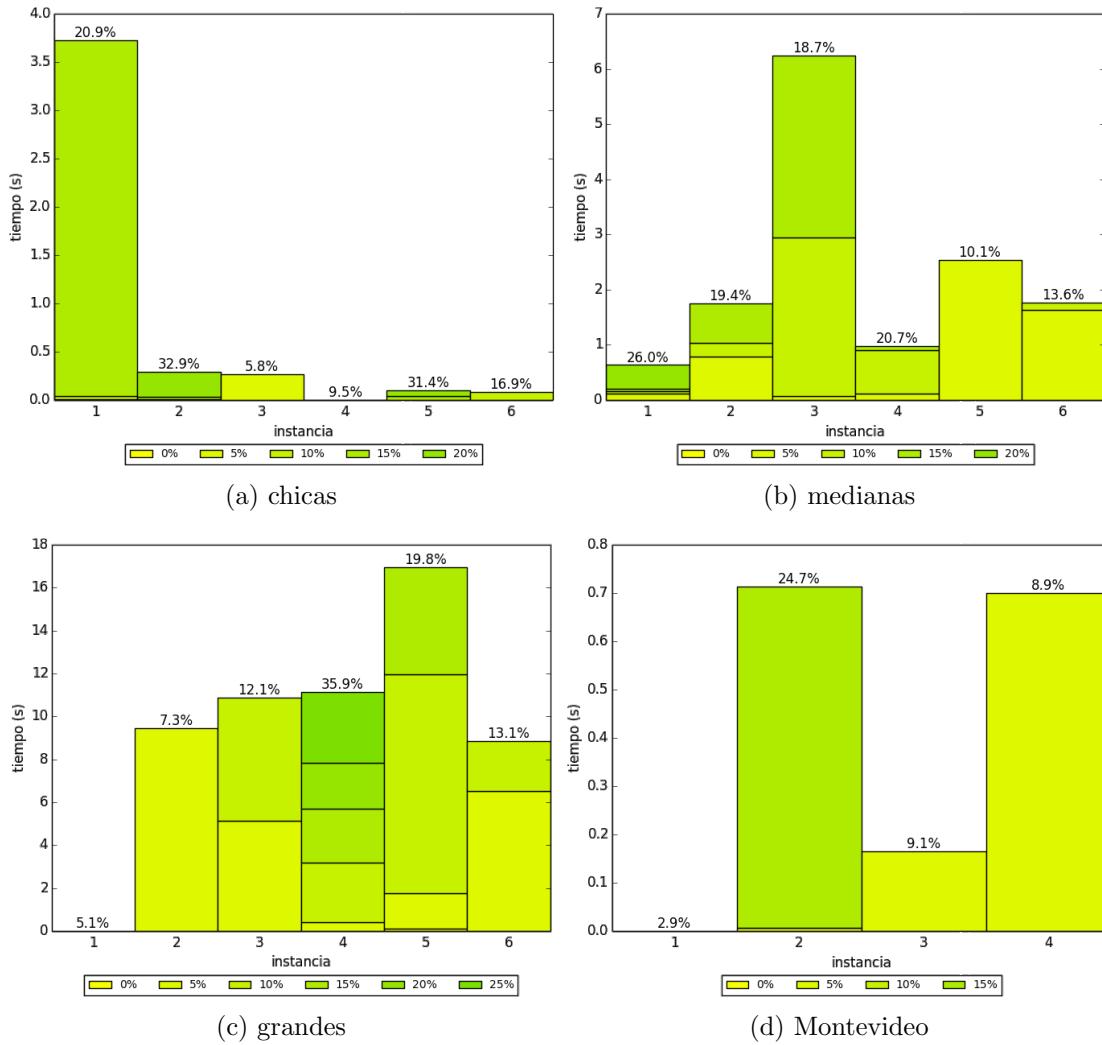
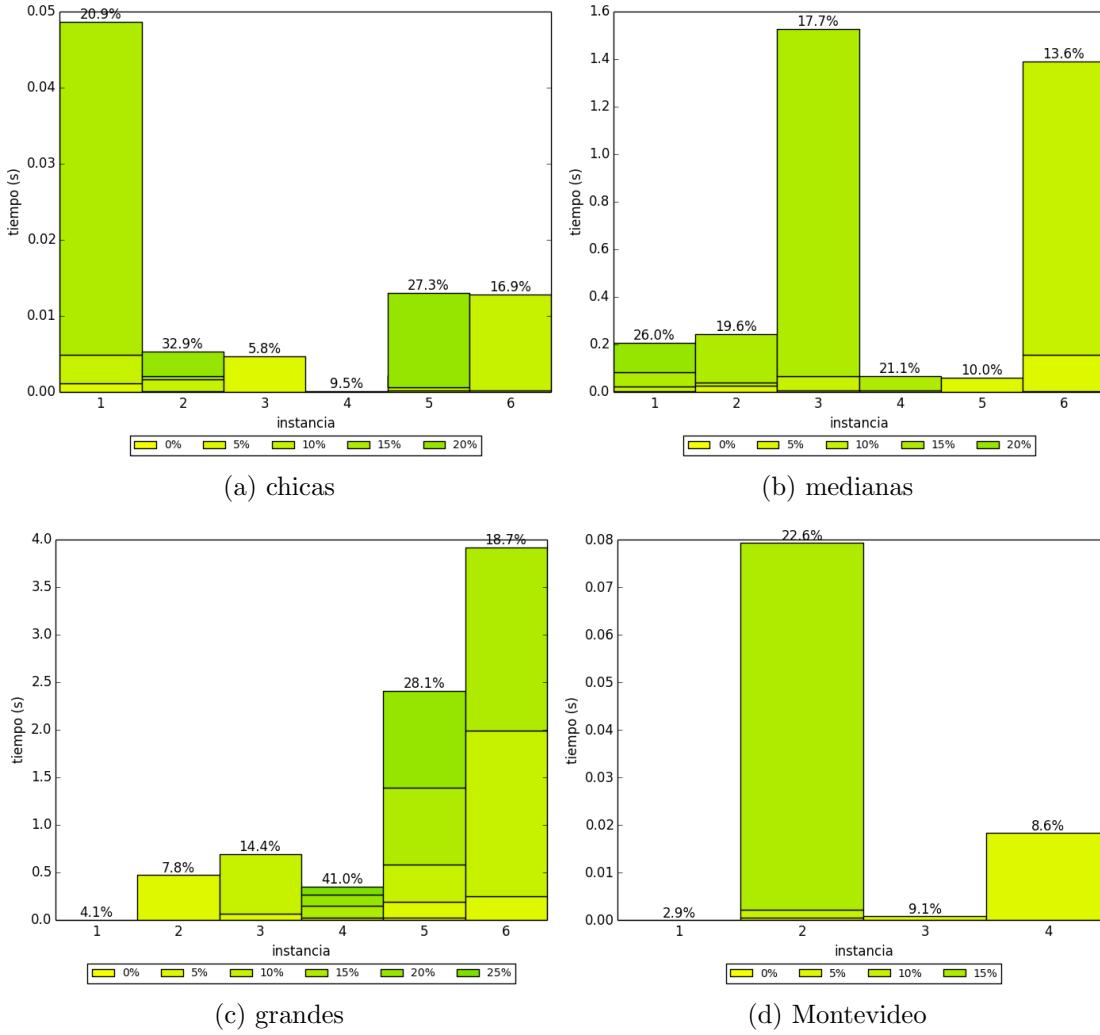


Figura 6.3: *seqEA*—mejoras logradas sobre el algoritmo ávido.

pμEA

El algoritmo *pμEA* fue evaluado sobre el mismo conjunto de instancias de prueba utilizado en la evaluación de *seqEA*, tal como se describió en la Sección 6.1.2. La Figura 6.4 muestra el tiempo necesario para mejorar la solución encontrada por el algoritmo ávido en cada una de las instancias de prueba. Se muestran los tiempos necesarios para alcanzar mejoras con un paso de 5 % y se indica sobre cada barra la mejora final alcanzada por *pμEA* al finalizar las 10.000 generaciones. Los valores reportados corresponden a aquellas ejecuciones que alcanzaron la máxima mejora final.

El algoritmo *pμEA* fue capaz de alcanzar mejoras sobre las soluciones alcanzadas por el algoritmo ávido en **todas** las instancias de prueba. En el mejor caso (instancia grande #4), fue capaz de mejorar en un **41.0 %** el costo de la solución encontrada por el algoritmo ávido. Por otra parte, se observa que las mejoras alcanzadas se lograron en menos de **un segundo** en el caso de las instancias chicas y de Montevideo, y en menos de **4 segundos** para las instancias medianas y grandes.

Figura 6.4: *pμEA*—mejoras logradas sobre el algoritmo ávido.

Resultados comparativos: *seqEA* vs. *pμEA*

A continuación, se presenta una comparativa de los resultados experimentales obtenidos por los dos algoritmos implementados para resolver el problema de viajes compartidos en su formulación monoobjetivo. La Tabla 6.4 permite comparar los valores de costo alcanzados por los algoritmos *seqEA* y *pμEA*. En dicha tabla, se reporta el mejor valor de costo alcanzado por cada algoritmo ($\min(c)$) junto a el valor promedio y la desviación estándar de los valores de costo alcanzados en las 30 ejecuciones independientes de cada algoritmo ($\bar{c} \pm std$). Se reporta además el *p*-valor resultante de aplicar el test de Kruskal–Wallis ($pvK-W$) sobre el conjunto de soluciones encontradas por cada algoritmo en las 30 ejecuciones independientes. Para aquellas instancias de prueba en las que el test de Kruskal–Wallis permite afirmar (con un nivel de confianza del 95 %) que uno de los dos algoritmos alcanza mejores valores de costo, se destaca en negrita el algoritmo que alcanzó el menor costo en promedio.

<i>instancias</i>		<i>seqEA</i>		<i>pμEA</i>		<i>pvK-W</i>
		<i>min(c)</i>	$\bar{c} \pm std$	<i>min(c)</i>	$\bar{c} \pm std$	
chicas	#1	164.4	165.6±2.0	164.4	164.4±0.0	0.2×10^{-3}
	#2	220.7	225.7±5.0	220.7	220.7±0.0	9.7×10^{-6}
	#3	160.4	160.4±0.0	160.4	160.4±0.0	1.0
	#4	181.3	181.3±0.1	181.3	182.4±1.9	0.5×10^{-1}
	#5	152.1	155.6±4.5	152.1	152.1±0.0	5.1×10^{-6}
	#6	118.4	119.6±2.5	118.4	118.4±0.0	0.1×10^{-1}
medianas	#1	211.9	216.0±4.2	211.9	211.9±0.0	5.2×10^{-11}
	#2	428.6	444.1±11.7	427.9	429.4±1.6	7.0×10^{-10}
	#3	361.7	378.7±6.5	364.5	370.4±4.5	1.6×10^{-6}
	#4	267.5	279.8±5.5	266.8	266.8±0.0	7.6×10^{-12}
	#5	479.3	487.1±6.5	479.6	479.8±0.2	5.1×10^{-7}
	#6	306.0	321.2±7.7	306.0	307.7±3.4	2.0×10^{-9}
grandes	#1	421.9	435.1±5.0	425.9	437.7±3.2	0.1×10^{-1}
	#2	479.3	489.9±4.3	477.0	481.1±2.3	1.9×10^{-9}
	#3	332.8	349.7±7.7	326.3	331.7±4.0	2.6×10^{-10}
	#4	351.1	390.7±26.3	338.4	344.8±6.1	5.1×10^{-11}
	#5	395.9	429.6±16.2	370.2	380.0±4.4	2.7×10^{-11}
	#6	360.8	382.4±8.1	343.8	350.6±3.8	2.6×10^{-11}
Montevideo	#1	168.4	168.4±0.0	168.4	168.4±0.0	1.0
	#2	319.3	331.2±3.8	324.9	328.6±3.2	5.6×10^{-6}
	#3	266.7	269.1±2.3	266.7	266.7±0.0	3.1×10^{-7}
	#4	303.2	304.7±0.5	304.1	304.5±0.4	0.1

Tabla 6.4: Tabla comparativa *seqEA* vs. *pμEA*.

Sobre un total de 22 instancias estudiadas, el test de Kruskal–Wallis permite afirmar que *pμEA* es capaz de encontrar mejores resultados que *seqEA* en 17 instancias. Únicamente en la instancia grande #1 *seqEA* alcanzó mejores soluciones que *pμEA*.

La Tabla 6.5 muestra un resumen comparativo de las mejoras alcanzadas sobre el algoritmo ávido y de los tiempos de ejecución, para los algoritmos *seqEA* y *pμEA*. Se reporta la mejora porcentual alcanzada sobre el algoritmo ávido y el tiempo de ejecución en segundos de cada AE, indicándose el mejor valor, el valor promedio y la desviación estándar. Se destacan en negrita los valores del algoritmo que alcanzó mejores resultados. Los resultados se muestran agrupados por familia de instancias, tomando en cuenta las 30 ejecuciones independientes de cada instancia de prueba. Los resultados detallados alcanzados en cada instancia de prueba se pueden consultar en las Tablas A.1–A.4.

Se observa que el algoritmo *pμEA* es capaz de alcanzar mayores porcentajes de mejoras sobre el algoritmo ávido que las alcanzadas por *seqEA* al considerar cada una de las familias de instancias de prueba, tanto considerando los valores en promedio como los mejores valores. En la familia de instancias grandes, las mejoras en costo obtenidas sobre el algoritmo ávido son de 41.0 % para *pμEA* vs. 35.9 % para *seqEA* en el mejor caso (16.7 % vs. 8.9 % en promedio). De acuerdo a la formulación del problema, el costo de los taxis es proporcional a la distancia recorrida, por lo que es relevante comparar las mejoras obtenidas por los AE en costo con las mejoras reportadas para la métrica de distancia en los trabajos relacionados. Se comprueba que los algoritmos *seqEA* y *pμEA* alcanzan mejoras sobre el algoritmo ávido significativamente mayores a las reportadas en los trabajos de Ma et al. (13%) [41] y Lin et al. (19%) [38] en la distancia recorrida.

Adicionalmente, *pμEA* requiere de un tiempo de ejecución significativamente menor que el de *seqEA*. La Figura 6.5 muestra la evolución del costo alcanzado por cada uno de los AE a lo largo de la ejecución de cuatro instancias de prueba representativas. Los valores reportados corresponden a aquellas ejecuciones que alcanzaron los mejores valores de costo finales. Se muestran los costos alcanzados cada un segundo de ejecución y el costo final alcanzado al cabo de las 10.000 generaciones.

<i>instancia</i>	<i>métrica</i>		<i>seqEA</i>	<i>pμEA</i>
chicas	mejora sobre alg. ávido (%)	máxima	32.9	32.9
		promedio	18.3	19.0
		std	9.3	10.4
	tiempo (s)	mínimo	3.5	0.6
		promedio	4.3	1.1
		std	0.5	0.3
medianas	mejora sobre alg. ávido (%)	máxima	26.0	26.0
		promedio	14.1	17.4
		std	5.7	5.4
	tiempo (s)	mínimo	5.2	1.0
		promedio	7.7	1.8
		std	1.5	0.5
grandes	mejora sobre alg. ávido (%)	máxima	35.9	41.0
		promedio	8.9	16.7
		std	7.7	12.3
	tiempo (s)	mínimo	11.2	1.5
		promedio	14.3	3.1
		std	3.7	1.0
Montevideo	mejora sobre alg. ávido (%)	máxima	24.7	22.6
		promedio	9.9	10.3
		std	6.4	6.3
	tiempo (s)	mínimo	3.6	0.6
		promedio	5.7	1.4
		std	1.3	0.4

Tabla 6.5: Comparativa: mejoras y tiempos de ejecución entre *seqEA* y *pμEA*.

Se puede observar que *pμEA* logra superar a *seqEA*, tanto en los valores de costo obtenidos como en el tiempo necesario para alcanzar dichos resultados. La Figura 6.5a (correspondiente a la instancia chica #1) muestra que *pμEA* es capaz de alcanzar el mismo valor de costo que *seqEA* al finalizar su ejecución, pero en menos de la mitad de tiempo. Además, se observa que *seqEA* se estanca en una solución sub-óptima durante gran parte de su ejecución. En las Figuras 6.5b y 6.5c (correspondientes a las instancias mediana #2 y grande #2, respectivamente) se observan diferencias aún más amplias. El algoritmo *pμEA* es capaz de alcanzar un mejor valor de costo final, en una fracción del tiempo de ejecución requerido por *seqEA*. Por último, en la Figura 6.5d (correspondiente a la instancia #1 de Montevideo) se puede apreciar que, si bien ambos algoritmos alcanzan un mismo valor de costo, *pμEA* tiene un tiempo de ejecución total menor que el de *seqEA*. En la familia de instancias grandes, el algoritmo *pμEA* alcanza una aceleración de 7,5x en el mejor caso (4,6x en promedio) respecto al algoritmo *seqEA*. Los tiempos de ejecución (en el entorno de los dos segundos) obtenidos por *pμEA* se encuentran dentro de lo esperable por un usuario final de una aplicación en línea que resuelva el problema de viajes compartidos en taxis, como la que se describe en el Capítulo 7.

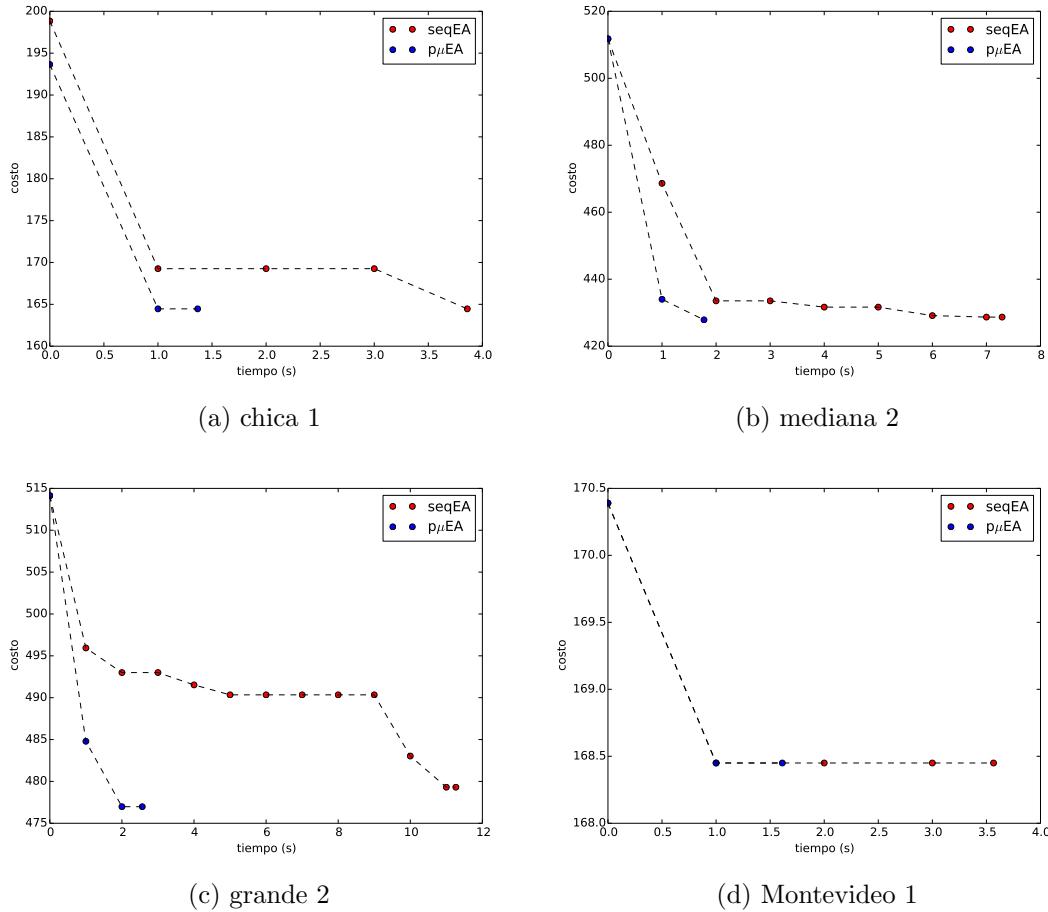


Figura 6.5: *seqEA* y *p μ EA*—evolución del costo a lo largo de una ejecución.

6.4. Variante multiobjetivo del PVCT

En la presente sección se detalla el análisis experimental correspondiente a la variante multiobjetivo del problema de viajes compartidos en taxis.

6.4.1. Entorno de ejecución

Las ejecuciones del algoritmo *p μ MOEA/D* fueron realizadas utilizando un total de 24 núcleos de procesador de un servidor HP Proliant DL585 conformado por procesadores AMD Opteron 6272 2.09GHz con 48GB de memoria RAM disponibles. La evaluación experimental del algoritmo *NSGA-II* fue realizada en un servidor HP Proliant DL385 G7, utilizando un único núcleo de un procesador AMD Opteron 6172 2.10GHz con 72GB de memoria RAM disponibles.

6.4.2. Configuración paramétrica

A continuación, se brinda un breve resumen de las configuraciones de parámetros evaluadas. Los resultados detallados se pueden consultar en los artículos correspondientes a cada AE en el Apéndice B.

p_μMOEA/D

Para la configuración paramétrica del algoritmo *p_μMOEA/D* se utilizaron 4 instancias del problema, distintas a las utilizadas en la evaluación experimental de forma de evitar sesgo. Luego de una evaluación inicial, el tamaño de la micro población se estableció en 15 individuos. El operador de migración se fijó de forma de seleccionar dos individuos de cada isla mediante una selección por torneo entre dos individuos donde se selecciona al mejor, y enviarlos a la isla vecina, donde reemplazan a los peores individuos. El operador de migración se aplica cada 1000 generaciones.

El ajuste paramétrico se centró en estudiar las probabilidades de recombinación (p_C , valores candidatos 0.6, 0.75 y 0.95) y de mutación (p_M , valores candidatos 0.001, 0.01 y 0.1). Se estudiaron todas las combinaciones de valores sobre las cuatro instancias de calibración, realizando 30 ejecuciones independientes de 20000 generaciones cada una. Los resultados del análisis paramétrico indican que los mejores resultados para las instancias estudiadas se obtienen utilizando $p_C = 0.75$ y $p_M = 0.1$.

NSGA-II

Para la configuración paramétrica del algoritmo *NSGA-II* se utilizaron 4 instancias del problema de dimensión media, distintas a las utilizadas en el análisis experimental de forma de evitar sesgo. Luego de experimentos iniciales, el tamaño de la población se fijó en 80 individuos. El ajuste paramétrico se concentró en estudiar las probabilidades de recombinación (p_C , valores candidatos 0.6, 0.75 y 0.95) y de mutación (p_M , valores candidatos 0.001, 0.01 y 0.1). Para cada combinación de parámetros se realizaron 30 ejecuciones independientes de 5000 generaciones cada una. Los resultados alcanzados sugieren que usar $p_C = 0.75$ y $p_M = 0.1$ permite encontrar las mejores soluciones para las instancias del problema estudiadas.

6.4.3. Métricas multiobjetivo

Una gran cantidad de métricas han sido propuestas en la literatura para evaluar algoritmos evolutivos multiobjetivo [14, 17], considerando tanto la convergencia hacia el frente de Pareto ideal como la amplitud del frente de soluciones hallado. A continuación, se definen las métricas utilizadas en la evaluación experimental:

- El número de soluciones no dominadas distintas (#ND).
- Distancia generacional (DG): la suma (normalizada) de distancias entre las soluciones no dominadas del frente de Pareto hallado por el algoritmo (soluciones v en el frente de Pareto aproximado P^*) y un conjunto de puntos uniformemente distribuidos en el frente de Pareto real (Ecuación 6.1). Valores de distancia generacional menores indican una mejor aproximación al frente de Pareto real.
- Spacing: evalúa la dispersión de soluciones no dominadas en el frente de Pareto calculado (Ecuación 6.2).
- Spread: evalúa la dispersión de las soluciones no dominadas en el frente de Pareto calculado, incluyendo la distancia hacia los extremos del frente de Pareto real (Ecuación 6.3). Valores de spread más cercanos a cero indican una distribución más equiespaciada de las soluciones no dominadas.

- Hipervolumen: el volumen (en el espacio de las funciones objetivo) cubierto por el frente de Pareto calculado.
- Hipervolumen relativo (RHV): la tasa entre los volúmenes (en el espacio de las funciones objetivo) cubierto por el frente de Pareto calculado y el frente de Pareto real. El valor ideal de hipervolumen relativo es 1.

$$\frac{\sum_{v \in P^*} d(v, P)}{|P^*|} \quad (6.1)$$

$$\sqrt{\frac{\sum_{i=1}^{ND} (\bar{d} - d_i)^2}{ND - 1}} \quad (6.2)$$

$$\frac{\sum_{h=1}^k d_h^e + \sum_{i=1}^{ND} |\bar{d} - d_i|}{\sum_{h=1}^k d_h^e + ND \times \bar{d}} \quad (6.3)$$

En las Ecuaciones 6.2 y 6.3, d_i es la distancia entre la i -ésima solución en el frente de Pareto calculado y su solución vecina más cercana, \bar{d} es el promedio de todas las d_i , y d_h^e es la distancia entre el extremo de la h -ésima función objetivo en el frente de Pareto real y el punto más cercano en el frente de Pareto calculado. El frente de Pareto real—que es desconocido para las instancias del problema estudiadas—es aproximado por el conjunto de soluciones no dominadas encontradas para cada instancia del problema en el total de ejecuciones realizadas para cada uno de los algoritmos.

6.4.4. Algoritmos ávidos

Con el fin de comparar los resultados obtenidos por los AE, se diseñaron e implementaron dos algoritmos ávidos, uno para cada uno de los objetivos del problema.

Algoritmo ávido para costo El algoritmo ávido que busca optimizar el costo total de la solución es similar al descrito para el problema monoobjetivo (descrito en la Sección 6.3.3). Sin embargo, para la variante multiobjetivo es necesario tener en cuenta las distintas capacidades de los vehículos y su disponibilidad. La única diferencia respecto al algoritmo para objetivo único (Algoritmo 9) se da al momento de chequear si un determinado taxi llegó al máximo número de pasajeros permitidos. En lugar de utilizar una capacidad fija C_{MAX} , se utiliza la mayor capacidad para la cual hay al menos un taxi disponible y se actualiza el número de taxis disponibles de dicha capacidad a medida que se completan los taxis.

Algoritmo ávido para demora El algoritmo ávido que busca minimizar la demora percibida por los pasajeros se presenta en el Algoritmo 10. En primer lugar, se crea un nuevo taxi para cada uno de los pasajeros con mayor nivel de apuro y se les asigna el primer lugar de dicho taxi (de forma que sus destinos sean los primeros en ser visitados). Luego, se recorre la lista de pasajeros aún no asignados, comenzando por aquellos con mayor nivel de apuro y asignándolos al taxi que minimice su demora (i.e., el taxi con el menor tiempo de recorrido entre el destino del último pasajero asignado y el destino del pasajero a asignar). Cuando un taxi tiene tantos pasajeros como la máxima capacidad disponible, se considera “completo” y deja de ser considerado al momento de ubicar a los pasajeros aún no asignados. En este caso, se actualiza la cantidad de taxis disponibles de esa capacidad.

Algoritmo 10 Algoritmo ávido para minimizar demora en el PVCT multiobjetivo.

```

desde i = 0 a i = cantidad_pasajeros hacer
    si tolerancia(pasajero[i]) == 0 entonces
        taxi = nuevoTaxi();
        agregarPasajero(pasajeros[i], taxi);
        agregarTaxi(taxi, solución);
    fin si
fin desde
    sin_asignar = pasajerosSinAsignar();
    mientras no_vacío(sin_asignar) hacer
        pasajero = pasajeroMasApurado(sin_asignar);
        taxi = mejorTaxi(solución, pasajero);
        si taxi == null entonces
            taxi = nuevoTaxi();
            agregarPasajero(pasajero, taxi);
            agregarTaxi(taxi, solución);
        en otro caso
            agregarPasajero(pasajero, taxi);
        fin si
        sin_asignar = pasajerosSinAsignar();
    fin mientras
retornar solución;

```

6.4.5. Resultados numéricos

A continuación, se presentan los resultados numéricos del análisis experimental realizado sobre los algoritmos $p\mu MOEA/D$ y $NSGA-II$, correspondientes a la variante multiobjetivo del problema de viajes compartidos en taxis. Se realizaron 30 ejecuciones independientes de cada AE para cada una de las instancias descritas en la Sección 6.1.

$p\mu MOEA/D$

La Tabla 6.6 muestra los resultados (agrupados por familia de instancias) obtenidos por $p\mu MOEA/D$ en las métricas definidas en la Sección 6.4.3, reportándose los valores promedio, la desviación estándar y el mejor valor alcanzado entre paréntesis. Los resultados detallados por instancia pueden ser consultados en las Tablas A.5–A.8.

	#ND	DG	spacing	spread	RHV
chicas	8.5 ± 2.1 (16.0)	3.1 ± 2.5 (0.0)	740.2 ± 746.3 (58.1)	0.6 ± 0.2 (0.1)	0.9 ± 0.1 (1.0)
medianas	9.1 ± 2.2 (19.0)	5.7 ± 2.5 (0.0)	1448.5 ± 1064.1 (141.6)	0.6 ± 0.1 (0.1)	0.9 ± 0.1 (1.0)
grandes	8.5 ± 2.2 (17.0)	7.9 ± 3.4 (2.0)	2917.2 ± 2041.5 (175.3)	0.6 ± 0.1 (0.0)	0.8 ± 0.1 (1.0)
Montevideo	8.0 ± 2.1 (14.0)	3.0 ± 2.0 (0.0)	663.5 ± 542.4 (61.5)	0.6 ± 0.2 (0.0)	0.9 ± 0.0 (1.0)

Tabla 6.6: Métricas multiobjetivo para $p\mu MOEA/D$.

Los valores pequeños de DG indican una buena convergencia al frente de Pareto ideal. Simultáneamente, los bajos valores de spread indican que $p\mu MOEA/D$ es capaz de conservar buena diversidad en el frente de soluciones halladas. Los valores de RHV cercanos a 1 refuerzan las afirmaciones previas sobre convergencia y diversidad de solu-

ciones; sin embargo, la cantidad de soluciones no dominadas resulta baja para el tamaño de población utilizado, lo que sugiere que aún hay lugar para mejoras.

La Figura 6.6 muestra el frente de soluciones hallado por $p\mu MOEA/D$ cada 2500 generaciones y las soluciones de los algoritmos ávidos, para cuatro instancias representativas. Los frentes corresponden a las ejecuciones que alcanzaron mayor cantidad de puntos no dominados. Se puede apreciar el avance del frente de soluciones hacia un hipotético frente de Pareto ideal del problema con el paso de las generaciones. Además, se observa una gran mejora de $p\mu MOEA/D$ sobre las soluciones de los algoritmos ávidos.

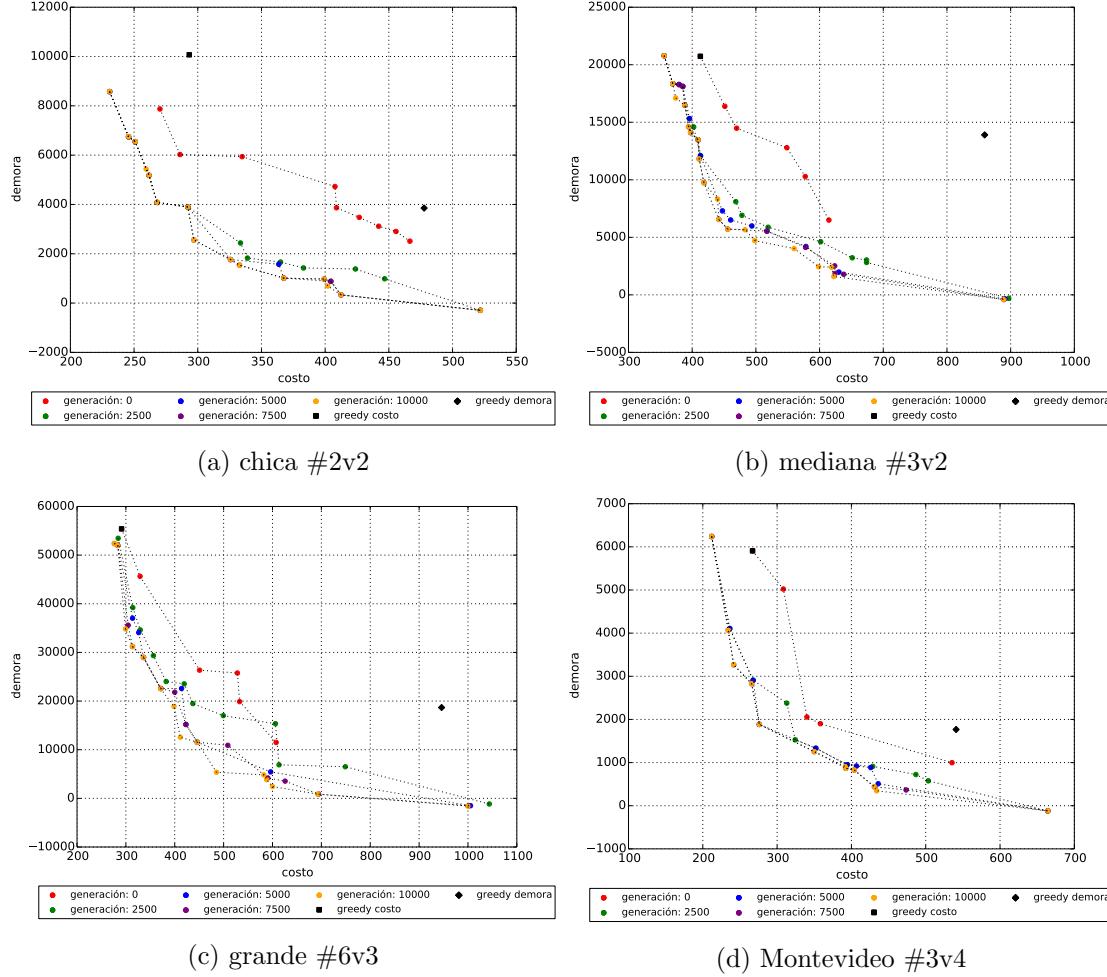
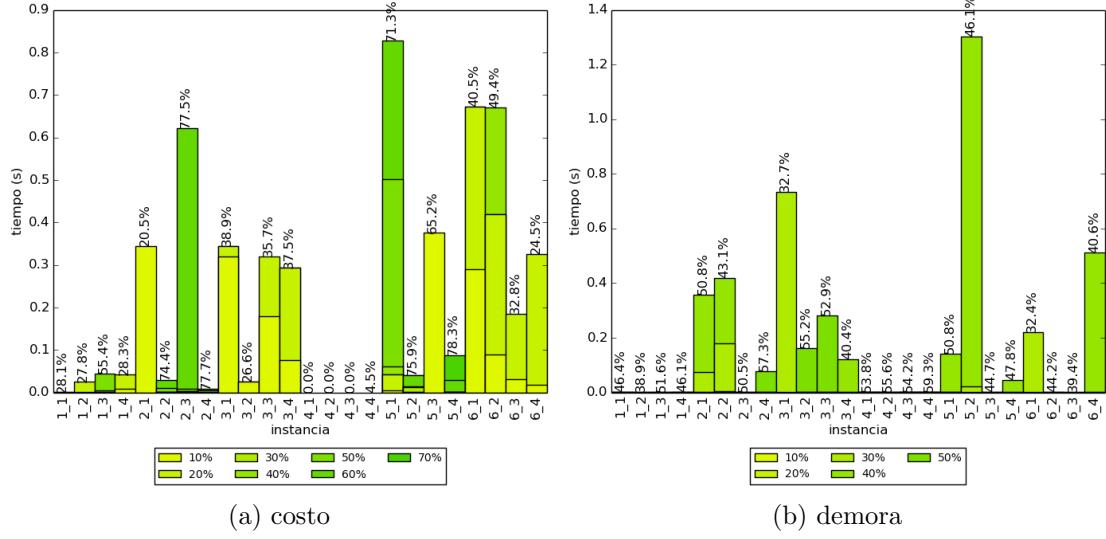
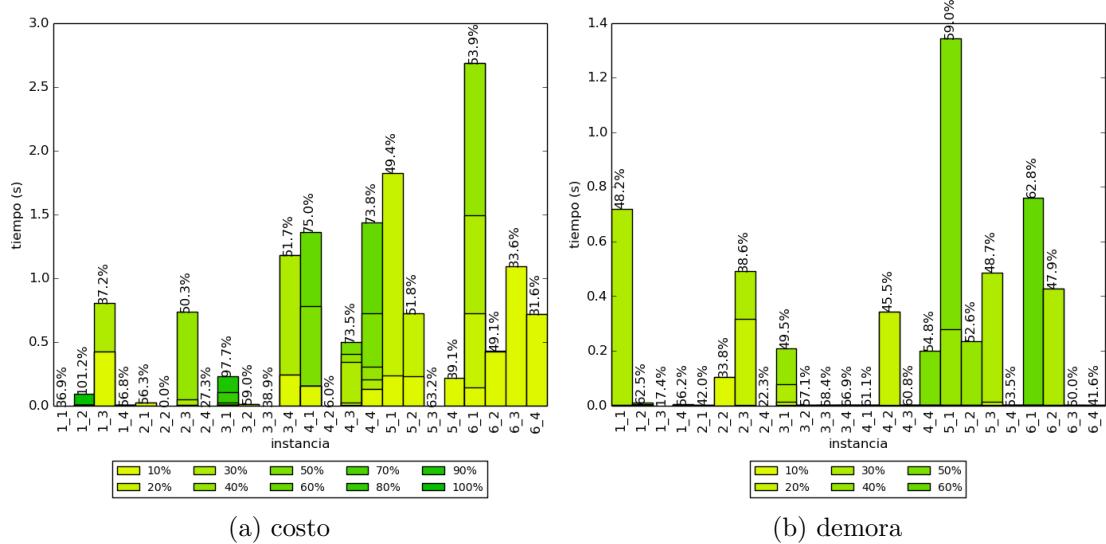
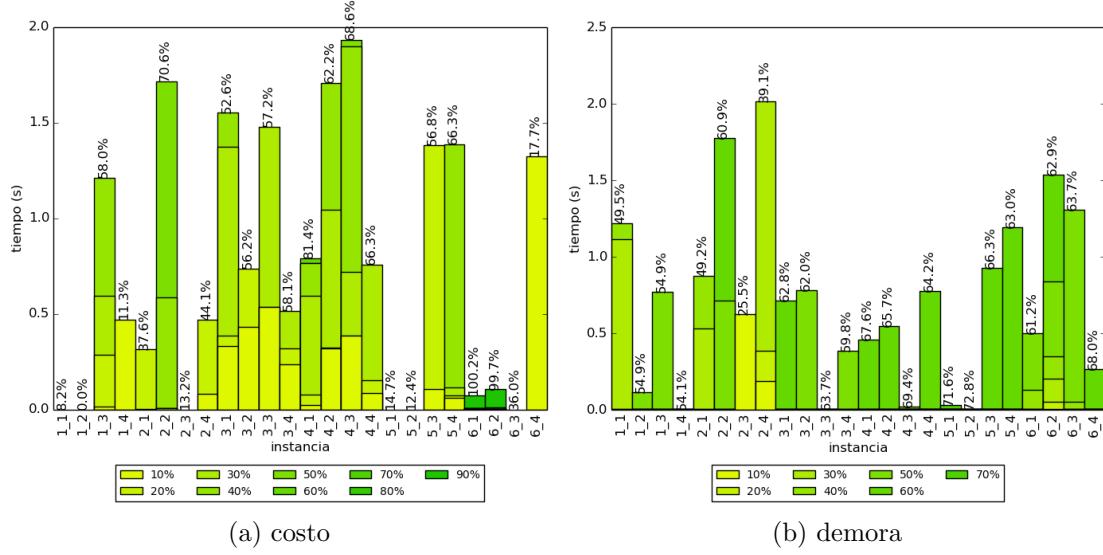
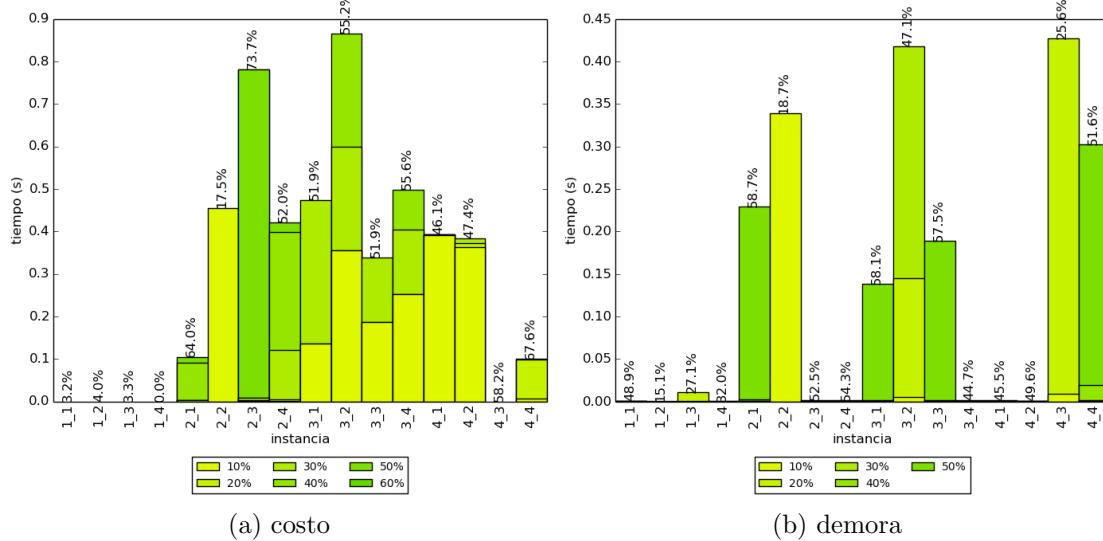


Figura 6.6: $p\mu MOEA/D$ —frentes de Pareto por generación.

Las Figuras 6.7 a 6.10 muestran el tiempo necesario para mejorar las soluciones encontradas por los algoritmos ávidos en cada instancia de prueba. La mejora sobre el algoritmo ávido para costo representa la mejora en demora obtenida por $p\mu MOEA/D$, en una solución con el mismo costo que la encontrada por dicho algoritmo ávido. Análogamente, la mejora sobre el algoritmo ávido de demora expresa la mejora en costo obtenida por $p\mu MOEA/D$, para una solución con la misma demora que la encontrada por el algoritmo ávido de demora. Se muestran los tiempos necesarios para alcanzar mejoras con un paso de 10 % y se indica, sobre cada barra, la mejora final alcanzada por $p\mu MOEA/D$ al completar las 10.000 generaciones. Los valores reportados corresponden a aquellas ejecuciones que alcanzaron el mayor valor de mejora al finalizar la ejecución.

Figura 6.7: $p\mu MOEA/D$ —mejoras sobre los algoritmos ávidos en instancias chicas.Figura 6.8: $p\mu MOEA/D$ —mejoras sobre los algoritmos ávidos en instancias medianas.

Figura 6.9: $p\mu$ MOEA/D—mejoras sobre los algoritmos ávidos en instancias grandes.Figura 6.10: $p\mu$ MOEA/D—mejoras sobre los algoritmos ávidos en instancias Montevideo.

El algoritmo $p\mu$ MOEA/D es capaz de alcanzar mejoras sobre las soluciones calculadas por los algoritmos ávidos en todas las instancias estudiadas. En el mejor caso, es capaz de mejorar en un **101.2 %** la demora (101.2 % en promedio), manteniendo el mismo costo que el alcanzado por el algoritmo ávido para costo (instancia mediana #1v2), y mejorar hasta en un **72.8 %** el costo de la solución obtenida por el algoritmo ávido de demora (70.2 % en promedio), manteniendo la misma demora (instancia grande #5v2). Además, se observa que las mejoras alcanzadas se logran, en la mayoría de los casos, en unos pocos segundos de ejecución. Esto sugiere la posibilidad de integrar el algoritmo en una aplicación de usuario en tiempo real, como la que se describe en el Capítulo 7.

NSGA-II

La Tabla 6.7 muestra los resultados alcanzados por el algoritmo *NSGA-II* en las métricas de optimización multiobjetivo definidas en la Sección 6.4.3. Los resultados se presentan agrupados por familia de instancias. Se muestran los valores promedio junto a la desviación estándar y se indica el mejor valor alcanzado entre paréntesis. Los resultados detallados para cada instancia de prueba se pueden consultar en las Tablas A.9–A.12.

	#ND	DG	spacing	spread	RHV
chicas	32.6±9.5 (55.0)	0.3±0.6 (0.0)	236.2±222.7 (43.2)	0.9±0.1 (0.7)	1.0±0.0 (1.0)
medianas	54.5±4.2 (67.0)	1.0±0.7 (0.0)	193.6±202.4 (26.2)	0.7±0.2 (0.4)	1.0±0.0 (1.0)
grandes	55.2±3.5 (67.0)	1.8±1.1 (0.4)	243.6±229.8 (26.4)	0.7±0.2 (0.4)	1.0±0.0 (1.0)
Montevideo	43.9±16.4 (61.0)	0.4±0.5 (0.0)	142.3±143.2 (20.8)	0.8±0.1 (0.5)	1.0±0.0 (1.0)

Tabla 6.7: Métricas multiobjetivo para *NSGA-II*.

Se observa que la cantidad de puntos no dominados hallada por *NSGA-II* es significativamente mayor que la alcanzada por el algoritmo *pμMOEA/D*, llegando a un máximo de 67 puntos no dominados distintos, en una población de 80 individuos. Los valores de distancia generacional son sumamente bajos, lo que evidencia una buena convergencia al frente de Pareto ideal. Los valores de spacing alcanzados por *NSGA-II* son significativamente menores que los de *pμMOEA/D*, lo que indica una mejor dispersión de las soluciones a lo largo del frente de Pareto calculado por el algoritmo. Los valores de hipervolumen relativo alcanzados también superan a los de *pμMOEA/D*, reforzando las observaciones previas sobre convergencia y diversidad.

La Figura 6.11 muestra los frentes de Pareto alcanzados para cuatro instancias representativas del conjunto de instancias de prueba. Dichos frentes corresponden a aquellas ejecuciones que alcanzaron mayor cantidad de puntos no dominados al finalizar la ejecución. Se muestran los frentes alcanzados cada 2500 generaciones junto a las soluciones halladas por los algoritmos ávidos.

Se puede apreciar el avance del frente de soluciones no dominadas calculadas por el algoritmo hacia un hipotético frente de Pareto ideal con el paso de las generaciones. Se observan también las significativas mejoras alcanzadas por el algoritmo *NSGA-II* sobre ambos algoritmos ávidos. Para la instancia chica #2v1, *NSGA-II* no presenta grandes diferencias en los frentes alcanzados con el paso de las generaciones, lo que sugiere que el algoritmo es capaz de calcular una buena aproximación al frente de Pareto ideal rápidamente. Sin embargo, en instancias más grandes se observa que el frente de soluciones continúa mejorando—aunque sea en pequeña medida—hasta el final de la ejecución. Esto plantea la disyuntiva acerca de la posibilidad de ejecutar el algoritmo durante una mayor cantidad de generaciones, en busca de obtener mejores resultados. Como se detalla en el Capítulo 7, el uso del algoritmo en una solución web y/o móvil para usuarios finales exige un tiempo de respuesta rápido, que no afecte negativamente la experiencia del usuario. Por esta razón, se impone el límite estricto de 10.000 generaciones en el criterio de parada del algoritmo, aúñ a expensas de posibles mejoras en los resultados alcanzados.

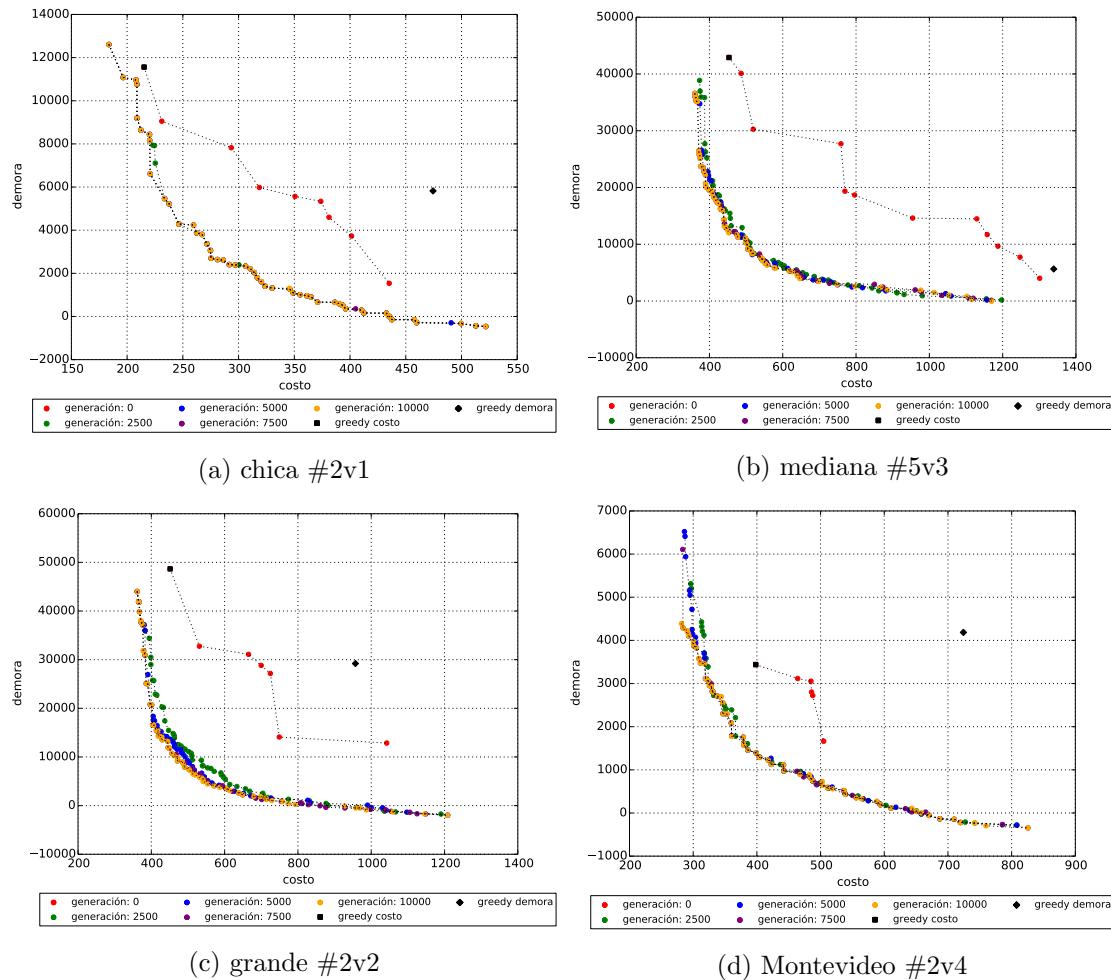


Figura 6.11: *NSGA-II*—frentes de Pareto por generación.

Por último, las Figuras 6.12 a 6.15 muestran el tiempo necesario para mejorar las soluciones encontradas por los algoritmos ávidos en cada instancia de prueba. La mejora sobre el algoritmo ávido de costo representa la mejora en demora obtenida por *NSGA-II*, manteniendo el mismo costo que el algoritmo ávido que optimiza el costo. La mejora sobre el algoritmo ávido de demora representa la mejora en costo obtenida por *NSGA-II*, en una solución con la misma demora que la encontrada por el algoritmo ávido de demora. Se muestran los tiempos necesarios para alcanzar mejoras con un paso de 10 % y se indica sobre cada barra la mejora final alcanzada por *NSGA-II* al finalizar las 10.000 generaciones. Los valores reportados corresponden a aquellas ejecuciones que alcanzaron la mayor mejora al finalizar la ejecución.

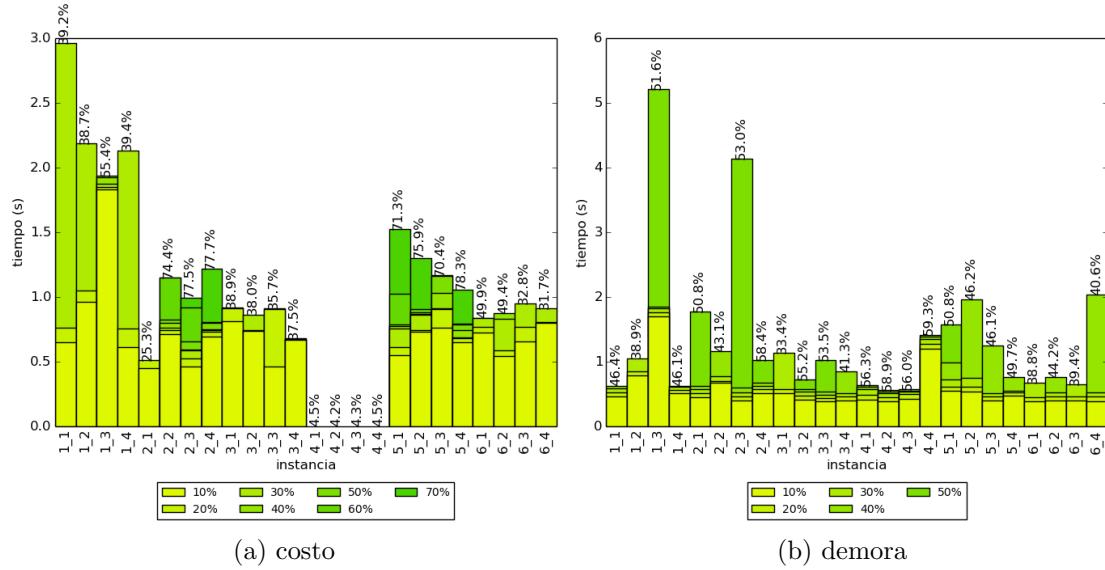


Figura 6.12: *NSGA-II*—mejoras sobre los algoritmos ávidos en instancias chicas.

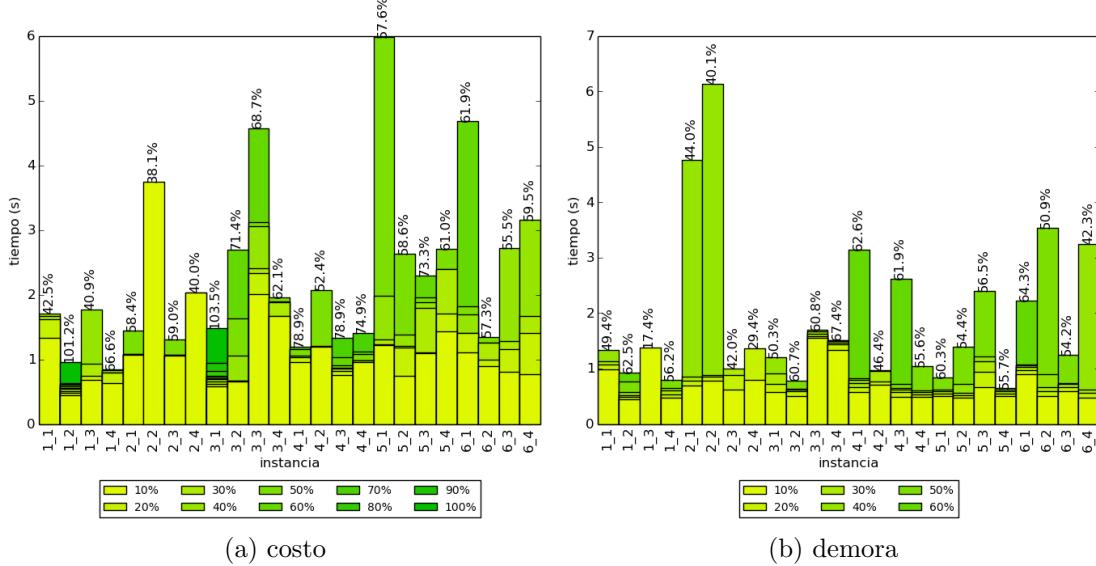
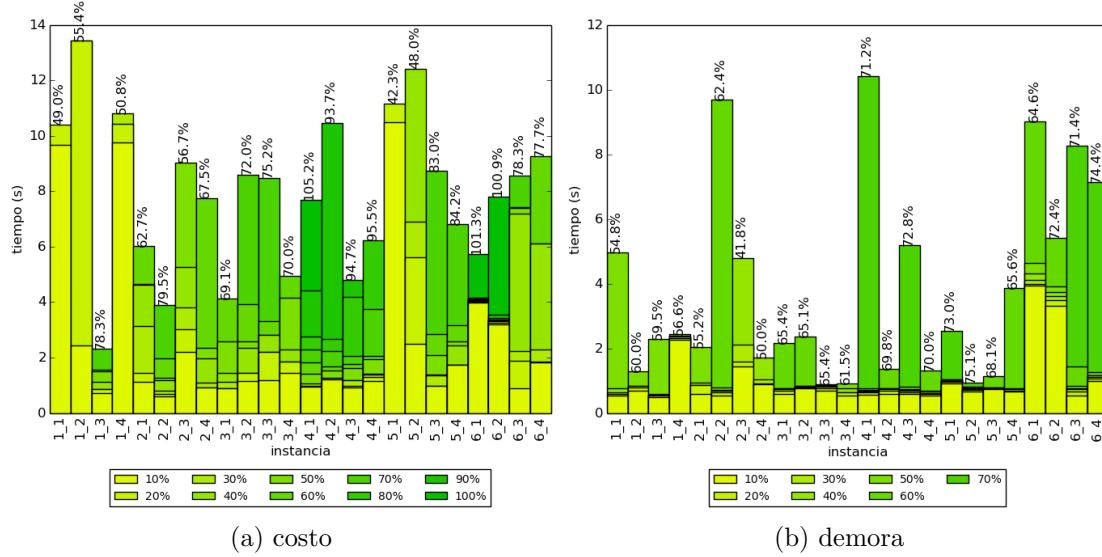
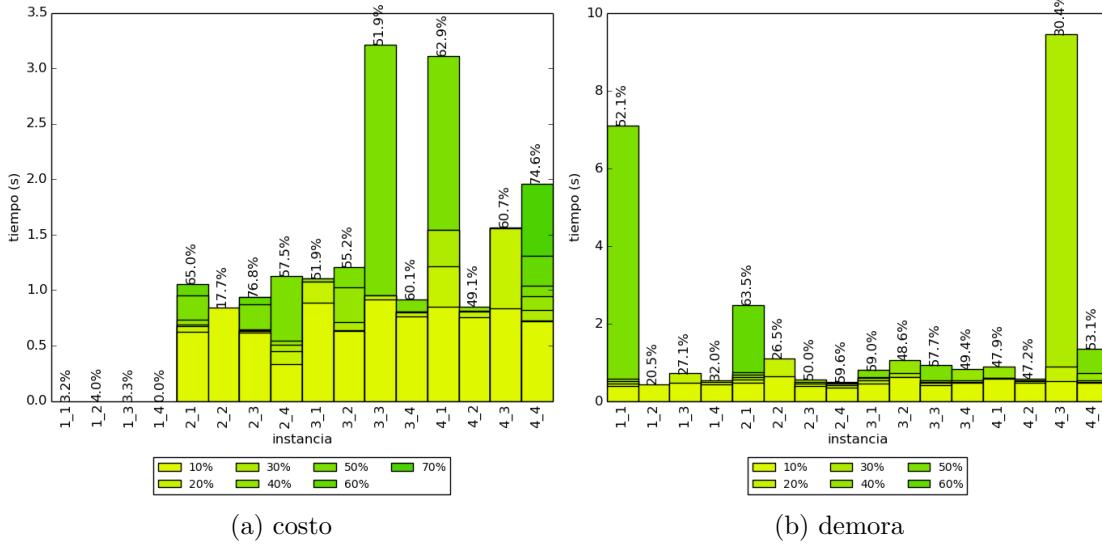


Figura 6.13: *NSGA-II*—mejoras sobre los algoritmos ávidos en instancias medianas.

Figura 6.14: *NSGA-II*—mejoras sobre los algoritmos ávidos en instancias grandes.Figura 6.15: *NSGA-II*—mejoras sobre los algoritmos ávidos en instancias Montevideo.

El algoritmo *NSGA-II* es capaz de alcanzar mejoras sobre las soluciones alcanzadas por los algoritmos ávidos en todas las instancias de prueba estudiadas. En el mejor caso, es capaz de mejorar en un **105.2 %** la demora (102.3 % en promedio), manteniendo el mismo costo que el alcanzado por el algoritmo ávido para costo (instancia grande #4v1), y mejorar hasta en un **75.1 %** el costo de la solución obtenida por el algoritmo ávido de demora (73.0 % en promedio), manteniendo la misma demora (instancia grande #5v1). Las mejoras alcanzadas se logran en un tiempo de ejecución del orden de un par de segundos en instancias chicas, y del orden de los 10 segundos en instancias grandes.

Resultados comparativos: $p\mu\text{MOEA}/D$ vs. NSGA-II

Es importante comparar los algoritmos implementados, tanto en la calidad de los resultados alcanzados como en su desempeño computacional, para su utilización en una herramienta de optimización que resuelva el problema y para su integración en una aplicación en línea que ofrezca una buena experiencia de uso.

La Tabla 6.8 muestra un resumen comparativo de los resultados alcanzados por los algoritmos $p\mu\text{MOEA}/D$ y NSGA-II . Se muestran las mejoras porcentuales obtenidas por cada algoritmo sobre los algoritmos ávidos, el tiempo de ejecución en segundos y el hipervolumen alcanzado. Se utiliza el hipervolumen (definido en la Sección 6.4.3) para comparar los resultados de ambos algoritmos, ya que es una métrica que considera ambos propósitos de los MOEA: la convergencia hacia el frente de Pareto ideal del problema y la diversidad de soluciones no dominadas halladas. Los valores de promedio, desviación estándar y mejor valor reportados, son los calculados considerando las 30 ejecuciones de cada instancia de la familia de instancias de prueba.

		mejora greedy (%)		tiempo(s)	HV($\times 10^6$)
		costo	demora		
chicas	$p\mu\text{MOEA}/D$	33.3 \pm 24.3 (78.3)	44.0 \pm 7.8 (59.3)	1.3 \pm 0.4 (0.7)	1.2 \pm 1.3 (4.8)
	NSGA-II	43.9 \pm 24.3 (78.3)	47.6 \pm 7.3 (59.3)	8.2 \pm 0.4 (7.3)	1.3 \pm 1.4 (5.2)
medianas	$p\mu\text{MOEA}/D$	34.7 \pm 27.3 (101.2)	44.8 \pm 15.0 (66.9)	2.1 \pm 0.7 (1.1)	5.8 \pm 4.3 (19.3)
	NSGA-II	59.7 \pm 19.2 (103.5)	50.7 \pm 11.6 (67.4)	10.1 \pm 0.9 (8.3)	7.2 \pm 5.3 (23.2)
grandes	$p\mu\text{MOEA}/D$	33.3 \pm 27.1 (100.2)	55.8 \pm 12.5 (72.8)	3.4 \pm 1.2 (1.9)	10.0 \pm 8.9 (39.2)
	NSGA-II	67.7 \pm 23.9 (105.2)	62.5 \pm 8.3 (75.1)	13.6 \pm 1.8 (11.6)	13.2 \pm 10.0 (43.1)
Montevideo	$p\mu\text{MOEA}/D$	27.2 \pm 20.2 (73.7)	38.1 \pm 16.6 (58.7)	1.7 \pm 0.6 (0.8)	2.5 \pm 1.8 (6.5)
	NSGA-II	42.7 \pm 26.3 (76.8)	44.1 \pm 13.9 (63.5)	9.0 \pm 0.8 (7.3)	2.9 \pm 2.1 (7.1)

Tabla 6.8: Tabla comparativa: $p\mu\text{MOEA}/D$ vs. NSGA-II .

Los resultados detallados por instancia se muestran en las Tablas A.13–A.16, donde se reporta además el p -valor correspondiente al test de Shapiro–Wilk [50] realizado sobre los valores de hipervolumen para contrastar normalidad. Los resultados del test de Shapiro–Wilk (con un nivel de confianza del 95 %) permiten rechazar la hipótesis nula de que los valores de hipervolumen alcanzados en las 30 ejecuciones independientes siguen una distribución normal, para un número considerable de instancias. En la última columna de las Tablas A.13–A.16 se reporta el p -valor correspondiente al test de Kruskal–Wallis [35] sobre los valores de hipervolumen, con el fin de comparar los resultados obtenidos por $p\mu\text{MOEA}/D$ con los obtenidos por NSGA-II .

Las mejoras sobre los algoritmos ávidos alcanzadas por NSGA-II son significativamente mayores que las alcanzadas por $p\mu\text{MOEA}/D$, tanto en mejor valor como en promedio. Por ejemplo, en la familia de instancias grandes, la mejora sobre el algoritmo ávido de costo fue de 105.2 % para NSGA-II vs. 100.2 % para $p\mu\text{MOEA}/D$ en el mejor caso (67.7 % vs. 33.3 % en promedio) y la mejora sobre el algoritmo ávido de demora fue de 75.1 % para NSGA-II vs. 72.8 % para $p\mu\text{MOEA}/D$ en el mejor caso (62.5 % vs. 55.8 % en promedio). El hipervolumen alcanzado por NSGA-II también supera al alcanzado por $p\mu\text{MOEA}/D$ en todas las dimensiones de las instancias estudiadas, lo que indica mayor amplitud en el frente de soluciones hallado y mejor convergencia hacia el frente de Pareto ideal. Los resultados del test de Kruskal–Wallis permiten afirmar que el algoritmo NSGA-II alcanza mejores valores de hipervolumen que el algoritmo $p\mu\text{MOEA}/D$ en todas las instancias de prueba estudiadas.

La Figura 6.16 reporta la relación entre la mejora sobre los algoritmos ávidos y el tiempo de ejecución de los algoritmos $p\mu MOEA/D$ y $NSGA-II$. Los valores se encuentran agrupados por familia de instancias y corresponden a los promedios obtenidos al considerar las 30 ejecuciones independientes de cada instancia.

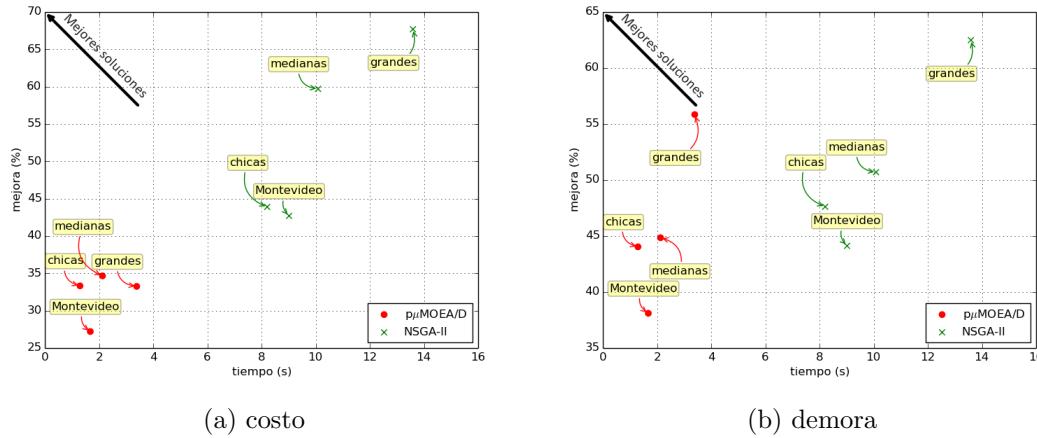


Figura 6.16: $p\mu MOEA/D$ y $NSGA-II$ —mejora sobre algoritmos ávidos vs. tiempo de ejecución.

El enfoque multiobjetivo explícito de $NSGA-II$ permite alcanzar mejores resultados que el enfoque de descomposición de dominio de $p\mu MOEA/D$. Sin embargo, los tiempos de ejecución del algoritmo $p\mu MOEA/D$ son significativamente menores a los alcanzados por $NSGA-II$. Es preciso remarcar que $p\mu MOEA/D$ hace uso de múltiples procesadores en una infraestructura de computación de alto desempeño, mientras que $NSGA-II$ fue ejecutado utilizando un único núcleo de procesador. En una aplicación en línea como la descrita en el Capítulo 7, es necesario calcular soluciones en tiempos de ejecución reducidos. Por esta razón, se plantea como una de las líneas de trabajo futuro la implementación de una variante del algoritmo $NSGA-II$ que utilice el modelo de islas distribuidas. De esta forma, se espera obtener las mejores características de ambas estrategias: los buenos resultados en calidad de soluciones asociados al enfoque multiobjetivo explícito de $NSGA-II$ y los tiempos de ejecución reducidos asociados al paralelismo del modelo de islas. Al momento de diseñar una solución que integre ambos enfoques, es necesario considerar que en $NSGA-II$ el cálculo de la distancia de crowding se realiza sobre el total de la población. Por lo tanto, una implementación que utilice el modelo de subpoblaciones distribuidas debe incluir un mecanismo centralizado para el cálculo de la distancia de crowding, o bien modificar la estrategia empleada para preservar diversidad.

En la Figura 6.17 se muestran los frentes de soluciones no dominadas alcanzados por los AE y las soluciones alcanzadas por los algoritmos ávidos, para una instancia representativa de cada familia. Los frentes mostrados son los globales, obtenidos combinando las soluciones no dominadas halladas en las 30 ejecuciones independientes de cada AE.

Ambos AE mejoran las soluciones alcanzadas por los algoritmos ávidos. Sin embargo, se observa que el algoritmo $NSGA-II$ alcanza mejores soluciones, particularmente en la instancia grande. Además, la cantidad de puntos no dominados encontrados por $NSGA-II$ es significativamente mayor y los mismos se distribuyen homogéneamente a lo largo del frente, logrando una mejor diversidad que la alcanzada por $p\mu MOEA/D$.

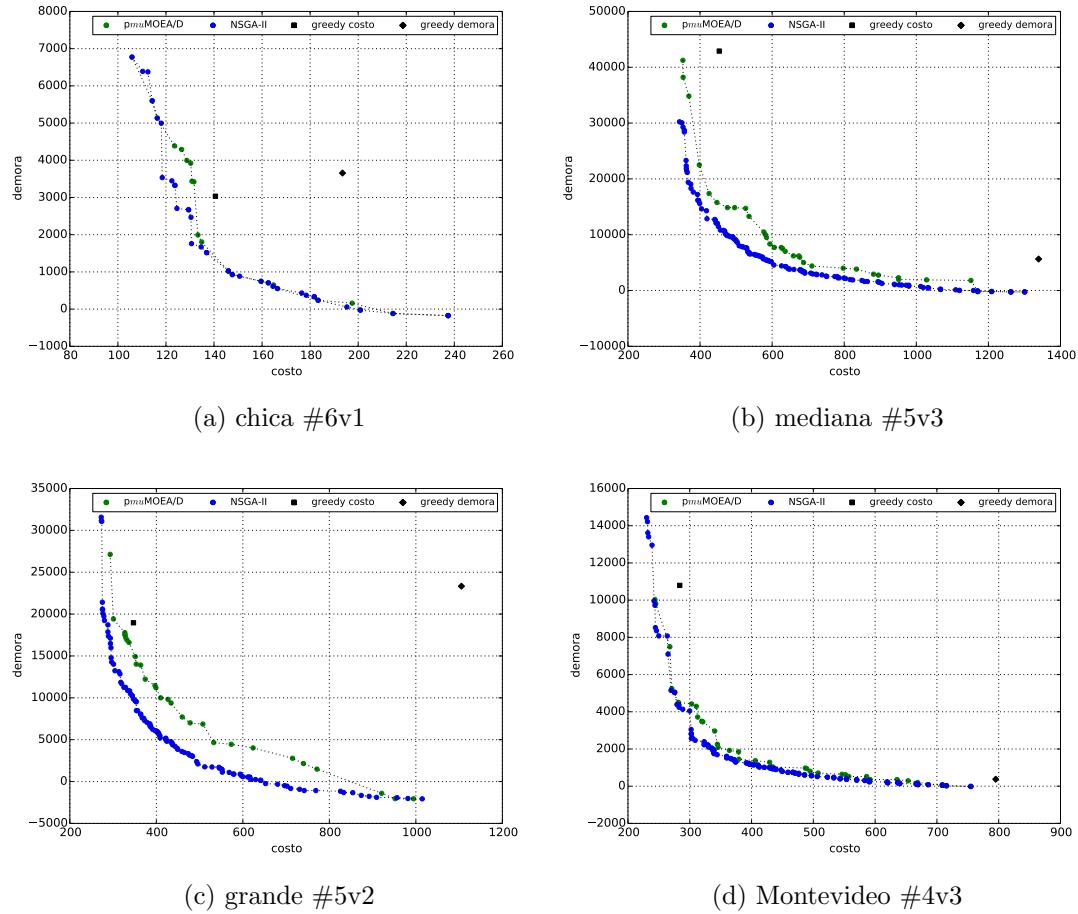


Figura 6.17: $p\mu$ MOEA/D vs. NSGA-II—frentes de Pareto.

Capítulo 7

Planificador de viajes compartidos en línea

Para la resolución de instancias reales del problema por parte de usuarios, se desarrolló un sistema conformado por una aplicación web y una aplicación móvil, al cual llamamos *planificador de viajes compartidos en línea*. La Sección 7.1 describe el funcionamiento del planificador de viajes compartidos, la Sección 7.2 describe la arquitectura y las tecnologías utilizadas para su implementación y finalmente las Secciones 7.3 y 7.4 describen los detalles de implementación de la aplicación web y móvil, respectivamente.

7.1. Descripción general del planificador

El planificador permite al usuario ingresar el origen común a todos los pasajeros y los destinos de cada uno de ellos y luego se encarga de ejecutar el algoritmo evolutivo que realiza la asignación de pasajeros a taxis buscando optimizar el costo total de los viajes.

El usuario ingresa los diferentes puntos a través de un mapa desplegado en la pantalla. En el ejemplo de la Figura 7.1 se aprecia un origen ingresado (indicado con un marcador de color verde) y los destinos (indicados con marcadores de color rojo). El usuario debe seleccionar la tarifa para calcular los costos entre dos opciones: tarifa diurna y nocturna. En la implementación realizada se utilizan los valores actuales para la ciudad de Montevideo, aunque se propone como trabajo futuro incorporar los datos del servicio Taxi Fare Finder para proveer soporte a mapas definidos en otras ciudades.

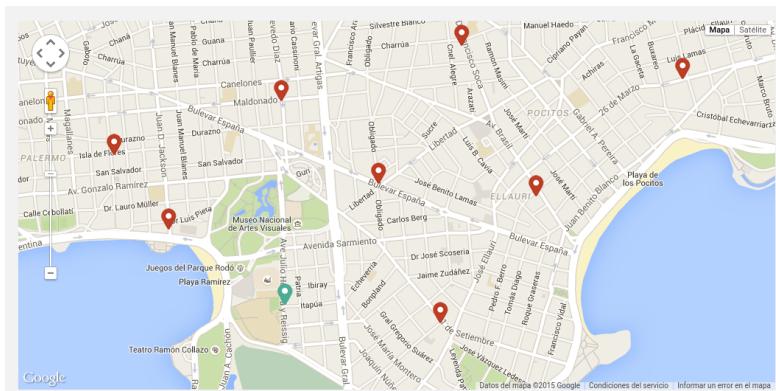


Figura 7.1: Ingreso de origen y destinos en el planificador de viajes compartidos en línea.

La salida desplegada al usuario indica la cantidad adecuada de taxis para satisfacer los pedidos, junto a las rutas asignadas y la tarifa estimada para cada taxi. La Figura 7.2 muestra la salida presentada al usuario con la planificación calculada para el caso de ejemplo de la Figura 7.1. Se presenta una lista de taxis con los pasajeros asignados a cada uno de los taxis en el orden en que serán trasladados. Adicionalmente, se despliega el mapa con los puntos ingresados y se proporciona al usuario la opción de ver de forma gráfica el recorrido de cada taxi en el mapa (al hacer clic sobre el ícono correspondiente).

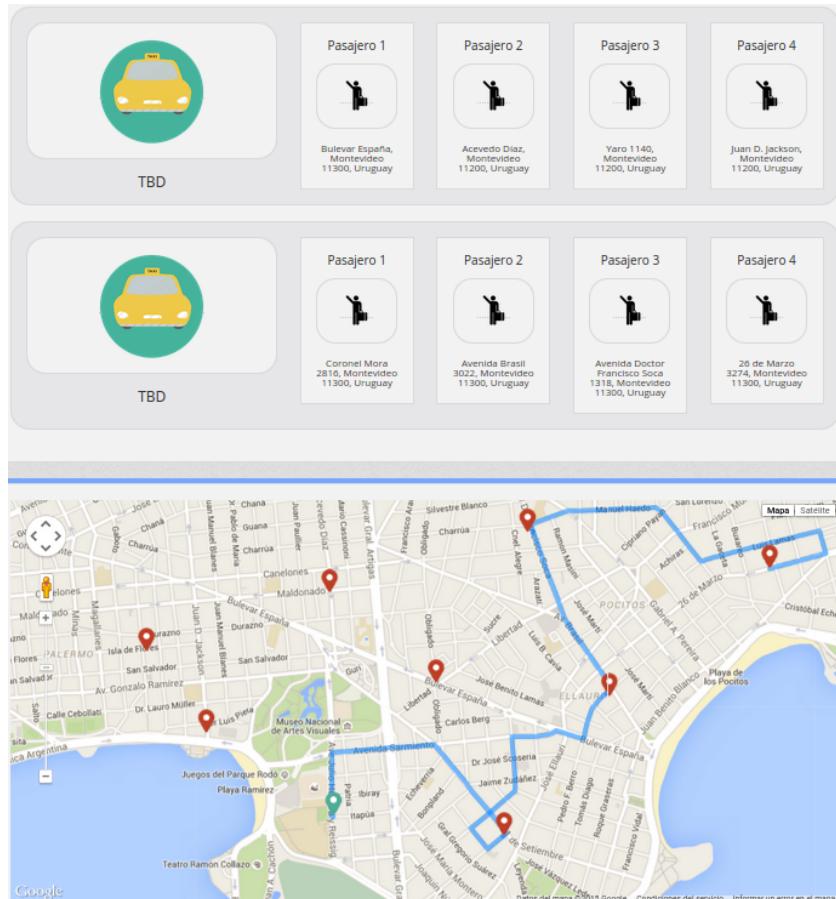


Figura 7.2: Visualización de resultados en el planificador de viajes compartidos en línea.

7.2. Arquitectura del sistema y tecnologías utilizadas

De acuerdo a las funcionalidades que se buscan brindar en el planificador de viajes compartidos en línea, los siguientes requisitos deben ser tenidos en cuenta al momento de diseñar la arquitectura del sistema:

- *soporte para la ejecución de programas con alto consumo de recursos computacionales*: además de atender las solicitudes de los usuarios, el planificador deberá encargarse de ejecutar el algoritmo evolutivo para resolver la instancia del problema que cada usuario ingrese al sistema. Considerando que varios usuarios pueden utilizar el servicio concurrentemente, la capacidad de procesamiento debe ser adecuada para poder ofrecer un servicio adecuado a todos los usuarios.

- *conexiones con servicios externos*: se utilizarán servicios de terceros para determinadas funciones del sitio web (e.g., *Google Maps* como proveedor de mapas, *TaxiFareFinder* para el cálculo de tarifas). Por esta razón, se debe tener en cuenta que la aplicación realizará grandes cantidades de consultas a dichos servicios, generando un volumen de tráfico a considerar en el diseño.
- *gran volumen de operaciones de lectura y escritura en archivos*: se debe tener en cuenta que el algoritmo evolutivo tendrá como entrada una serie de archivos con la información referida a la instancia que se desea resolver y devolverá un archivo de salida con la solución encontrada, para cada una de las instancias ingresadas por los usuarios.
- *conexiones con aplicaciones móviles*: se deben proveer servicios accesibles para las aplicaciones móviles, que permitan acceder a las mismas funcionalidades que se ofrecen a través de la web.

En base a los puntos mencionados, se desarrolló un sistema con una arquitectura simple pero escalable. Los detalles de la arquitectura se presentan en la Figura 7.3 y se describen a continuación.

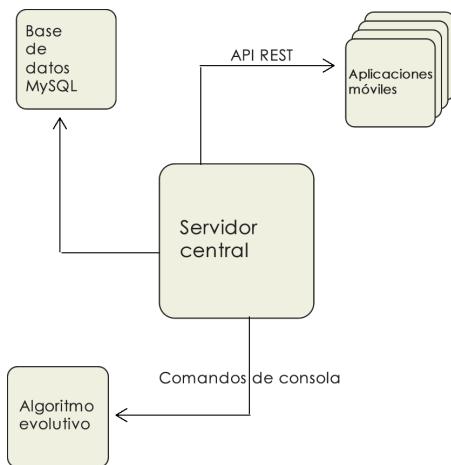


Figura 7.3: Arquitectura del planificador de viajes compartidos en línea.

Servidor central

El servidor central es el componente principal del planificador de viajes compartidos, siendo el encargado de recibir las solicitudes de los usuarios y retornar las vistas que se desplegarán en el navegador web del usuario. Por otra parte, brinda una *Interfaz de Programación de Aplicaciones (API)* para la comunicación y posterior ejecución de instancias en aplicaciones móviles.

El servidor central está implementado en Ruby on Rails (*RoR*) [53]. *RoR* es un framework de aplicaciones web de código libre escrito en el lenguaje de programación Ruby [23], siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC). *RoR* busca una combinación apropiada entre la simplicidad del código escrito y la posibilidad de desarrollar aplicaciones reales escalables en pocas líneas de código. *Rails* se distribuye a través de *RubyGems*, que es el formato oficial de paquete y canal de distribución de bibliotecas y aplicaciones Ruby.

La API para la interacción con las aplicaciones móviles está implementada con la arquitectura conocida como Transferencia de Estado Representacional (*Representational State Transfer, REST*). *REST* se utiliza para describir cualquier interfaz web simple basada en *XML* y *HTTP*, sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes, tal como el protocolo de servicios web *SOAP*.

JavaScript es un lenguaje de programación interpretado, implementación del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Javascript se utiliza principalmente en su forma del lado del cliente (*client-side*), implementado como parte de un navegador web, permitiendo mejoras en la interfaz de usuario a través de páginas web dinámicas. Una vez que las páginas son servidas al usuario, se utiliza JavaScript para manejar la interacción con las vistas, agregando dinamismo al sitio web.

Base de datos

Con el fin de optimizar las consultas realizadas al sistema y persistirlas en el tiempo, se hace uso de una base de datos relacional. En la implementación actual, la base de datos se encuentra ubicada conjuntamente con el servidor central, aunque podría ser asignada a un servidor independiente en el futuro, aplicando una arquitectura cliente-servidor de tres niveles con el fin de mejorar el rendimiento del sistema.

Para la implementación de la base de datos se eligió la tecnología de código libre MySQL. MySQL es un sistema de administración de bases de datos relacionales escrito en los lenguajes *C* y *C++*, que destaca por su fácil adaptación a diferentes entornos de desarrollo, permitiendo su interacción con diversos lenguajes de programación.

Algoritmo evolutivo

La ejecución del algoritmo evolutivo se encuentra aislada en un componente de software, el cual se comunica con el servidor central utilizando comandos de consola. A través de la lectura y escritura de archivos, el algoritmo evolutivo recibe las variables de entrada específicas a la instancia del problema a resolver (e.g., la matriz de costos, la cantidad de pasajeros, el costo de la bandera). Al finalizar su ejecución, el AE almacena en la base de datos la solución calculada, con el objetivo de tenerla disponible en caso que el usuario desee acceder nuevamente al resultado de la consulta.

Aplicaciones móviles

Este componente permite utilizar el planificador de viajes compartidos desde teléfonos móviles. Una primera versión de la aplicación móvil fue desarrollada para teléfonos inteligentes con sistema operativo *iOS* y se plantea como trabajo futuro desarrollar versiones para otras plataformas.

7.3. Aplicación web

La aplicación web permite resolver instancias reales del problema de viajes compartidos en taxis por medio de una página web. La misma se encuentra disponible públicamente en la dirección www.mepaseaste.uy. La interfaz de usuario de la aplicación web se presenta en la Figura 7.4.

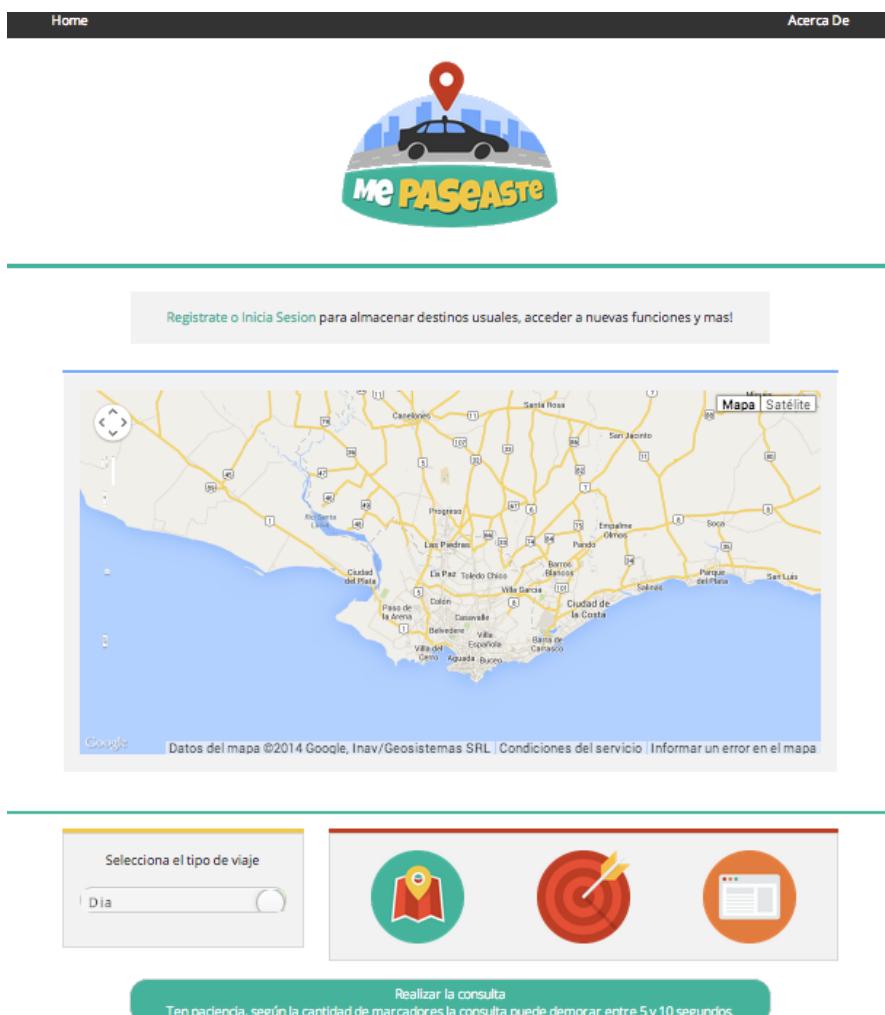


Figura 7.4: Interfaz de la aplicación web del planificador de viajes compartidos en línea.

Modelos

En las aplicaciones web orientadas a objetos, el *modelo* consiste en las clases que representan a los objetos del sistema, que generalmente coinciden con las tablas de la base de datos. Dos modelos importantes implementados en el servidor central son *consulta* y *ubicación*. *Consulta* representa las consultas del usuario, con una serie de atributos particulares, por ejemplo si la consulta es pública o no, el número de destinos, entre otros. *Ubicación* modela las ubicaciones ingresadas por el usuario, identificadas por su latitud y longitud. Además, se implementaron modelos para los *usuarios* y para los *roles*, de forma de restringir el acceso a los diferentes servicios de la aplicación web.

Vistas

En las aplicaciones *MVC* existe un componente fundamental llamado *vista*, que representa los datos obtenidos por el *controlador* en el navegador del usuario. Usualmente, en las aplicaciones web la *vista* consiste en una cantidad mínima de código escrito en el lenguaje HTML. En el caso de la aplicación web desarrollada para la planificación de taxis, existen vistas para ingresar los puntos y vistas para visualizar los resultados.

Asimismo, existen vistas orientadas a mejorar la experiencia de usuario, incluyendo un perfil de usuario y páginas de inicio de sesión, entre otros.

Controladores

En *MVC*, las clases del *controlador* responden a la interacción del usuario e invocan a la lógica de la aplicación. Esta lógica es la encargada de manipular los datos de las clases del *modelo* y mostrar los resultados usando las *vistas*. En las aplicaciones web basadas en *MVC*, los métodos del *controlador* son invocados por el usuario a través del navegador web. En la aplicación web implementada para el planificador de viajes compartidos, existen controladores para los resultados calculados por el AE y para el manejo de los puntos ingresados en el mapa.

Procesamiento en el navegador

Para poder realizar la planificación utilizando el algoritmo evolutivo, es necesario calcular las distancias entre todos los puntos ingresados por el usuario a través del mapa. El cálculo de estas distancias se realiza utilizando la API de Google Maps. Dado que las consultas a la API implican un tiempo de respuesta considerable, se tomaron ciertas medidas para solapar el cómputo de datos, de forma de mejorar el tiempo de respuesta y brindar una mejor experiencia de usuario. Cada vez que el usuario inserta un punto en el mapa, se ejecuta una serie de métodos asíncronos en el navegador utilizando la tecnología JavaScript, de forma de obtener la distancia del punto ingresado a todos los puntos ingresados anteriormente. De esta forma, las consultas a la API de Google Maps se realizan a medida que el usuario ingresa los puntos en el mapa, lo que mejora notablemente la experiencia de usuario, respecto a realizar el total de las consultas una vez que todos los puntos fueron ingresados.

7.4. Aplicación móvil

La aplicación móvil permite resolver instancias del problema de viajes compartidos en taxi a través de dispositivos móviles. Se describe a continuación un prototipo implementado utilizando una plataforma que soporta distintos sistemas operativos y luego una aplicación desarrollada de forma nativa para dispositivos con sistema operativo iOS.

7.4.1. Prototipo basado en PhoneGap

Al comenzar el desarrollo se implementó un prototipo basado en la tecnología *PhoneGap* [60]. PhoneGap es una plataforma que permite a los programadores desarrollar aplicaciones para dispositivos móviles utilizando herramientas genéricas tales como JavaScript, HTML5 y CSS3. Dado que estas tecnologías fueron utilizadas para el desarrollo de la aplicación web, el pasaje a una aplicación móvil es relativamente sencillo. Las aplicaciones desarrolladas utilizando esta plataforma son llamadas aplicaciones híbridas; por un lado, no son consideradas aplicaciones nativas al dispositivo, ya que la generación de las vistas se realiza mediante un proceso análogo al realizado por los navegadores web, sin utilizar interfaces gráficas específicas para cada sistema. Por otro lado, tampoco son aplicaciones web, dado que son empaquetadas y luego desplegadas en el dispositivo, teniendo acceso a las funciones provistas por el sistema operativo móvil.

El sistema implementado requiere de continuas interacciones con la API de Google Maps, así como el despliegue de un mapa en la pantalla y la consulta al *GPS* del dispositivo para determinar la ubicación del usuario. Al evaluar el prototipo desarrollado se pudo verificar que una aplicación híbrida brinda una mala experiencia de usuario debido a estos requisitos. Los pobres resultados de usabilidad y desempeño alcanzados por el prototipo desarrollado llevaron a considerar al desarrollo nativo como la mejor alternativa para la aplicación móvil. Esta decisión acota el número de dispositivos soportados por la aplicación, pero brinda una mejor experiencia de uso y un desempeño superior.

7.4.2. Desarrollo Nativo

A diferencia del prototipo, la aplicación nativa fue implementada y evaluada en un conjunto específico de dispositivos. La interfaz de usuario de la aplicación móvil desarrollada se presenta en la Figura 7.5. La aplicación móvil solamente actúa como intermediaria entre el usuario y el servidor central, dado que el cálculo de la asignación de pasajeros y recorridos de los taxis se realiza en el servidor central. La aplicación móvil consume servicios de la API tanto para enviar los puntos ingresados como para obtener el resultado de la ejecución del AE. Se optó por desarrollar una versión de la aplicación para teléfonos inteligentes con el sistema operativo iOS de Apple. Entre los modelos de dispositivos soportados se encuentran los iPhones 3G, 3GS, 4, 4S, 5, 5S, 6 y 6 Plus.

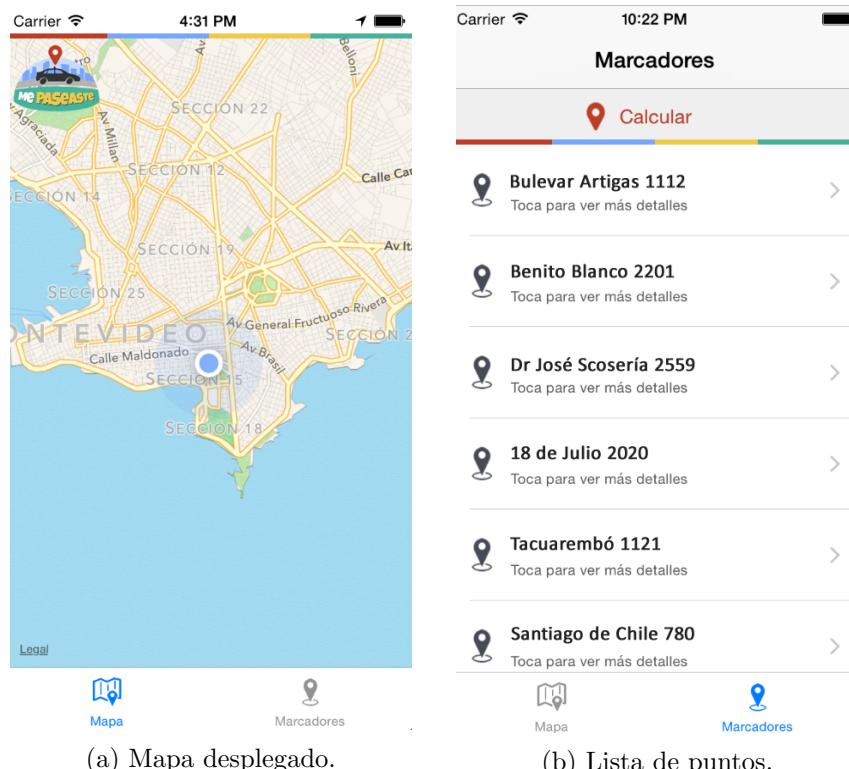


Figura 7.5: Interfaz de la aplicación móvil del planificador de viajes compartidos en línea.

Capítulo 8

Conclusiones y trabajo futuro

Esta sección presenta las conclusiones del trabajo realizado en el marco del proyecto y las principales líneas de trabajo futuro.

8.1. Conclusiones

Este proyecto de grado presentó la resolución del problema de viajes compartidos en taxis utilizando algoritmos evolutivos. Se presentaron dos variantes del problema: una variante monoobjetivo, donde se desea minimizar el costo total de un grupo de pasajeros; y una variante multiobjetivo, donde se considera como objetivo complementario minimizar la demora percibida por cada pasajero al compartir su viaje. Se estudió el problema y se relevaron los principales trabajos de la literatura relacionada, incluyendo trabajos que resuelven problemas similares (*CPP*, *DARP*, *TPP*). Un total de cuatro algoritmos evolutivos fueron implementados, dos para cada una de las variantes del problema estudiadas. Para la variante monoobjetivo del problema, se desarrolló un algoritmo secuencial (*seqEA*) y un micro algoritmo evolutivo paralelo (*pμEA*). Para la variante multiobjetivo del problema, se implementó un micro algoritmo evolutivo paralelo con un enfoque de descomposición de dominio (*pμMOEA/D*) y un algoritmo evolutivo con un enfoque multiobjetivo explícito (*NSGA-II*).

El análisis experimental fue realizado sobre un conjunto de instancias realistas del problema, utilizando información obtenida de la literatura relacionada. Las instancias de prueba fueron generadas a partir de datos reales, recolectados mediante dispositivos GPS instalados en más de 33.000 taxis en la ciudad de Beijing, China durante un período de 87 días. Adicionalmente, se desarrollaron instancias del problema ubicadas en la ciudad de Montevideo, tomando en cuenta puntos de interés de la ciudad. Para la evaluación experimental de los algoritmos que resuelven la variante monoobjetivo del problema se generó y posteriormente se utilizó un conjunto de **22** instancias de prueba, mientras que para los algoritmos que resuelven la variante multiobjetivo se generó y posteriormente se utilizó un conjunto de **88** instancias del problema.

Los algoritmos evolutivos desarrollados fueron comparados frente a algoritmos ávidos implementados en base a ideas presentadas en trabajos de la literatura relacionada, los cuales emulan una estrategia intuitiva y se aproximan a lo que un grupo de usuarios humanos puede idear de manera simple para resolver el problema. En el caso de los algoritmos evolutivos que resuelven la variante monoobjetivo del problema, se compararon dos métodos de inicialización de la población: una inicialización completamente aleatoria

y una inicialización que parte de la solución encontrada por el algoritmo ávido. Los resultados experimentales sugieren que la inicialización a partir de la solución del algoritmo ávido permite alcanzar mejores resultados que la inicialización aleatoria. El algoritmo *seqEA* fue capaz de mejorar los resultados del algoritmo ávido hasta en un **35.9 %**, mientras que el algoritmo *pμEA* fue capaz de alcanzar mejoras de hasta un **41.0 %**. Los resultados experimentales muestran que *pμEA* fue capaz de encontrar mejoras estadísticamente significativas sobre los resultados de *seqEA* en **17** de las **22** instancias de prueba. Adicionalmente, estas mejoras se logran en tiempos de ejecución significativamente menores a los registrados por *seqEA*. Ambos AE alcanzan mejoras ampliamente mayores a las reportadas por los algoritmos propuestos en la literatura relacionada.

Los algoritmos implementados para resolver la variante multiobjetivo del problema utilizan una inicialización de la población a partir de las soluciones encontradas por los algoritmos ávidos diseñados para cada objetivo. Los resultados experimentales muestran que el algoritmo *pμMOEA/D* logró mejorar hasta en un **101.2 %** la demora alcanzada por el algoritmo ávido de costo, manteniendo el mismo costo; y logró mejorar hasta en un **72.8 %** el costo alcanzado por el algoritmo ávido de demora, manteniendo la misma demora. Por su parte, el algoritmo *NSGA-II* logró mejorar hasta en un **105.2 %** la demora alcanzada por el algoritmo ávido de costo, manteniendo el mismo costo; y logró mejorar hasta en un **75.1 %** el costo alcanzado por el algoritmo ávido de demora, manteniendo la misma demora. Los resultados experimentales permiten afirmar con significancia estadística que el algoritmo *NSGA-II* alcanzó mejores resultados que el *pμMOEA/D* en el total de instancias de prueba estudiadas. Sin embargo, los tiempos de ejecución de *pμMOEA/D* son significativamente menores que los de *NSGA-II*, por lo que es necesario establecer un compromiso entre la calidad de las soluciones halladas y el tiempo de respuesta.

Finalmente, se diseñó e implementó el planificador de viajes en línea, que consiste en una aplicación web y una aplicación móvil que permiten a los usuarios resolver instancias reales del problema de viajes compartidos en taxis. Los usuarios pueden seleccionar sus destinos a través de un mapa; la aplicación realiza la planificación utilizando los algoritmos evolutivos implementados y le ofrece a los usuarios información acerca de la cantidad de taxis necesarios, las tarifas y los recorridos de cada taxi. La aplicación se encuentra disponible públicamente en el sitio www.mepaseaste.uy.

En el marco de este proyecto se publicaron **cuatro artículos científicos** en distintos congresos internacionales, cada uno de ellos presentando uno de los algoritmos evolutivos implementados. En el Apéndice B se pueden consultar los siguientes artículos publicados:

1. “*Online taxi sharing optimization using evolutionary algorithms*” presentado por Gabriel Fagúndez en el Simposio Latinoamericano de Investigación de Operaciones e Inteligencia Artificial de la XL Conferencia Latinoamericana en Informática (CLEI), celebrada del 15 al 19 de setiembre de 2014 en la ciudad de Montevideo, Uruguay. El artículo describe el algoritmo secuencial (*seqEA*) que resuelve la formulación monoobjetivo del problema de viajes compartidos en taxis.
2. “*A parallel micro evolutionary algorithm for taxi sharing optimization*” presentado por Renzo Massobrio en el VIII ALIO/EURO Workshop on Applied Combinatorial Optimization, celebrado entre el 8 y 10 de diciembre de 2014 en la ciudad de Montevideo, Uruguay. El artículo describe el algoritmo paralelo con micro poblaciones distribuidas (*pμEA*) que resuelve la formulación monoobjetivo del problema de viajes compartidos en taxis.

3. “*Planificación multiobjetivo de viajes compartidos en taxis utilizando un micro algoritmo evolutivo paralelo*” presentado por Sergio Nesmachnow en el X Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB) celebrado en las ciudades de Mérida y Almendralejo del 4 al 6 de febrero de 2015. El artículo describe el algoritmo paralelo por descomposición de dominio ($p\mu MOEA/D$) que resuelve la variante multiobjetivo del problema de viajes compartidos en taxis.
4. “*Multiobjective taxi sharing optimization using the NSGA-II evolutionary algorithm*” presentado por Sergio Nesmachnow en la sesión especial “Metaheuristics for Smart Cities” en la 11th edition of the Metaheuristics International Conference (MIC) celebrada los días 7 a 10 de junio de 2015 en la ciudad de Agadir, Marruecos. El artículo describe el algoritmo evolutivo con enfoque multiobjetivo explícito (NSGA-II) que resuelve la variante multiobjetivo del problema de viajes compartidos en taxis.

Adicionalmente, el trabajo realizado fue presentado en la edición 2014 de Ingeniería deMuestra [32], el evento anual que organiza la Facultad de Ingeniería de la Universidad de la República en conjunto con la Fundación Julio Ricaldoni, donde se presentan proyectos, investigaciones, y emprendimientos de estudiantes y docentes, con el fin de exhibir las actividades que se realizan en la Facultad de Ingeniería. En dicha muestra, el proyecto obtuvo el **primer premio** del jurado en la categoría “Proyectos de fin de carrera de Ingeniería en Computación”. En dicho evento, el grupo de trabajo realizó una entrevista presentando el proyecto, la cual fue publicada en el sitio web *Cromo* del diario *El Observador* [10].

8.2. Trabajo futuro

A continuación, se detallan las principales líneas de trabajo futuro que surgen a partir del trabajo realizado en el marco de este proyecto.

Diversos algoritmos fueron implementados para resolver el problema de viajes compartidos en taxis, mejorando progresivamente tanto la calidad de las soluciones alcanzadas como el desempeño computacional. En particular, para el problema de viajes compartidos en su variante multiobjetivo se implementaron dos algoritmos evolutivos: $p\mu MOEA/D$ y NSGA-II. Si bien NSGA-II fue capaz de alcanzar mejoras significativas respecto a los resultados alcanzados por $p\mu MOEA/D$, estas mejoras fueron alcanzadas en tiempos de ejecución mayores a los registrados por $p\mu MOEA/D$ que, gracias al modelo de subpoblaciones distribuidas, registró tiempos de ejecución menores. Por esta razón, se plantea como una de las líneas de trabajo futuro la implementación de una variante del algoritmo NSGA-II que utilice el modelo de subpoblaciones distribuidas, de forma de obtener buenos resultados en calidad de soluciones y tiempos de ejecución reducidos.

El problema abordado no considera información del tráfico al momento de elegir las rutas que sigue un determinado taxi: la ruta para todo par de puntos es la misma, sin importar la hora del día o las condiciones del tránsito. La incorporación de datos realistas del tráfico permitiría considerar rutas alternativas para los taxis, que incidan en el costo del viaje y, en el caso de la variante multiobjetivo del problema, en la demora percibida por los usuarios. Diversas reuniones fueron realizadas con el *Departamento de Tránsito y Transporte* y con la *Unidad Ejecutiva del Plan de Movilidad Urbana* (ambos

dependientes de la *Intendencia de Montevideo*) con el fin de conseguir acceso a datos del tráfico en la ciudad de Montevideo. Los esfuerzos realizados permitieron obtener acceso únicamente a datos parciales, correspondientes a un mes de recorrido de las principales líneas de ómnibus de la ciudad. No se logró obtener acceso a datos de conteos de vehículos que permitan realizar un mapa del tránsito de la capital. Tampoco fue posible conseguir información sobre los recorridos de los taxis, a pesar de que la misma se encuentra en poder de la Intendencia de Montevideo. De igual forma, sería útil contar con datos de la disponibilidad de taxis y de sus capacidades para la ciudad de Montevideo. Dicha información está en poder de las empresas privadas de taxímetros, por lo que el acceso a dichos datos es aún más complejo que en el caso de los datos del tránsito. El desarrollo de una aplicación orientada al usuario con mayores prestaciones puede ser un medio para que las empresas de taxis se interesen en el proyecto y ofrezcan datos acerca de sus flotillas de vehículos.

En cuanto a la aplicación para el usuario final, es necesario dedicar esfuerzos en pos de mejorar la experiencia de los usuarios. Con este motivo, se realizó una reunión con la *Gerencia de Innovación* de la *Administración Nacional de Telecomunicaciones (ANTEL)*, la cual brinda apoyo a proyectos de investigación e innovación. En dicha reunión, se presentó el trabajo realizado y se discutieron las posibles mejoras a realizar, con el fin de conseguir el apoyo necesario para continuar con el proyecto. Al momento de escribir este informe no se obtuvo una respuesta al planteo realizado. La versión actual del planificador de viajes en línea requiere que un único usuario seleccione manualmente los destinos de cada uno de los pasajeros. Una versión donde cada pasajero sea el encargado de indicar su destino, que permita almacenar los destinos más frecuentes y que utilice la ubicación del usuario para establecer automáticamente el origen, ofrecería una experiencia de usuario más positiva. De igual forma, se plantea como trabajo futuro el desarrollo de versiones de la aplicación móvil que soporten dispositivos con sistemas *Android* y *Windows Phone*, de forma de abarcar un mayor conjunto de usuarios de teléfonos inteligentes. Adicionalmente, resulta interesante el estudio de una arquitectura que permita integrar los algoritmos paralelos diseñados con el planificador de viajes en línea, de forma de reducir los tiempos de respuesta y mejorar la experiencia de usuario. Actualmente, la aplicación web utiliza la versión monoobjetivo secuencial (*seqEA*), pero el análisis experimental mostró que el AE paralelo con subpoblaciones distribuidas ($p\mu EA$) alcanza mejores resultados. De igual forma, se plantea como trabajo a futuro desarrollar una versión del planificador de viajes en línea que permita resolver el problema en su variante multiobjetivo, utilizando los algoritmos $p\mu MOEA/D$ o *NSGA-II* desarrollados.

Por último, es de interés estudiar la aplicabilidad de los algoritmos desarrollados en este proyecto a escenarios distintos al de los viajes compartidos en taxis (e.g., viajes compartidos en vehículos particulares, transporte de escolares y de personas con movilidad reducida). Interesa a su vez estudiar variantes del problema, ya sea el problema inverso (de muchos orígenes hacia un único destino) como la versión genérica de muchos orígenes a muchos destinos. Creemos que muchos de los problemas atacados en este proyecto pueden facilitar el desarrollo de algoritmos para estas u otras variantes del problema.

Bibliografía

- [1] E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, C. León, G. Luque, J. Petit, C. Rodríguez, A. Rojas, y F. Xhafa. Efficient parallel LAN/WAN algorithms for optimization. The Mallba project. *Parallel Computing*, 32(5–6):415 – 440, 2006.
- [2] E. Alba, G. Luque, y S. Nesmachnow. Parallel metaheuristics: Recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48, 2013.
- [3] T. Back, D. B. Fogel, y Z. Michalewicz, editores. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK, primera edición, 1997.
- [4] R. Baldacci, V. Maniezzo, y A. Mingozzi. An exact method for the car pooling problem based on Lagrangean column generation. *Operations Research*, 52(3):422–439, 2004.
- [5] M. Bergara. Resolución 476/014: fijación de tarifas máximas para los taxímetros. *Centro de Información Oficial*, 2014. <http://www.impo.com.uy/bases/resoluciones/476-2014>, Online; accedido en Julio 2015.
- [6] C. Bialik. How to Split a Shared Cab Ride? Very Carefully, Say Economists. *The Wall Street Journal*, 2005. <http://www.wsj.com/articles/SB113279169439805647>, Online; accedido en Julio 2015.
- [7] BlaBlaCar: conectamos conductores con pasajeros para compartir coche. <https://www.blablacar.es/>. Online; accedido en Julio 2015.
- [8] Carma Carpooling: your commuting rideshare companion. <https://carmacarpool.com/>. Online; accedido en Julio 2015.
- [9] Carpling taxi pooling. <https://www.carpling.com/info/taxi>. Online; accedido en Julio 2015.
- [10] M. Castineiras. Creatividad con forma de ingeniería. *Cromo*, 2014. <http://www.cromo.com.uy/creatividad-forma-ingenieria-n583610>, Online; accedido en Julio 2015.
- [11] R. Chevrier, A. Liefoghe, L. Jourdan, y C. Dhaenens. Solving a dial-a-ride problem with a hybrid evolutionary multi-objective approach: Application to demand responsive transport. *Applied Soft Computing*, 12(4):1247 – 1258, 2012.
- [12] Cluster FING. Plataforma de computación de alto desempeño de Facultad de Ingeniería, Universidad de la República. <http://www.fing.edu.uy/cluster/>, 2015. Online; accedido en Julio 2015.
- [13] C. Coello y G. Pulido. A micro-genetic algorithm for multiobjective optimization. En *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization*, páginas 126–140, London, UK, 2001.
- [14] C. Coello, D. Van Veldhuizen, y G. Lamont. *Evolutionary algorithms for solving multi-objective problems*. Kluwer, New York, 2002.
- [15] J.-F. Cordeau y G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579 – 594, 2003.

- [16] M. Deakin y H. A. Waer. From intelligent to smart cities. *Intelligent Buildings International*, 3(3):133–139, 2011.
- [17] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. J. Wiley & Sons, Chichester, 2001.
- [18] K. Deb, A. Pratap, S. Agarwal, y T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
- [19] S. DeLoach y T. Tiemann. Not driving alone? American commuting in the twenty-first century. *Transportation*, 39(3):521–537, 2012.
- [20] ECJ 23: a Java-based Evolutionary Computation Research System. <https://cs.gmu.edu/~eclab/projects/ecj/>. Online; accedido en Julio 2015.
- [21] El Observador. En Maldonado se paga una ficha de taxi tres veces más cara que en Montevideo. 2013. <http://www.elobservador.com.uy/en-maldonado-se-paga-una-ficha-taxi-tres-veces-mas-cara-que-montevideo-n263207>, Online; accedido en Julio 2015.
- [22] N. Fellows y D. Pitfield. An economic and operational evaluation of urban car-sharing. *Transportation Research Part D: Transport and Environment*, 5(1):1 – 10, 2000.
- [23] D. Flanagan y Y. Matsumoto. *The Ruby Programming Language*. O'Reilly, primera edición, 2008.
- [24] C. A. Floudas y P. M. Pardalos, editores. *Encyclopedia of Optimization*. Springer, segunda edición, 2009.
- [25] C. Fuhs y J. Obenberger. Development of high-occupancy vehicle facilities: Review of national trends. *Transportation Research Record*, 1781:1–9, 2002.
- [26] F. Glover y G. Kochenberger. *Handbook of Metaheuristics*. International series in operations research & management science. Kluwer Academic Publishers, 2003.
- [27] Google Maps API. <https://developers.google.com/maps/>. Online; accedido en Julio 2015.
- [28] I. B.-A. Hartman, D. Keren, A. A. Dbai, E. Cohen, L. Knapen, A.-U.-H. Yasar, y D. Janssens. Theory and practice in large carpooling problems. *Procedia Computer Science*, 32:339–347, 2014.
- [29] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA, 1992.
- [30] J. E. Hopcroft y R. M. Karp. A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. En *12th Annual Symposium on Switching and Automata Theory*, páginas 122–125, 1971.
- [31] H. Hosni, N. Farhat, R. Nimer, N. Alawieh, C. Masri, M. Saroufim, H. Artail, y J. Naoum-Sawaya. An optimization-based approach for passenger to shared taxi allocation. En *20th International Conference on Software, Telecommunications and Computer Networks*, páginas 1–7, 2012.
- [32] Proyectos premiados–Ingeniería deMuestra–Facultad de Ingeniería. <https://www.fing.edu.uy/ingenieriademuestra/proyectos-premiados>. Online; accedido en Julio 2015.
- [33] J.-J. Jaw, A. R. Odoni, H. N. Psaraftis, y N. H. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243 – 257, 1986.
- [34] L. Knapen, I. B.-A. Hartman, D. Keren, A.-U.-H. Yasar, S. Cho, T. Bellemans, D. Janssens, y G. Wets. Scalability issues in optimal assignment for carpooling. *Journal of Computer and System Sciences*, 81(3):568 – 584, 2015.

- [35] W. H. Kruskal y W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.
- [36] H. W. Kuhn. Variants of the Hungarian method for assignment problems. *Naval Research Logistics Quarterly*, 3(4):253–258, 1956.
- [37] A. N. Letchford, R. W. Eglese, y J. Lysgaard. Multistars, partial multistars and the capacitated vehicle routing problem. *Mathematical Programming*, 94(1):21–40, 2002.
- [38] Y. Lin, W. Li, F. Qiu, y H. Xu. Research on optimization of vehicle routing problem for ride-sharing taxi. *Procedia - Social and Behavioral Sciences*, 43:494 – 502, 2012.
- [39] D. Lois. Una batalla diaria por las calles montevideanas. *El Observador*, 2013. <http://www.elobservador.com.uy/una-batalla-diaria-las-calles-montevideanas-n256829>, Online; accedido en Julio 2015.
- [40] Introducing Lyft Line, your daily ride. <https://www.lyft.com/line>. Online; accedido en Julio 2015.
- [41] S. Ma, Y. Zheng, y O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. En *IEEE 29th International Conference on Data Engineering*, páginas 410–421, 2013.
- [42] O. Madsen, H. Ravn, y J. Rygaard. A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60(1):193–208, 1995.
- [43] MPI Forum. MPI: A Message-Passing Interface Standard. Technical report, University of Tennessee Knoxville, TN, USA, 1994.
- [44] MPICH: high-performance portable MPI. <https://www.mpich.org>. Online; accedido en Julio 2015.
- [45] S. Nesmachnow. Computación científica de alto desempeño en la Facultad de Ingeniería, Universidad de la República. *Revista de la Asociación de Ingenieros del Uruguay*, 61:12–15, 2010.
- [46] S. Nesmachnow. An overview of metaheuristics: accurate and efficient methods for optimisation. *International Journal of Metaheuristics*, 3(4):320, 2014.
- [47] S. Nesmachnow. Using metaheuristics as soft computing techniques for efficient optimization. En *Encyclopedia of Information Science and Technology*, páginas 7390–7399. IGI Global, tercera edición, 2015.
- [48] S. Nesmachnow y S. Iturriaga. Multiobjective grid scheduling using a domain decomposition based parallel micro evolutionary algorithm. *International Journal of Grid and Utility Computing*, 4:70–84, 2013.
- [49] C. M. Schweik, M. T. Fernandez, M. P. Hamel, P. Kashwan, Q. Lewis, y A. Stepanov. Reflections of an online geographic information systems course based on open source software. *Social Science Computer Review*, 27(1):118–129, 2008.
- [50] S. S. Shapiro y M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3-4):591–611, 1965.
- [51] Sharetransport: Car pool, taxi pool, and bus pool/charter. <http://www.sharetransport.sg/>. Online; accedido en Julio 2015.
- [52] C.-C. Tao y C.-Y. Chen. Heuristic algorithms for the dynamic taxipooling problem based on intelligent transportation system technologies. En *4th International Conference on Fuzzy Systems and Knowledge Discovery*, volumen 3, páginas 590–595, 2007.
- [53] B. Tate y C. Hibbs. *Ruby on Rails: Up and Running*. O'Reilly Media, Inc., 2006.

- [54] TaxiFareFinder API (beta): Taxi API for developers. <http://www.taxifarefinder.com/api.php>. Online; accedido en Julio 2015.
- [55] R. F. Teal. Carpooling: Who, how and why. *Transportation Research Part A: General*, 21(3):203 – 214, 1987.
- [56] A. Toor. French taxi drivers lock down Paris in huge anti-Uber protest. *The Verge*, 2015. <http://www.theverge.com/2015/6/25/8844649/french-taxi-driver-protest-uber-pop-paris>, Online; accedido en Julio 2015.
- [57] M. Tran. Taxi drivers in European capitals strike over Uber - as it happened. *The Guardian*, 2014. <http://www.theguardian.com/politics/2014/jun/11/taxi-drivers-strike-uber-london-live-updates>, Online; accedido en Julio 2015.
- [58] Announcing UberPool. <http://newsroom.uber.com/announcing-uberpool/>. Online; accedido en Julio 2015.
- [59] Unidad Nacional de Seguridad Vial. Evidencia e implicancias del binomio alcohol–conducción en el Uruguay. UNASEV, 2015. www.fing.edu.uy/inco/grupos/cecal/hpc/AG-Taxi/unasev.pdf, Online; accedido en Julio 2015.
- [60] J. Wargo. *PhoneGap Essentials: Building Cross-Platform Mobile Apps*. Addison-Wesley Professional, Primera edición, 2012.
- [61] S. Yan y C.-Y. Chen. A model and a solution algorithm for the car pooling problem with pre-matching information. *Computers & Industrial Engineering*, 61(3):512 – 524, 2011.
- [62] Q. Zhang y H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

Apéndice A

Resultados experimentales

<i>instancia</i>	<i>métrica</i>		<i>seqEA</i>	<i>pμEA</i>
1	mejora sobre alg. ávido (%)	máxima	20.9	20.9
		promedio	20.1	20.9
		std	1.4	0.0
	tiempo (s)	mínimo	3.8	0.6
		promedio	3.9	1.0
		std	0.0	0.3
2	mejora sobre alg. ávido (%)	máxima	32.9	32.9
		promedio	30.0	32.9
		std	2.9	0.0
	tiempo (s)	mínimo	4.5	0.8
		promedio	4.5	1.2
		std	0.0	0.3
3	mejora sobre alg. ávido (%)	máxima	5.8	5.8
		promedio	5.8	5.8
		std	0.0	0.0
	tiempo (s)	mínimo	3.5	0.7
		promedio	3.6	1.0
		std	0.0	0.2
4	mejora sobre alg. ávido (%)	máxima	9.5	9.5
		promedio	9.4	8.9
		std	0.1	1.1
	tiempo (s)	mínimo	4.8	0.7
		promedio	4.9	1.3
		std	0.1	0.3
5	mejora sobre alg. ávido (%)	máxima	31.4	27.3
		promedio	28.7	26.4
		std	3.7	2.1
	tiempo (s)	mínimo	4.7	0.7
		promedio	4.9	1.2
		std	0.1	0.3
6	mejora sobre alg. ávido (%)	máxima	16.9	16.9
		promedio	15.8	16.9
		std	2.3	0.0
	tiempo (s)	mínimo	3.9	0.6
		promedio	3.9	1.1
		std	0.0	0.2

Tabla A.1: Comparativa: mejoras y tiempos de ejecución entre *seqEA* y *pμEA* para instancias chicas.

<i>instancia</i>	<i>métrica</i>		<i>seqEA</i>	<i>pμEA</i>
1	mejora sobre alg. ávido (%)	máxima	26.0	26.0
		promedio	23.7	26.0
		std	2.3	0.0
	tiempo (s)	mínimo	5.2	1.0
		promedio	5.2	1.4
		std	0.0	0.3
2	mejora sobre alg. ávido (%)	máxima	19.4	19.6
		promedio	15.3	19.2
		std	2.9	0.5
	tiempo (s)	mínimo	7.2	1.1
		promedio	7.3	1.7
		std	0.0	0.3
3	mejora sobre alg. ávido (%)	máxima	18.7	17.7
		promedio	13.4	15.9
		std	2.0	1.4
	tiempo (s)	mínimo	9.1	1.3
		promedio	9.2	2.0
		std	0.1	0.5
4	mejora sobre alg. ávido (%)	máxima	20.7	21.1
		promedio	15.5	21.0
		std	2.3	0.0
	tiempo (s)	mínimo	6.8	1.0
		promedio	6.9	1.7
		std	0.0	0.4
5	mejora sobre alg. ávido (%)	máxima	10.1	10.0
		promedio	8.3	10.0
		std	2.4	0.0
	tiempo (s)	mínimo	9.6	1.4
		promedio	9.7	2.2
		std	0.1	0.6
6	mejora sobre alg. ávido (%)	máxima	13.6	13.6
		promedio	8.3	13.0
		std	2.6	1.2
	tiempo (s)	mínimo	7.7	1.1
		promedio	7.8	1.8
		std	0.0	0.3

Tabla A.2: Comparativa: mejoras y tiempos de ejecución entre *seqEA* y *pμEA* para instancias medianas.

<i>instancia</i>	<i>métrica</i>		<i>seqEA</i>	<i>pμEA</i>
1	mejora sobre alg. ávido (%)	máxima	5.1	4.1
		promedio	1.9	1.3
		std	1.2	0.7
	tiempo (s)	mínimo	11.7	1.6
		promedio	11.8	2.6
		std	0.1	0.6
2	mejora sobre alg. ávido (%)	máxima	7.3	7.8
		promedio	4.9	6.9
		std	0.9	0.5
	tiempo (s)	mínimo	11.2	1.6
		promedio	11.3	2.5
		std	0.1	0.5
3	mejora sobre alg. ávido (%)	máxima	12.1	14.4
		promedio	6.8	12.5
		std	2.4	1.4
	tiempo (s)	mínimo	11.2	1.5
		promedio	11.3	2.5
		std	0.1	0.6
4	mejora sobre alg. ávido (%)	máxima	35.9	41.0
		promedio	22.6	38.4
		std	8.0	2.4
	tiempo (s)	mínimo	14.1	1.9
		promedio	14.2	2.9
		std	0.1	0.8
5	mejora sobre alg. ávido (%)	máxima	19.8	28.1
		promedio	10.6	24.8
		std	4.2	1.5
	tiempo (s)	mínimo	21.7	2.9
		promedio	21.9	4.3
		std	0.2	0.9
6	mejora sobre alg. ávido (%)	máxima	13.1	18.7
		promedio	6.8	16.4
		std	2.3	1.3
	tiempo (s)	mínimo	15.4	2.5
		promedio	15.5	3.6
		std	0.1	0.7

Tabla A.3: Comparativa: mejoras y tiempos de ejecución entre *seqEA* y *pμEA* para instancias grandes.

<i>instancia</i>	<i>métrica</i>		<i>seqEA</i>	<i>pμEA</i>
1	mejora sobre alg. ávido (%)	máxima	2.9	2.9
		promedio	2.9	2.9
		std	0.0	0.0
	tiempo (s)	mínimo	3.6	0.6
		promedio	3.6	1.0
		std	0.0	0.2
2	mejora sobre alg. ávido (%)	máxima	24.7	22.6
		promedio	20.3	21.2
		std	1.4	1.2
	tiempo (s)	mínimo	6.0	1.0
		promedio	6.1	1.6
		std	0.0	0.4
3	mejora sobre alg. ávido (%)	máxima	9.1	9.1
		promedio	8.2	9.1
		std	0.9	0.0
	tiempo (s)	mínimo	6.4	0.9
		promedio	6.5	1.4
		std	0.0	0.3
4	mejora sobre alg. ávido (%)	máxima	8.9	8.6
		promedio	8.4	8.5
		std	0.2	0.2
	tiempo (s)	mínimo	6.8	1.1
		promedio	6.9	1.6
		std	0.0	0.3

Tabla A.4: Comparativa: mejoras y tiempos de ejecución entre *seqEA* y *pμEA* para instancias de Montevideo.

	#ND	DG	spacing	spread	RHV
ch#1v1	8.4±1.5 (12.0)	1.6±0.9 (0.0)	413.7±82.0 (202.8)	0.5±0.1 (0.3)	0.9±0.0 (1.0)
ch#1v2	9.5±1.1 (12.0)	1.9±1.3 (0.0)	377.3±84.9 (219.8)	0.6±0.1 (0.4)	0.9±0.0 (1.0)
ch#1v3	9.7±1.3 (12.0)	1.4±1.1 (0.0)	433.2±355.2 (104.5)	0.7±0.1 (0.5)	0.9±0.0 (1.0)
ch#1v4	10.0±1.3 (13.0)	6.0±3.4 (1.1)	1249.9±996.6 (306.0)	0.6±0.2 (0.3)	0.9±0.0 (1.0)
ch#2v1	12.0±1.6 (16.0)	2.6±0.6 (1.1)	597.5±142.0 (296.0)	0.6±0.1 (0.4)	0.9±0.0 (1.0)
ch#2v2	12.3±1.7 (15.0)	2.1±0.7 (0.9)	728.0±663.3 (179.7)	0.8±0.1 (0.4)	1.0±0.0 (1.0)
ch#2v3	11.5±1.7 (14.0)	2.8±0.6 (1.7)	675.4±399.9 (317.5)	0.7±0.1 (0.5)	0.9±0.0 (1.0)
ch#2v4	9.6±1.9 (13.0)	8.9±3.6 (3.6)	2857.4±1656.7 (561.6)	0.7±0.1 (0.3)	0.9±0.0 (0.9)
ch#3v1	7.6±1.4 (10.0)	1.2±1.1 (0.0)	353.9±190.2 (135.9)	0.6±0.1 (0.4)	0.9±0.0 (1.0)
ch#3v2	8.5±1.3 (11.0)	1.5±0.9 (0.3)	419.6±244.5 (106.5)	0.6±0.1 (0.3)	0.9±0.0 (1.0)
ch#3v3	6.7±1.2 (9.0)	1.1±1.0 (0.0)	296.9±182.2 (104.5)	0.6±0.1 (0.3)	0.9±0.0 (1.0)
ch#3v4	7.7±1.7 (10.0)	0.8±0.8 (0.0)	320.2±204.0 (58.1)	0.6±0.2 (0.3)	1.0±0.0 (1.0)
ch#4v1	6.1±1.0 (8.0)	1.8±0.8 (0.0)	432.9±142.7 (176.4)	0.6±0.1 (0.4)	0.8±0.0 (0.9)
ch#4v2	7.1±1.2 (10.0)	1.6±0.7 (0.0)	352.4±116.7 (121.7)	0.5±0.1 (0.2)	0.8±0.1 (0.9)
ch#4v3	7.1±0.9 (9.0)	2.1±0.9 (0.0)	650.5±256.7 (192.7)	0.5±0.1 (0.2)	0.9±0.0 (1.0)
ch#4v4	6.9±1.2 (10.0)	2.2±0.7 (0.0)	380.0±118.5 (162.0)	0.5±0.1 (0.3)	0.8±0.0 (0.9)
ch#5v1	8.6±1.6 (13.0)	5.5±1.4 (2.2)	555.5±287.8 (117.7)	0.5±0.1 (0.2)	0.9±0.0 (0.9)
ch#5v2	8.7±1.1 (12.0)	4.5±1.4 (1.8)	785.4±409.8 (199.6)	0.6±0.1 (0.5)	0.9±0.0 (0.9)
ch#5v3	7.7±1.3 (11.0)	5.1±2.2 (0.9)	1176.3±650.4 (367.8)	0.6±0.1 (0.4)	0.9±0.0 (0.9)
ch#5v4	8.6±0.8 (11.0)	4.2±1.6 (1.6)	629.1±418.7 (191.0)	0.7±0.1 (0.5)	0.9±0.0 (0.9)
ch#6v1	8.1±1.0 (10.0)	3.8±2.1 (1.2)	698.3±337.9 (228.6)	0.4±0.1 (0.2)	0.9±0.0 (0.9)
ch#6v2	7.3±1.2 (10.0)	3.9±1.5 (1.5)	1194.8±379.2 (353.9)	0.4±0.2 (0.1)	0.9±0.0 (1.0)
ch#6v3	7.3±1.2 (10.0)	3.5±2.0 (0.6)	1127.7±546.7 (113.9)	0.7±0.1 (0.4)	0.9±0.0 (0.9)
ch#6v4	6.3±1.1 (8.0)	3.4±2.2 (0.0)	1058.4±611.7 (309.2)	0.6±0.2 (0.1)	0.9±0.0 (1.0)

Tabla A.5: Métricas multiobjetivo *pμMOEA/D* – instancias chicas.

	#ND	DG	spacing	spread	RHV
me#1v1	9.5±1.6 (13.0)	5.4±1.2 (2.1)	892.6±221.2 (612.7)	0.6±0.1 (0.4)	0.9±0.0 (0.9)
me#1v2	10.7±1.4 (13.0)	2.8±1.2 (1.2)	1005.6±537.0 (192.9)	0.6±0.1 (0.3)	0.9±0.0 (1.0)
me#1v3	8.2±1.4 (11.0)	6.3±1.6 (2.4)	1066.1±365.4 (535.4)	0.6±0.2 (0.2)	0.8±0.1 (0.9)
me#1v4	9.6±1.9 (13.0)	5.8±2.2 (1.7)	719.5±357.8 (315.1)	0.5±0.1 (0.2)	0.9±0.0 (0.9)
me#2v1	8.4±1.5 (11.0)	6.0±2.5 (3.5)	1350.0±675.3 (735.3)	0.5±0.1 (0.2)	0.8±0.0 (0.9)
me#2v2	9.2±1.4 (12.0)	5.2±1.4 (2.8)	3346.2±1640.1 (915.3)	0.5±0.1 (0.3)	0.9±0.0 (0.9)
me#2v3	9.6±1.6 (13.0)	5.9±1.6 (4.3)	1159.3±561.1 (577.5)	0.6±0.1 (0.3)	0.9±0.0 (0.9)
me#2v4	10.8±1.3 (13.0)	10.6±2.5 (3.1)	1145.0±519.1 (210.7)	0.7±0.1 (0.3)	0.8±0.0 (0.9)
me#3v1	9.2±1.7 (14.0)	5.0±3.1 (2.2)	2175.0±1236.6 (1019.7)	0.7±0.1 (0.4)	0.8±0.0 (0.9)
me#3v2	12.4±2.5 (19.0)	4.8±0.9 (2.5)	1023.5±329.7 (569.0)	0.6±0.1 (0.3)	0.9±0.0 (0.9)
me#3v3	10.2±1.2 (13.0)	6.4±2.1 (3.3)	1968.5±934.2 (802.3)	0.5±0.1 (0.3)	0.9±0.0 (0.9)
me#3v4	11.9±1.4 (14.0)	4.6±1.1 (3.1)	1909.4±411.8 (1222.2)	0.6±0.1 (0.4)	0.9±0.0 (0.9)
me#4v1	7.1±1.7 (10.0)	5.1±2.7 (0.0)	1039.5±599.9 (320.5)	0.6±0.1 (0.4)	0.8±0.1 (0.9)
me#4v2	9.2±1.2 (12.0)	5.0±0.8 (3.1)	2012.2±1072.6 (449.1)	0.7±0.1 (0.5)	0.9±0.0 (0.9)
me#4v3	6.7±0.9 (9.0)	4.4±1.7 (2.0)	1099.5±339.8 (141.6)	0.4±0.1 (0.2)	0.8±0.1 (0.9)
me#4v4	7.6±1.3 (10.0)	4.3±2.9 (0.3)	771.3±772.4 (163.9)	0.6±0.1 (0.3)	0.8±0.1 (0.9)
me#5v1	9.9±1.7 (13.0)	5.2±1.1 (3.2)	1183.3±371.2 (696.1)	0.7±0.1 (0.6)	0.9±0.0 (1.0)
me#5v2	9.9±2.1 (13.0)	4.3±1.0 (3.3)	1105.1±239.8 (443.6)	0.6±0.1 (0.3)	0.9±0.0 (0.9)
me#5v3	10.6±1.5 (13.0)	8.1±2.0 (5.0)	3374.2±1959.1 (993.1)	0.6±0.1 (0.3)	0.9±0.0 (0.9)
me#5v4	8.8±1.7 (13.0)	7.9±1.6 (4.0)	1982.1±684.1 (1096.2)	0.7±0.1 (0.5)	0.9±0.0 (1.0)
me#6v1	6.6±1.3 (9.0)	6.7±3.5 (1.6)	1718.9±698.5 (613.9)	0.5±0.1 (0.2)	0.8±0.1 (0.9)
me#6v2	8.6±1.6 (11.0)	6.4±1.8 (2.8)	1074.3±411.5 (387.3)	0.5±0.1 (0.3)	0.8±0.0 (0.9)
me#6v3	7.5±1.4 (10.0)	5.0±1.6 (3.5)	763.7±420.3 (385.9)	0.5±0.1 (0.1)	0.8±0.0 (0.9)
me#6v4	7.2±1.5 (11.0)	5.4±2.4 (1.2)	879.4±443.8 (301.9)	0.5±0.1 (0.2)	0.8±0.1 (0.9)

Tabla A.6: Métricas multiobjetivo $p\mu MOEA/D$ – instancias medianas.

	#ND	DG	spacing	spread	RHV
gr#1v1	8.9±1.0 (10.0)	7.9±1.0 (5.3)	1387.5±305.5 (790.3)	0.5±0.1 (0.2)	0.9±0.0 (0.9)
gr#1v2	8.5±1.4 (12.0)	5.9±1.1 (2.7)	3550.0±725.3 (992.2)	0.5±0.1 (0.2)	0.9±0.0 (0.9)
gr#1v3	8.3±1.6 (12.0)	6.6±2.1 (2.5)	2636.6±1339.4 (755.9)	0.6±0.1 (0.4)	0.9±0.0 (0.9)
gr#1v4	6.7±1.4 (10.0)	4.4±1.0 (2.5)	1079.4±539.8 (175.3)	0.6±0.1 (0.2)	0.9±0.1 (0.9)
gr#2v1	9.3±1.6 (13.0)	12.0±3.5 (5.1)	2374.7±867.4 (901.7)	0.6±0.1 (0.3)	0.8±0.0 (0.9)
gr#2v2	11.6±1.9 (17.0)	6.5±1.1 (4.6)	5987.5±1924.1 (1159.1)	0.6±0.1 (0.3)	0.9±0.0 (1.0)
gr#2v3	9.8±1.3 (12.0)	5.3±1.3 (2.3)	2186.7±817.7 (463.1)	0.7±0.2 (0.4)	0.9±0.0 (0.9)
gr#2v4	11.2±1.7 (14.0)	8.9±2.2 (5.0)	2211.3±1249.1 (852.6)	0.7±0.1 (0.4)	0.9±0.0 (0.9)
gr#3v1	6.7±1.2 (9.0)	5.5±1.4 (3.0)	1077.3±388.8 (545.4)	0.4±0.1 (0.0)	0.7±0.1 (0.9)
gr#3v2	9.0±1.6 (12.0)	6.6±1.5 (3.4)	618.3±183.0 (266.6)	0.5±0.1 (0.2)	0.8±0.0 (0.9)
gr#3v3	8.7±1.5 (11.0)	5.4±1.1 (2.8)	853.2±234.8 (492.7)	0.5±0.1 (0.2)	0.8±0.0 (0.9)
gr#3v4	8.0±1.6 (12.0)	4.8±1.0 (2.0)	645.6±261.8 (340.8)	0.6±0.1 (0.3)	0.9±0.0 (0.9)
gr#4v1	6.5±1.6 (10.0)	7.4±4.3 (3.0)	2111.9±1024.9 (888.8)	0.5±0.2 (0.1)	0.7±0.1 (0.9)
gr#4v2	7.8±1.8 (12.0)	8.9±3.0 (4.0)	4361.6±2297.6 (611.0)	0.5±0.2 (0.2)	0.9±0.0 (0.9)
gr#4v3	7.6±1.3 (10.0)	15.4±4.8 (5.2)	5112.5±1987.5 (1805.5)	0.6±0.1 (0.5)	0.8±0.1 (0.9)
gr#4v4	7.6±1.7 (12.0)	11.9±2.4 (7.2)	5442.5±2021.3 (1838.8)	0.5±0.1 (0.3)	0.9±0.0 (0.9)
gr#5v1	6.4±1.2 (9.0)	8.8±2.7 (3.5)	3937.9±1345.8 (2080.4)	0.5±0.2 (0.1)	0.7±0.1 (0.9)
gr#5v2	8.3±1.3 (12.0)	10.6±3.4 (3.7)	5375.5±1281.2 (1984.7)	0.7±0.1 (0.5)	0.8±0.1 (0.9)
gr#5v3	7.5±1.5 (11.0)	7.6±1.7 (4.1)	2855.2±856.0 (1419.5)	0.6±0.1 (0.2)	0.8±0.1 (0.9)
gr#5v4	6.6±1.8 (11.0)	7.4±1.9 (3.1)	2125.2±728.8 (720.0)	0.7±0.1 (0.3)	0.8±0.1 (0.9)
gr#6v1	8.3±2.1 (13.0)	8.2±2.4 (5.4)	2406.1±1197.6 (798.4)	0.6±0.1 (0.3)	0.9±0.0 (0.9)
gr#6v2	9.3±1.9 (14.0)	8.2±2.2 (4.8)	5029.2±1749.1 (1910.8)	0.6±0.1 (0.3)	0.9±0.0 (0.9)
gr#6v3	10.6±1.9 (15.0)	8.3±1.4 (5.4)	3703.3±1681.3 (1158.7)	0.6±0.1 (0.3)	0.9±0.0 (0.9)
gr#6v4	10.7±1.6 (15.0)	7.8±1.4 (4.2)	2943.5±1377.4 (1043.7)	0.6±0.1 (0.4)	0.9±0.0 (0.9)

Tabla A.7: Métricas multiobjetivo $p\mu MOEA/D$ – instancias grandes.

	#ND	DG	spacing	spread	RHV
Mo#1v1	5.8±0.8 (7.0)	0.6±0.6 (0.0)	193.1±32.7 (126.9)	0.3±0.1 (0.0)	0.9±0.1 (1.0)
Mo#1v2	5.7±0.8 (8.0)	0.6±0.6 (0.0)	184.6±44.8 (69.3)	0.3±0.1 (0.1)	0.9±0.1 (1.0)
Mo#1v3	6.7±1.0 (9.0)	0.3±0.6 (0.0)	199.2±111.6 (61.5)	0.7±0.1 (0.5)	1.0±0.0 (1.0)
Mo#1v4	6.7±1.4 (9.0)	1.6±1.0 (0.0)	336.9±153.5 (90.7)	0.4±0.1 (0.1)	0.9±0.1 (0.9)
Mo#2v1	8.5±1.4 (11.0)	6.1±2.2 (2.3)	1726.2±740.8 (526.1)	0.8±0.1 (0.5)	0.9±0.0 (0.9)
Mo#2v2	11.2±1.2 (14.0)	2.1±0.7 (0.8)	320.7±149.8 (78.6)	0.7±0.1 (0.6)	0.9±0.0 (1.0)
Mo#2v3	8.2±1.6 (12.0)	3.0±1.3 (1.5)	610.4±283.6 (208.3)	0.7±0.1 (0.5)	0.8±0.1 (0.9)
Mo#2v4	8.7±1.1 (11.0)	4.7±0.9 (2.7)	1436.3±451.6 (484.1)	0.6±0.1 (0.4)	0.9±0.0 (0.9)
Mo#3v1	8.6±1.2 (11.0)	3.2±0.9 (1.7)	586.6±419.6 (126.3)	0.6±0.1 (0.3)	0.9±0.0 (1.0)
Mo#3v2	8.9±1.6 (12.0)	3.0±1.3 (0.8)	687.0±185.3 (245.8)	0.5±0.1 (0.2)	0.9±0.0 (1.0)
Mo#3v3	9.0±1.4 (12.0)	4.1±1.1 (2.2)	729.9±395.8 (179.5)	0.6±0.1 (0.3)	0.9±0.0 (1.0)
Mo#3v4	9.5±1.1 (12.0)	3.7±0.6 (1.8)	789.3±221.8 (309.7)	0.6±0.1 (0.3)	0.9±0.0 (0.9)
Mo#4v1	8.7±1.6 (13.0)	5.9±1.0 (4.4)	553.8±142.1 (292.1)	0.6±0.1 (0.3)	0.9±0.0 (1.0)
Mo#4v2	5.4±0.8 (7.0)	2.6±0.8 (0.0)	583.3±239.8 (286.5)	0.7±0.1 (0.5)	0.9±0.0 (0.9)
Mo#4v3	10.2±1.5 (14.0)	3.5±0.9 (1.9)	963.0±617.3 (204.7)	0.6±0.2 (0.2)	0.9±0.0 (0.9)
Mo#4v4	6.7±1.6 (11.0)	3.5±1.8 (0.0)	715.8±425.0 (223.4)	0.6±0.1 (0.2)	0.9±0.0 (0.9)

Tabla A.8: Métricas multiobjetivo $\mu MOEA/D$ – instancias Montevideo.

	#ND	DG	spacing	spread	RHV
ch#1v1	27.0±0.0 (27.0)	0.0±0.0 (0.0)	254.9±0.0 (254.9)	0.9±0.0 (0.9)	1.0±0.0 (1.0)
ch#1v2	27.0±0.0 (27.0)	0.0±0.0 (0.0)	254.9±0.0 (254.9)	0.9±0.0 (0.9)	1.0±0.0 (1.0)
ch#1v3	24.0±0.3 (25.0)	0.0±0.0 (0.0)	170.8±86.2 (151.7)	0.9±0.0 (0.9)	1.0±0.0 (1.0)
ch#1v4	27.0±0.0 (27.0)	0.0±0.0 (0.0)	254.9±0.0 (254.9)	0.9±0.0 (0.9)	1.0±0.0 (1.0)
ch#2v1	51.4±1.0 (53.0)	0.2±0.3 (0.0)	263.8±7.1 (245.7)	0.9±0.0 (0.8)	1.0±0.0 (1.0)
ch#2v2	47.6±1.1 (50.0)	0.1±0.2 (0.0)	243.1±132.5 (84.5)	0.9±0.0 (0.8)	1.0±0.0 (1.0)
ch#2v3	47.7±1.0 (49.0)	0.1±0.2 (0.0)	220.7±136.7 (65.0)	0.9±0.0 (0.8)	1.0±0.0 (1.0)
ch#2v4	51.7±1.7 (55.0)	1.0±0.4 (0.0)	760.4±241.9 (127.3)	1.0±0.0 (0.9)	1.0±0.0 (1.0)
ch#3v1	22.9±1.8 (26.0)	0.6±0.6 (0.0)	151.8±26.3 (127.3)	0.9±0.0 (0.9)	1.0±0.0 (1.0)
ch#3v2	22.6±0.6 (24.0)	0.0±0.0 (0.0)	142.6±0.9 (141.7)	0.9±0.0 (0.9)	1.0±0.0 (1.0)
ch#3v3	22.6±1.5 (25.0)	0.8±0.6 (0.0)	138.8±7.7 (122.7)	0.9±0.0 (0.9)	1.0±0.0 (1.0)
ch#3v4	22.0±1.5 (23.0)	1.0±0.5 (0.0)	155.3±23.8 (120.9)	0.9±0.0 (0.9)	1.0±0.0 (1.0)
ch#4v1	39.3±0.6 (41.0)	0.0±0.1 (0.0)	102.2±8.4 (57.2)	0.8±0.0 (0.8)	1.0±0.0 (1.0)
ch#4v2	38.9±0.5 (40.0)	0.0±0.1 (0.0)	104.9±28.9 (62.1)	0.8±0.0 (0.8)	1.0±0.0 (1.0)
ch#4v3	39.3±0.8 (42.0)	0.0±0.0 (0.0)	104.0±0.9 (101.3)	0.8±0.0 (0.8)	1.0±0.0 (1.0)
ch#4v4	39.6±0.8 (42.0)	0.0±0.1 (0.0)	103.1±1.7 (99.2)	0.8±0.0 (0.8)	1.0±0.0 (1.0)
ch#5v1	30.4±1.3 (33.0)	0.5±0.7 (0.0)	142.5±28.0 (128.1)	1.0±0.1 (0.8)	1.0±0.0 (1.0)
ch#5v2	29.3±1.2 (33.0)	0.3±0.5 (0.0)	222.7±165.1 (131.4)	0.9±0.1 (0.8)	1.0±0.0 (1.0)
ch#5v3	34.1±1.5 (36.0)	2.0±0.8 (0.0)	798.1±393.3 (123.0)	1.2±0.1 (1.1)	1.0±0.0 (1.0)
ch#5v4	31.9±1.7 (37.0)	0.2±0.2 (0.0)	231.8±190.1 (123.7)	0.9±0.0 (0.8)	1.0±0.0 (1.0)
ch#6v1	30.1±1.8 (34.0)	0.5±0.8 (0.0)	154.1±138.0 (96.8)	0.9±0.1 (0.7)	1.0±0.0 (1.0)
ch#6v2	29.1±2.0 (32.0)	0.4±0.5 (0.0)	137.4±38.3 (43.2)	0.9±0.0 (0.8)	1.0±0.0 (1.0)
ch#6v3	22.5±0.6 (24.0)	0.1±0.1 (0.0)	217.1±233.5 (55.4)	0.8±0.0 (0.8)	1.0±0.0 (1.0)
ch#6v4	23.7±0.6 (25.0)	0.1±0.3 (0.0)	338.8±238.2 (59.7)	0.9±0.1 (0.8)	1.0±0.0 (1.0)

Tabla A.9: Métricas multiobjetivo $NSGA-II$ – instancias chicas.

	#ND	DG	spacing	spread	RHV
me#1v1	47.8±2.2 (52.0)	0.1±0.1 (0.0)	397.6±358.4 (65.9)	0.9±0.1 (0.7)	1.0±0.0 (1.0)
me#1v2	49.3±1.7 (54.0)	0.6±0.3 (0.0)	103.5±22.4 (57.6)	0.7±0.0 (0.6)	1.0±0.0 (1.0)
me#1v3	47.8±1.7 (51.0)	2.0±1.3 (0.0)	296.6±200.9 (147.6)	1.1±0.1 (0.9)	1.0±0.0 (1.0)
me#1v4	48.0±2.0 (53.0)	0.3±0.4 (0.0)	103.7±125.8 (52.4)	0.7±0.1 (0.6)	1.0±0.0 (1.0)
me#2v1	56.6±2.9 (63.0)	0.7±0.2 (0.3)	205.5±166.0 (69.7)	0.8±0.1 (0.6)	1.0±0.0 (1.0)
me#2v2	53.4±2.6 (58.0)	1.6±0.4 (0.8)	368.0±201.4 (122.8)	0.7±0.1 (0.5)	1.0±0.0 (1.0)
me#2v3	56.0±2.7 (61.0)	1.1±0.2 (0.6)	251.6±188.4 (70.0)	0.7±0.1 (0.5)	1.0±0.0 (1.0)
me#2v4	54.4±3.0 (62.0)	1.6±0.5 (0.5)	231.8±196.5 (105.3)	0.7±0.1 (0.6)	1.0±0.0 (1.0)
me#3v1	53.2±2.9 (59.0)	0.7±0.2 (0.4)	105.1±65.6 (60.9)	0.5±0.1 (0.4)	1.0±0.0 (1.0)
me#3v2	55.0±2.5 (60.0)	1.2±0.5 (0.7)	174.8±142.5 (71.1)	0.6±0.1 (0.4)	1.0±0.0 (1.0)
me#3v3	56.3±2.3 (61.0)	2.0±0.8 (0.9)	158.2±59.6 (91.1)	0.6±0.1 (0.4)	1.0±0.0 (1.0)
me#3v4	58.6±3.3 (67.0)	1.6±0.2 (1.1)	297.0±196.3 (115.7)	0.6±0.1 (0.5)	1.0±0.0 (1.0)
me#4v1	54.1±2.4 (60.0)	0.4±0.1 (0.2)	98.3±94.7 (26.5)	0.7±0.0 (0.6)	1.0±0.0 (1.0)
me#4v2	55.3±2.4 (60.0)	1.2±1.2 (0.3)	248.6±269.5 (66.2)	0.9±0.1 (0.6)	1.0±0.0 (1.0)
me#4v3	56.7±2.2 (61.0)	0.6±0.1 (0.3)	81.0±50.7 (39.6)	0.5±0.1 (0.4)	1.0±0.0 (1.0)
me#4v4	53.4±2.8 (59.0)	0.4±0.1 (0.2)	122.2±132.9 (26.2)	0.7±0.1 (0.5)	1.0±0.0 (1.0)
me#5v1	54.8±1.9 (58.0)	0.8±0.3 (0.4)	166.8±157.1 (41.6)	0.5±0.1 (0.5)	1.0±0.0 (1.0)
me#5v2	54.9±2.8 (61.0)	0.8±0.1 (0.6)	108.9±95.2 (52.7)	0.5±0.1 (0.4)	1.0±0.0 (1.0)
me#5v3	56.5±2.8 (62.0)	2.3±0.9 (1.1)	243.4±130.0 (116.1)	0.7±0.1 (0.6)	1.0±0.0 (1.0)
me#5v4	58.1±2.9 (63.0)	1.1±0.3 (0.7)	257.6±269.6 (58.9)	0.7±0.1 (0.5)	1.0±0.0 (1.0)
me#6v1	56.9±3.3 (62.0)	0.8±0.3 (0.4)	173.1±207.0 (46.4)	0.6±0.1 (0.5)	1.0±0.0 (1.0)
me#6v2	58.5±3.1 (66.0)	0.5±0.2 (0.2)	223.0±353.5 (34.5)	0.7±0.1 (0.4)	1.0±0.0 (1.0)
me#6v3	56.2±3.2 (65.0)	0.8±0.3 (0.4)	103.8±49.6 (44.8)	0.6±0.1 (0.5)	1.0±0.0 (1.0)
me#6v4	57.3±3.1 (66.0)	1.2±0.3 (0.6)	125.3±107.6 (44.7)	0.6±0.1 (0.5)	1.0±0.0 (1.0)

Tabla A.10: Métricas multiobjetivo *NSGA-II* – instancias medianas.

	#ND	DG	spacing	spread	RHV
gr#1v1	54.2±3.4 (63.0)	2.1±0.5 (1.1)	373.3±199.0 (143.9)	1.0±0.1 (0.9)	1.0±0.0 (1.0)
gr#1v2	54.8±3.2 (62.0)	2.3±1.0 (0.8)	490.6±293.3 (144.9)	1.0±0.1 (0.9)	1.0±0.0 (1.0)
gr#1v3	56.2±3.5 (63.0)	1.6±0.5 (0.9)	357.0±311.4 (77.9)	0.9±0.1 (0.8)	1.0±0.0 (1.0)
gr#1v4	58.2±2.4 (65.0)	1.2±0.4 (0.4)	232.7±275.2 (26.4)	0.9±0.1 (0.7)	1.0±0.0 (1.0)
gr#2v1	56.7±3.0 (62.0)	2.3±1.0 (1.0)	230.0±216.7 (110.4)	0.6±0.1 (0.4)	1.0±0.0 (1.0)
gr#2v2	56.7±2.6 (61.0)	2.5±0.7 (1.5)	375.3±196.3 (158.4)	0.9±0.1 (0.7)	1.0±0.0 (1.0)
gr#2v3	55.4±2.9 (63.0)	1.6±0.3 (1.1)	290.0±218.0 (126.2)	0.7±0.1 (0.5)	1.0±0.0 (1.0)
gr#2v4	57.7±2.7 (62.0)	1.7±0.4 (1.1)	249.1±122.6 (131.8)	0.7±0.1 (0.6)	1.0±0.0 (1.0)
gr#3v1	56.9±3.7 (67.0)	0.7±0.2 (0.4)	63.3±19.4 (38.3)	0.6±0.1 (0.5)	1.0±0.0 (1.0)
gr#3v2	55.5±3.2 (61.0)	1.0±0.8 (0.4)	99.4±105.3 (33.3)	0.6±0.1 (0.4)	1.0±0.0 (1.0)
gr#3v3	56.4±3.1 (64.0)	0.8±0.4 (0.4)	119.7±143.3 (29.1)	0.6±0.1 (0.4)	1.0±0.0 (1.0)
gr#3v4	55.8±3.3 (65.0)	0.9±0.3 (0.6)	104.2±120.3 (37.0)	0.6±0.1 (0.4)	1.0±0.0 (1.0)
gr#4v1	53.9±3.4 (61.0)	1.1±0.3 (0.7)	143.7±117.3 (52.3)	0.7±0.1 (0.4)	1.0±0.0 (1.0)
gr#4v2	54.2±2.8 (60.0)	2.8±1.4 (1.1)	278.1±217.7 (69.3)	0.8±0.1 (0.5)	0.9±0.0 (1.0)
gr#4v3	54.7±3.0 (60.0)	2.1±1.3 (1.0)	266.2±269.5 (98.5)	0.8±0.1 (0.6)	1.0±0.0 (1.0)
gr#4v4	53.9±3.6 (60.0)	2.7±1.4 (1.1)	307.9±351.6 (79.9)	0.8±0.1 (0.6)	1.0±0.0 (1.0)
gr#5v1	55.5±3.9 (62.0)	1.8±0.4 (1.2)	233.2±187.0 (145.3)	0.5±0.1 (0.4)	1.0±0.0 (1.0)
gr#5v2	56.6±2.7 (65.0)	1.5±0.4 (1.0)	194.0±142.6 (93.3)	0.7±0.1 (0.5)	1.0±0.0 (1.0)
gr#5v3	56.4±2.9 (62.0)	1.5±0.3 (0.7)	175.1±95.6 (72.8)	0.7±0.1 (0.5)	1.0±0.0 (1.0)
gr#5v4	54.2±2.6 (59.0)	1.2±0.3 (1.0)	134.5±87.6 (55.4)	0.6±0.1 (0.4)	0.9±0.0 (1.0)
gr#6v1	51.3±3.1 (59.0)	1.3±0.4 (0.8)	138.6±90.3 (52.5)	0.7±0.1 (0.5)	0.9±0.0 (1.0)
gr#6v2	52.7±3.3 (59.0)	1.7±0.4 (1.2)	242.9±172.0 (78.0)	0.7±0.1 (0.5)	0.9±0.0 (1.0)
gr#6v3	54.0±2.9 (62.0)	4.0±1.5 (1.0)	368.6±230.8 (166.4)	0.7±0.1 (0.4)	1.0±0.0 (1.0)
gr#6v4	53.8±3.5 (61.0)	2.3±0.4 (1.5)	379.0±275.2 (154.9)	0.7±0.1 (0.5)	1.0±0.0 (1.0)

Tabla A.11: Métricas multiobjetivo *NSGA-II* – instancias grandes.

	#ND	DG	spacing	spread	RHV
Mo#1v1	15.9±0.6 (17.0)	1.2±1.0 (0.0)	296.1±215.9 (54.2)	0.7±0.1 (0.5)	1.0±0.0 (1.0)
Mo#1v2	15.3±0.6 (17.0)	0.4±0.8 (0.0)	117.0±142.5 (55.1)	0.7±0.1 (0.5)	1.0±0.0 (1.0)
Mo#1v3	15.6±0.9 (18.0)	0.3±0.8 (0.0)	166.2±158.1 (51.7)	0.7±0.1 (0.5)	1.0±0.0 (1.0)
Mo#1v4	17.2±0.5 (18.0)	0.0±0.0 (0.0)	101.3±2.2 (98.1)	0.6±0.0 (0.6)	1.0±0.0 (1.0)
Mo#2v1	55.5±2.5 (61.0)	0.6±0.4 (0.2)	194.7±99.9 (61.8)	0.7±0.1 (0.6)	1.0±0.0 (1.0)
Mo#2v2	54.7±3.0 (59.0)	0.5±0.1 (0.3)	202.0±5.7 (191.0)	0.6±0.0 (0.5)	1.0±0.0 (1.0)
Mo#2v3	54.6±2.6 (61.0)	0.3±0.1 (0.0)	42.4±17.0 (25.6)	0.7±0.0 (0.6)	1.0±0.0 (1.0)
Mo#2v4	56.8±1.7 (60.0)	0.5±0.2 (0.3)	103.1±71.9 (29.4)	0.7±0.0 (0.6)	1.0±0.0 (1.0)
Mo#3v1	53.4±2.7 (61.0)	0.4±0.2 (0.2)	104.6±163.9 (24.8)	0.8±0.1 (0.6)	1.0±0.0 (1.0)
Mo#3v2	55.1±1.8 (60.0)	0.3±0.2 (0.1)	38.9±18.8 (20.8)	0.9±0.0 (0.8)	1.0±0.0 (1.0)
Mo#3v3	54.1±2.7 (61.0)	0.4±0.2 (0.1)	159.1±179.6 (28.8)	0.8±0.1 (0.6)	1.0±0.0 (1.0)
Mo#3v4	55.5±1.9 (60.0)	0.3±0.1 (0.1)	34.9±5.9 (24.3)	0.9±0.0 (0.8)	1.0±0.0 (1.0)
Mo#4v1	50.1±2.1 (55.0)	0.7±0.3 (0.3)	231.2±196.3 (46.2)	1.0±0.1 (0.9)	1.0±0.0 (1.0)
Mo#4v2	50.8±3.1 (56.0)	0.3±0.1 (0.2)	71.4±89.0 (22.5)	0.8±0.0 (0.7)	1.0±0.0 (1.0)
Mo#4v3	49.4±2.1 (54.0)	0.6±0.2 (0.2)	227.7±86.2 (48.4)	1.1±0.1 (1.0)	1.0±0.0 (1.0)
Mo#4v4	49.2±2.2 (53.0)	0.3±0.1 (0.1)	186.4±129.5 (21.5)	0.7±0.1 (0.5)	1.0±0.0 (1.0)

Tabla A.12: Métricas multiobjetivo *NSGA-II* – instancias Montevideo.

		mejora greedy (%)			HV($\times 10^6$)	<i>p</i> valores	
		costo	demora	tiempo(s)		S-W	K-W
ch#1v1	p μ MOEA/D	28.1 \pm 0.0 (28.1)	46.4 \pm 0.0 (46.4)	1.1 \pm 0.3 (0.8)	0.5 \pm 0.0 (0.6)	0.7	0.0
	NSGA-II	39.2 \pm 0.0 (39.2)	46.4 \pm 0.0 (46.4)	8.2 \pm 0.4 (7.9)	0.6 \pm 0.0 (0.6)	1.0	
ch#1v2	p μ MOEA/D	27.8 \pm 0.0 (27.8)	37.3 \pm 3.2 (38.9)	1.3 \pm 0.3 (0.9)	0.5 \pm 0.0 (0.6)	0.2	0.0
	NSGA-II	38.7 \pm 0.0 (38.7)	38.9 \pm 0.0 (38.9)	8.2 \pm 0.3 (8.0)	0.6 \pm 0.0 (0.6)	1.0	
ch#1v3	p μ MOEA/D	55.4 \pm 0.0 (55.4)	49.6 \pm 1.8 (51.6)	1.2 \pm 0.3 (0.8)	0.5 \pm 0.0 (0.6)	0.0	0.0
	NSGA-II	55.4 \pm 0.0 (55.4)	51.6 \pm 0.0 (51.6)	8.2 \pm 0.4 (7.8)	0.6 \pm 0.0 (0.6)	0.0	
ch#1v4	p μ MOEA/D	28.3 \pm 0.0 (28.3)	46.1 \pm 0.0 (46.1)	1.2 \pm 0.3 (0.8)	0.5 \pm 0.0 (0.5)	0.6	0.0
	NSGA-II	39.4 \pm 0.0 (39.4)	46.1 \pm 0.0 (46.1)	7.9 \pm 0.2 (7.7)	0.6 \pm 0.0 (0.6)	1.0	
ch#2v1	p μ MOEA/D	16.4 \pm 6.8 (20.5)	45.1 \pm 4.3 (50.8)	1.4 \pm 0.4 (0.9)	3.8 \pm 0.1 (3.9)	0.7	0.0
	NSGA-II	25.3 \pm 0.0 (25.3)	50.6 \pm 0.3 (50.8)	8.4 \pm 0.2 (8.1)	4.1 \pm 0.0 (4.1)	0.0	
ch#2v2	p μ MOEA/D	65.3 \pm 5.8 (74.4)	39.3 \pm 2.8 (43.1)	1.3 \pm 0.3 (1.0)	3.9 \pm 0.0 (4.0)	1.0	0.0
	NSGA-II	74.4 \pm 0.0 (74.4)	43.1 \pm 0.0 (43.1)	8.2 \pm 0.3 (8.0)	4.2 \pm 0.0 (4.2)	0.0	
ch#2v3	p μ MOEA/D	67.8 \pm 6.5 (77.5)	48.0 \pm 1.2 (50.5)	1.2 \pm 0.3 (1.0)	3.4 \pm 0.1 (3.5)	0.4	0.0
	NSGA-II	77.5 \pm 0.0 (77.5)	50.3 \pm 0.6 (53.0)	8.2 \pm 0.2 (8.0)	3.7 \pm 0.0 (3.7)	0.0	
ch#2v4	p μ MOEA/D	60.3 \pm 7.1 (77.7)	52.2 \pm 2.1 (57.3)	1.4 \pm 0.4 (0.9)	4.5 \pm 0.1 (4.8)	0.5	0.0
	NSGA-II	77.7 \pm 0.0 (77.7)	58.4 \pm 0.4 (58.4)	8.2 \pm 0.2 (8.0)	5.2 \pm 0.0 (5.2)	0.0	
ch#3v1	p μ MOEA/D	31.2 \pm 10.2 (38.9)	29.5 \pm 2.6 (32.7)	1.0 \pm 0.2 (0.7)	0.4 \pm 0.0 (0.4)	0.0	0.0
	NSGA-II	38.9 \pm 0.0 (38.9)	32.2 \pm 1.5 (33.4)	7.8 \pm 0.2 (7.5)	0.4 \pm 0.0 (0.4)	0.0	
ch#3v2	p μ MOEA/D	26.4 \pm 0.4 (26.6)	53.3 \pm 2.2 (55.2)	1.2 \pm 0.3 (0.8)	0.4 \pm 0.0 (0.4)	0.0	0.0
	NSGA-II	38.0 \pm 0.0 (38.0)	55.2 \pm 0.0 (55.2)	7.8 \pm 0.3 (7.6)	0.4 \pm 0.0 (0.4)	0.0	
ch#3v3	p μ MOEA/D	24.7 \pm 10.7 (35.7)	49.8 \pm 1.6 (52.9)	1.2 \pm 0.3 (0.7)	0.4 \pm 0.0 (0.4)	0.0	0.0
	NSGA-II	35.7 \pm 0.0 (35.7)	52.9 \pm 0.7 (53.5)	7.8 \pm 0.2 (7.6)	0.4 \pm 0.0 (0.4)	0.0	
ch#3v4	p μ MOEA/D	24.4 \pm 11.8 (37.5)	36.5 \pm 1.6 (40.4)	1.1 \pm 0.3 (0.8)	0.4 \pm 0.0 (0.4)	0.0	0.0
	NSGA-II	37.5 \pm 0.0 (37.5)	40.1 \pm 1.3 (41.3)	7.6 \pm 0.2 (7.3)	0.4 \pm 0.0 (0.4)	0.0	
ch#4v1	p μ MOEA/D	0.0 \pm 0.0 (0.0)	51.9 \pm 0.5 (53.8)	1.6 \pm 0.4 (1.1)	0.5 \pm 0.0 (0.5)	0.2	0.0
	NSGA-II	4.5 \pm 0.0 (4.5)	56.2 \pm 0.6 (56.3)	8.5 \pm 0.2 (8.2)	0.6 \pm 0.0 (0.6)	0.0	
ch#4v2	p μ MOEA/D	0.0 \pm 0.0 (0.0)	54.7 \pm 0.4 (55.6)	1.4 \pm 0.4 (1.1)	0.5 \pm 0.0 (0.6)	0.1	0.0
	NSGA-II	4.2 \pm 0.0 (4.2)	58.5 \pm 1.0 (58.9)	8.6 \pm 0.2 (8.4)	0.6 \pm 0.0 (0.6)	0.0	
ch#4v3	p μ MOEA/D	0.0 \pm 0.0 (0.0)	54.0 \pm 0.5 (54.2)	1.5 \pm 0.4 (0.9)	0.4 \pm 0.0 (0.5)	0.5	0.0
	NSGA-II	4.0 \pm 0.7 (4.3)	56.0 \pm 0.0 (56.0)	8.4 \pm 0.2 (8.3)	0.6 \pm 0.0 (0.6)	0.0	
ch#4v4	p μ MOEA/D	0.3 \pm 1.1 (4.5)	55.2 \pm 0.8 (59.3)	1.5 \pm 0.4 (1.1)	0.5 \pm 0.0 (0.5)	0.1	0.0
	NSGA-II	4.5 \pm 0.0 (4.5)	59.3 \pm 0.0 (59.3)	8.5 \pm 0.4 (8.3)	0.6 \pm 0.0 (0.6)	1.0	
ch#5v1	p μ MOEA/D	63.2 \pm 5.2 (71.3)	44.5 \pm 2.5 (50.8)	1.4 \pm 0.4 (1.0)	0.6 \pm 0.0 (0.7)	0.7	0.0
	NSGA-II	71.3 \pm 0.0 (71.3)	50.8 \pm 0.0 (50.8)	8.6 \pm 0.3 (8.3)	0.7 \pm 0.0 (0.7)	0.0	
ch#5v2	p μ MOEA/D	73.8 \pm 2.4 (75.9)	39.0 \pm 2.9 (46.1)	1.4 \pm 0.4 (1.0)	0.6 \pm 0.0 (0.7)	0.2	0.0
	NSGA-II	75.9 \pm 0.0 (75.9)	43.0 \pm 1.2 (46.2)	8.6 \pm 0.3 (8.3)	0.7 \pm 0.0 (0.7)	0.0	
ch#5v3	p μ MOEA/D	30.1 \pm 15.1 (65.2)	40.7 \pm 1.9 (44.7)	1.4 \pm 0.4 (1.0)	2.0 \pm 0.1 (2.1)	0.0	0.0
	NSGA-II	69.2 \pm 1.3 (70.4)	45.0 \pm 1.6 (46.1)	8.5 \pm 0.2 (8.3)	2.3 \pm 0.0 (2.4)	0.0	
ch#5v4	p μ MOEA/D	74.8 \pm 4.4 (78.3)	44.0 \pm 1.7 (47.8)	1.5 \pm 0.4 (1.0)	0.9 \pm 0.0 (1.0)	0.3	0.0
	NSGA-II	78.3 \pm 0.0 (78.3)	46.3 \pm 1.0 (49.7)	8.4 \pm 0.2 (8.2)	1.1 \pm 0.0 (1.1)	0.0	
ch#6v1	p μ MOEA/D	20.2 \pm 11.4 (40.5)	32.1 \pm 0.2 (32.4)	1.1 \pm 0.3 (0.8)	0.6 \pm 0.0 (0.7)	0.3	0.0
	NSGA-II	49.9 \pm 0.0 (49.9)	38.8 \pm 0.0 (38.8)	7.8 \pm 0.2 (7.6)	0.7 \pm 0.0 (0.7)	0.0	
ch#6v2	p μ MOEA/D	36.0 \pm 7.0 (49.4)	38.4 \pm 2.3 (44.2)	1.2 \pm 0.3 (0.8)	0.6 \pm 0.0 (0.7)	0.9	0.0
	NSGA-II	49.4 \pm 0.0 (49.4)	43.2 \pm 0.9 (44.2)	8.2 \pm 0.2 (8.0)	0.7 \pm 0.0 (0.7)	0.0	
ch#6v3	p μ MOEA/D	24.6 \pm 5.5 (32.8)	33.4 \pm 2.2 (39.4)	1.3 \pm 0.3 (0.9)	0.7 \pm 0.0 (0.7)	0.2	0.0
	NSGA-II	32.8 \pm 0.1 (32.8)	39.4 \pm 0.0 (39.4)	8.2 \pm 0.2 (7.9)	0.7 \pm 0.0 (0.7)	0.0	
ch#6v4	p μ MOEA/D	20.7 \pm 1.5 (24.5)	35.7 \pm 2.1 (40.6)	1.3 \pm 0.4 (0.8)	0.7 \pm 0.0 (0.8)	0.8	0.0
	NSGA-II	31.7 \pm 0.0 (31.7)	40.6 \pm 0.0 (40.6)	8.2 \pm 0.2 (7.9)	0.8 \pm 0.0 (0.8)	0.0	

Tabla A.13: Tabla comparativa: p μ MOEA/D vs. NSGA-II – Instancias chicas.

		mejora greedy (%)			HV($\times 10^6$)	<i>p</i> valores
		costo	demora	tiempo(s)		S-W K-W
me#1v1	p μ MOEA/D NSGA-II	12.9 \pm 8.9 (36.9) 42.5 \pm 0.1 (42.5)	41.3 \pm 4.5 (48.2) 48.9 \pm 0.6 (49.4)	1.7 \pm 0.4 (1.1) 8.7 \pm 0.3 (8.5)	3.3 \pm 0.1 (3.4) 3.7 \pm 0.0 (3.7)	0.6 0.0
me#1v2	p μ MOEA/D NSGA-II	101.2 \pm 0.0 (101.2) 101.2 \pm 0.0 (101.2)	58.3 \pm 2.6 (62.5) 62.5 \pm 0.0 (62.5)	1.5 \pm 0.4 (1.1) 8.9 \pm 0.2 (8.6)	1.2 \pm 0.0 (1.3) 1.4 \pm 0.0 (1.4)	0.1 0.0
me#1v3	p μ MOEA/D NSGA-II	20.9 \pm 9.5 (37.2) 40.9 \pm 0.0 (40.9)	6.6 \pm 3.2 (17.4) 15.6 \pm 2.5 (17.4)	1.5 \pm 0.4 (1.1) 8.6 \pm 0.3 (8.3)	2.7 \pm 0.2 (3.0) 3.2 \pm 0.0 (3.3)	0.0 0.2
me#1v4	p μ MOEA/D NSGA-II	44.3 \pm 10.6 (56.8) 66.3 \pm 0.4 (66.6)	55.6 \pm 0.5 (56.2) 56.1 \pm 0.2 (56.2)	1.6 \pm 0.4 (1.1) 8.7 \pm 0.2 (8.4)	1.1 \pm 0.0 (1.2) 1.4 \pm 0.0 (1.4)	0.0 0.0
me#2v1	p μ MOEA/D NSGA-II	32.9 \pm 6.9 (56.3) 57.8 \pm 0.8 (58.4)	39.3 \pm 1.8 (42.0) 41.4 \pm 1.6 (44.0)	2.2 \pm 0.6 (1.5) 9.8 \pm 0.2 (9.6)	7.0 \pm 0.3 (7.4) 8.4 \pm 0.1 (8.5)	0.1 0.0
me#2v2	p μ MOEA/D NSGA-II	0.0 \pm 0.0 (0.0) 21.9 \pm 11.3 (38.1)	22.6 \pm 7.1 (33.8) 39.4 \pm 1.0 (40.1)	2.0 \pm 0.5 (1.5) 9.9 \pm 0.3 (9.5)	12.0 \pm 0.4 (12.8) 15.0 \pm 0.1 (15.2)	0.7 0.6
me#2v3	p μ MOEA/D NSGA-II	44.3 \pm 2.1 (50.3) 54.6 \pm 3.0 (59.0)	30.0 \pm 4.4 (38.6) 41.0 \pm 1.0 (42.0)	1.9 \pm 0.4 (1.5) 9.9 \pm 0.2 (9.7)	7.4 \pm 0.2 (7.7) 8.8 \pm 0.1 (9.0)	0.1 0.1
me#2v4	p μ MOEA/D NSGA-II	2.7 \pm 8.2 (27.3) 28.6 \pm 9.5 (40.0)	13.7 \pm 4.4 (22.3) 28.5 \pm 0.4 (29.4)	2.0 \pm 0.6 (1.4) 9.8 \pm 0.2 (9.6)	11.8 \pm 0.4 (12.3) 14.0 \pm 0.1 (14.1)	0.0 0.9
me#3v1	p μ MOEA/D NSGA-II	93.4 \pm 1.9 (97.7) 102.9 \pm 0.4 (103.5)	44.5 \pm 3.0 (49.5) 48.3 \pm 1.1 (50.3)	2.4 \pm 0.8 (1.8) 11.0 \pm 0.2 (10.6)	4.2 \pm 0.2 (4.6) 5.2 \pm 0.0 (5.3)	0.9 0.6
me#3v2	p μ MOEA/D NSGA-II	38.7 \pm 9.5 (59.0) 68.4 \pm 1.5 (71.4)	52.9 \pm 1.9 (57.1) 59.5 \pm 0.7 (60.7)	2.5 \pm 0.8 (1.8) 11.2 \pm 0.2 (10.9)	6.2 \pm 0.2 (6.5) 7.6 \pm 0.1 (7.7)	0.8 0.4
me#3v3	p μ MOEA/D NSGA-II	17.8 \pm 12.4 (38.9) 62.9 \pm 3.3 (68.7)	56.9 \pm 0.6 (58.4) 59.3 \pm 0.9 (60.8)	2.4 \pm 0.6 (1.7) 11.0 \pm 0.4 (10.7)	5.6 \pm 0.2 (5.9) 7.2 \pm 0.1 (7.4)	0.1 0.9
me#3v4	p μ MOEA/D NSGA-II	29.6 \pm 13.1 (51.7) 59.4 \pm 1.8 (62.1)	65.9 \pm 0.3 (66.9) 66.4 \pm 0.5 (67.4)	2.4 \pm 0.7 (1.7) 11.0 \pm 0.4 (10.6)	13.7 \pm 0.4 (14.4) 16.2 \pm 0.1 (16.3)	0.5 0.4
me#4v1	p μ MOEA/D NSGA-II	63.0 \pm 6.8 (75.0) 78.8 \pm 0.2 (78.9)	58.8 \pm 0.9 (61.1) 61.6 \pm 0.4 (62.6)	2.0 \pm 0.5 (1.4) 9.7 \pm 0.3 (9.3)	1.8 \pm 0.1 (2.0) 2.2 \pm 0.0 (2.3)	0.2 0.5
me#4v2	p μ MOEA/D NSGA-II	1.7 \pm 2.1 (6.0) 45.9 \pm 9.1 (52.4)	37.1 \pm 4.1 (45.5) 45.8 \pm 0.3 (46.4)	2.0 \pm 0.6 (1.3) 9.7 \pm 0.2 (9.4)	6.1 \pm 0.2 (6.3) 7.0 \pm 0.1 (7.1)	0.3 0.0
me#4v3	p μ MOEA/D NSGA-II	61.1 \pm 9.5 (73.5) 77.5 \pm 0.8 (78.9)	58.4 \pm 1.9 (60.8) 60.9 \pm 0.6 (61.9)	2.2 \pm 0.6 (1.4) 9.5 \pm 0.3 (9.3)	1.5 \pm 0.1 (1.8) 2.0 \pm 0.0 (2.0)	0.0 0.3
me#4v4	p μ MOEA/D NSGA-II	59.4 \pm 7.2 (73.8) 74.8 \pm 0.1 (74.9)	51.5 \pm 1.1 (54.8) 54.9 \pm 0.3 (55.6)	1.9 \pm 0.5 (1.4) 9.7 \pm 0.2 (9.5)	2.1 \pm 0.1 (2.2) 2.5 \pm 0.0 (2.5)	0.0 0.3
me#5v1	p μ MOEA/D NSGA-II	37.4 \pm 8.2 (49.4) 55.4 \pm 2.2 (57.6)	56.2 \pm 1.6 (59.0) 58.5 \pm 0.8 (60.3)	2.7 \pm 0.8 (1.8) 11.2 \pm 0.2 (10.9)	5.2 \pm 0.3 (5.8) 7.1 \pm 0.0 (7.2)	1.0 0.2
me#5v2	p μ MOEA/D NSGA-II	40.9 \pm 8.1 (51.8) 56.1 \pm 2.4 (58.6)	49.3 \pm 2.9 (52.6) 52.8 \pm 0.5 (54.4)	2.6 \pm 0.9 (1.8) 11.1 \pm 0.2 (10.8)	6.5 \pm 0.2 (6.9) 8.5 \pm 0.0 (8.6)	0.5 0.0
me#5v3	p μ MOEA/D NSGA-II	29.2 \pm 19.3 (63.2) 69.9 \pm 2.0 (73.3)	38.5 \pm 7.4 (48.7) 55.2 \pm 1.0 (56.5)	2.5 \pm 0.7 (1.8) 11.1 \pm 0.3 (10.8)	18.1 \pm 0.6 (19.3) 22.9 \pm 0.2 (23.2)	0.0 0.3
me#5v4	p μ MOEA/D NSGA-II	20.8 \pm 5.6 (39.1) 57.5 \pm 2.9 (61.0)	52.9 \pm 0.5 (53.5) 54.0 \pm 0.8 (55.7)	2.5 \pm 0.7 (1.9) 11.1 \pm 0.2 (10.8)	8.5 \pm 0.5 (9.6) 12.1 \pm 0.1 (12.3)	0.0 0.6
me#6v1	p μ MOEA/D NSGA-II	34.8 \pm 9.4 (53.9) 60.8 \pm 0.9 (61.9)	59.8 \pm 1.6 (62.8) 62.4 \pm 1.1 (64.3)	2.2 \pm 0.7 (1.5) 10.1 \pm 0.3 (9.8)	3.0 \pm 0.3 (3.5) 4.2 \pm 0.0 (4.2)	0.3 0.1
me#6v2	p μ MOEA/D NSGA-II	15.6 \pm 12.9 (49.1) 56.3 \pm 1.2 (57.3)	41.1 \pm 2.9 (47.9) 50.5 \pm 0.8 (50.9)	2.1 \pm 0.6 (1.5) 10.1 \pm 0.2 (9.8)	5.2 \pm 0.1 (5.5) 6.1 \pm 0.0 (6.2)	0.4 0.0
me#6v3	p μ MOEA/D NSGA-II	11.6 \pm 10.5 (33.6) 45.6 \pm 4.9 (55.5)	48.2 \pm 0.8 (50.0) 52.3 \pm 1.0 (54.2)	2.2 \pm 0.6 (1.5) 10.1 \pm 0.3 (9.9)	2.7 \pm 0.1 (2.9) 3.4 \pm 0.0 (3.5)	0.0 0.9
me#6v4	p μ MOEA/D NSGA-II	17.8 \pm 9.0 (31.6) 46.7 \pm 4.7 (59.5)	36.8 \pm 1.4 (41.6) 40.9 \pm 0.9 (42.3)	2.1 \pm 0.6 (1.6) 10.1 \pm 0.2 (9.9)	1.6 \pm 0.2 (1.8) 2.3 \pm 0.0 (2.4)	0.0 0.1

Tabla A.14: Tabla comparativa: p μ MOEA/D vs. NSGA-II – Instancias medianas.

		mejora greedy (%)			HV($\times 10^6$)	<i>p</i> valores
		costo	demora	tiempo(s)		S-W
gr#1v1	p μ MOEA/D NSGA-II	1.7 \pm 1.4 (8.2) 24.3 \pm 12.9 (49.0)	45.2 \pm 3.9 (49.5) 52.9 \pm 0.8 (54.8)	2.4 \pm 0.5 (2.1) 12.2 \pm 0.3 (11.9)	7.3 \pm 0.2 (7.8) 9.3 \pm 0.1 (9.5)	0.9 0.3
gr#1v2	p μ MOEA/D NSGA-II	0.0 \pm 0.0 (0.0) 33.6 \pm 17.2 (55.4)	43.7 \pm 9.9 (54.9) 58.7 \pm 1.1 (60.0)	3.1 \pm 0.9 (2.1) 12.5 \pm 0.4 (11.9)	12.2 \pm 0.4 (12.8) 15.0 \pm 0.2 (15.4)	0.0 0.0
gr#1v3	p μ MOEA/D NSGA-II	41.3 \pm 6.7 (58.0) 75.9 \pm 2.2 (78.3)	50.9 \pm 1.2 (54.9) 55.7 \pm 1.5 (59.5)	2.6 \pm 0.7 (2.1) 12.4 \pm 0.3 (12.0)	4.3 \pm 0.3 (4.8) 6.2 \pm 0.1 (6.3)	0.1 0.7
gr#1v4	p μ MOEA/D NSGA-II	11.1 \pm 0.2 (11.3) 28.3 \pm 14.6 (50.8)	53.9 \pm 0.1 (54.1) 54.7 \pm 0.7 (56.6)	3.1 \pm 0.9 (2.2) 12.5 \pm 0.5 (12.0)	1.9 \pm 0.4 (2.4) 3.3 \pm 0.0 (3.3)	0.0 0.2
gr#2v1	p μ MOEA/D NSGA-II	28.6 \pm 4.2 (37.6) 58.1 \pm 3.0 (62.7)	39.7 \pm 6.0 (49.2) 53.7 \pm 0.8 (55.2)	2.9 \pm 0.9 (2.0) 11.8 \pm 0.2 (11.6)	8.9 \pm 0.5 (9.8) 12.2 \pm 0.1 (12.4)	0.4 0.5
gr#2v2	p μ MOEA/D NSGA-II	54.3 \pm 7.8 (70.6) 76.4 \pm 2.5 (79.5)	58.9 \pm 1.9 (60.9) 60.3 \pm 1.2 (62.4)	2.8 \pm 0.7 (2.1) 12.1 \pm 0.2 (11.8)	38.2 \pm 0.7 (39.2) 42.4 \pm 0.4 (43.1)	0.2 0.3
gr#2v3	p μ MOEA/D NSGA-II	8.1 \pm 1.9 (13.2) 45.4 \pm 7.7 (56.7)	19.1 \pm 3.6 (25.5) 39.4 \pm 1.4 (41.8)	2.6 \pm 0.7 (1.9) 12.4 \pm 0.3 (12.2)	19.0 \pm 0.8 (20.4) 23.3 \pm 0.2 (23.7)	0.3 0.7
gr#2v4	p μ MOEA/D NSGA-II	26.6 \pm 5.5 (44.1) 60.9 \pm 5.5 (67.5)	31.1 \pm 4.7 (39.1) 48.4 \pm 1.3 (50.0)	2.8 \pm 0.8 (2.1) 12.1 \pm 0.2 (11.9)	25.1 \pm 0.6 (26.1) 29.1 \pm 0.3 (29.5)	0.0 0.0
gr#3v1	p μ MOEA/D NSGA-II	34.5 \pm 6.3 (52.6) 66.7 \pm 3.5 (69.1)	61.4 \pm 0.5 (62.8) 63.1 \pm 0.8 (65.4)	2.9 \pm 0.8 (2.1) 11.8 \pm 0.3 (11.6)	3.0 \pm 0.3 (3.5) 4.4 \pm 0.0 (4.4)	0.1 0.0
gr#3v2	p μ MOEA/D NSGA-II	24.9 \pm 12.4 (56.2) 70.5 \pm 1.0 (72.0)	57.8 \pm 2.2 (62.0) 63.7 \pm 1.0 (65.1)	2.9 \pm 0.8 (2.1) 12.1 \pm 0.3 (11.8)	1.6 \pm 0.2 (1.8) 2.5 \pm 0.0 (2.5)	0.2 0.0
gr#3v3	p μ MOEA/D NSGA-II	24.6 \pm 13.3 (57.2) 73.9 \pm 1.0 (75.2)	62.4 \pm 0.5 (63.7) 64.2 \pm 0.9 (65.4)	2.7 \pm 0.6 (2.1) 12.1 \pm 0.3 (11.7)	1.8 \pm 0.2 (2.1) 2.8 \pm 0.0 (2.9)	0.3 0.2
gr#3v4	p μ MOEA/D NSGA-II	37.6 \pm 7.3 (58.1) 68.2 \pm 1.5 (70.0)	57.2 \pm 2.0 (59.8) 60.8 \pm 0.7 (61.5)	2.6 \pm 0.7 (2.1) 12.0 \pm 0.2 (11.7)	3.6 \pm 0.1 (3.9) 4.6 \pm 0.0 (4.6)	0.6 0.1
gr#4v1	p μ MOEA/D NSGA-II	61.2 \pm 9.6 (81.4) 102.3 \pm 1.4 (105.2)	65.2 \pm 1.4 (67.6) 68.8 \pm 1.2 (71.2)	3.4 \pm 0.9 (2.6) 13.7 \pm 0.2 (13.4)	2.5 \pm 0.4 (3.2) 4.8 \pm 0.1 (4.9)	0.1 0.9
gr#4v2	p μ MOEA/D NSGA-II	44.9 \pm 10.1 (62.2) 88.1 \pm 3.5 (93.7)	62.6 \pm 1.1 (65.7) 67.9 \pm 1.2 (69.8)	3.6 \pm 1.1 (2.4) 13.3 \pm 0.3 (13.0)	5.7 \pm 0.5 (6.5) 8.8 \pm 0.2 (9.1)	0.1 0.1
gr#4v3	p μ MOEA/D NSGA-II	45.2 \pm 11.8 (68.6) 90.3 \pm 3.1 (94.7)	66.9 \pm 1.5 (69.4) 70.8 \pm 1.1 (72.8)	3.5 \pm 1.0 (2.4) 13.7 \pm 0.2 (13.3)	6.7 \pm 0.6 (8.0) 10.2 \pm 0.1 (10.5)	0.0 0.2
gr#4v4	p μ MOEA/D NSGA-II	43.0 \pm 9.5 (66.3) 92.1 \pm 2.6 (95.5)	61.9 \pm 1.0 (64.2) 67.6 \pm 1.3 (70.0)	3.4 \pm 1.0 (2.4) 13.5 \pm 0.3 (13.1)	6.5 \pm 0.5 (7.3) 9.8 \pm 0.2 (10.1)	0.0 0.4
gr#5v1	p μ MOEA/D NSGA-II	10.7 \pm 3.2 (14.7) 29.5 \pm 5.2 (42.3)	71.2 \pm 0.3 (71.6) 71.8 \pm 0.6 (73.0)	4.9 \pm 1.4 (3.5) 17.1 \pm 0.3 (16.7)	9.9 \pm 1.5 (12.1) 16.3 \pm 0.2 (16.5)	0.0 0.1
gr#5v2	p μ MOEA/D NSGA-II	8.1 \pm 1.6 (12.4) 40.8 \pm 3.8 (48.0)	70.2 \pm 0.5 (72.8) 73.0 \pm 0.8 (75.1)	4.9 \pm 1.4 (3.4) 17.1 \pm 0.3 (16.9)	15.3 \pm 1.0 (17.2) 20.9 \pm 0.2 (21.3)	0.9 0.8
gr#5v3	p μ MOEA/D NSGA-II	34.0 \pm 9.1 (56.8) 78.9 \pm 2.6 (83.0)	62.9 \pm 1.7 (66.3) 66.0 \pm 1.1 (68.1)	4.6 \pm 1.3 (3.5) 17.2 \pm 0.2 (17.1)	7.8 \pm 0.9 (9.2) 12.4 \pm 0.1 (12.7)	0.1 1.0
gr#5v4	p μ MOEA/D NSGA-II	44.5 \pm 6.5 (66.3) 74.5 \pm 2.8 (84.2)	60.2 \pm 1.0 (63.0) 63.1 \pm 1.4 (65.6)	5.0 \pm 1.5 (3.5) 17.2 \pm 0.3 (16.7)	2.6 \pm 0.5 (3.4) 5.3 \pm 0.1 (5.5)	0.1 0.4
gr#6v1	p μ MOEA/D NSGA-II	99.5 \pm 0.4 (100.2) 101.2 \pm 0.1 (101.3)	56.9 \pm 3.1 (61.2) 62.3 \pm 1.3 (64.6)	3.8 \pm 1.1 (2.7) 14.2 \pm 0.8 (13.7)	3.8 \pm 0.2 (4.2) 5.7 \pm 0.1 (5.9)	0.2 0.5
gr#6v2	p μ MOEA/D NSGA-II	98.8 \pm 0.4 (99.7) 100.5 \pm 0.1 (100.9)	58.6 \pm 1.8 (62.9) 69.7 \pm 1.8 (72.4)	3.8 \pm 1.1 (2.6) 14.3 \pm 0.7 (13.8)	11.8 \pm 0.3 (12.6) 15.6 \pm 0.3 (16.1)	0.4 0.4
gr#6v3	p μ MOEA/D NSGA-II	8.3 \pm 6.5 (36.0) 71.2 \pm 4.5 (78.3)	57.6 \pm 3.5 (63.7) 70.1 \pm 0.8 (71.4)	3.6 \pm 1.0 (2.7) 14.4 \pm 0.4 (13.6)	17.4 \pm 0.7 (18.3) 22.3 \pm 0.4 (22.9)	0.0 0.7
gr#6v4	p μ MOEA/D NSGA-II	6.5 \pm 3.3 (17.7) 73.0 \pm 4.6 (77.7)	64.7 \pm 2.1 (68.0) 73.0 \pm 0.6 (74.4)	3.5 \pm 1.1 (2.4) 14.4 \pm 0.3 (14.1)	23.2 \pm 0.7 (24.4) 29.1 \pm 0.3 (29.6)	0.2 0.3

Tabla A.15: Tabla comparativa: p μ MOEA/D vs. NSGA-II – Instancias grandes.

		mejora greedy (%)			HV($\times 10^6$)	$p_{valores}$	
		costo	demora	tiempo(s)		S-W	K-W
Mo#1v1	p μ MOEA/D	3.2 \pm 0.0 (3.2)	48.9 \pm 0.0 (48.9)	1.2 \pm 0.3 (0.8)	0.3 \pm 0.0 (0.3)	0.0	0.0
	NSGA-II	3.2 \pm 0.0 (3.2)	50.0 \pm 1.2 (52.1)	7.6 \pm 0.2 (7.3)	0.3 \pm 0.0 (0.3)	1.0	
Mo#1v2	p μ MOEA/D	4.0 \pm 0.0 (4.0)	15.1 \pm 0.0 (15.1)	1.2 \pm 0.3 (0.8)	0.3 \pm 0.0 (0.3)	0.1	0.0
	NSGA-II	4.0 \pm 0.0 (4.0)	15.3 \pm 1.0 (20.5)	7.6 \pm 0.2 (7.4)	0.3 \pm 0.0 (0.3)	1.0	
Mo#1v3	p μ MOEA/D	3.3 \pm 0.0 (3.3)	26.8 \pm 1.5 (27.1)	1.1 \pm 0.3 (0.8)	0.3 \pm 0.0 (0.3)	0.1	0.0
	NSGA-II	3.3 \pm 0.0 (3.3)	27.1 \pm 0.0 (27.1)	7.9 \pm 0.3 (7.6)	0.3 \pm 0.0 (0.3)	1.0	
Mo#1v4	p μ MOEA/D	0.0 \pm 0.0 (0.0)	25.6 \pm 3.6 (32.0)	1.2 \pm 0.3 (0.8)	0.3 \pm 0.0 (0.4)	0.4	0.0
	NSGA-II	0.0 \pm 0.0 (0.0)	32.0 \pm 0.0 (32.0)	8.5 \pm 0.2 (8.2)	0.4 \pm 0.0 (0.4)	1.0	
Mo#2v1	p μ MOEA/D	43.8 \pm 8.7 (64.0)	54.7 \pm 1.9 (58.7)	1.8 \pm 0.5 (1.2)	4.1 \pm 0.1 (4.3)	0.5	0.0
	NSGA-II	64.9 \pm 0.3 (65.0)	62.9 \pm 1.6 (63.5)	9.2 \pm 0.3 (9.0)	4.9 \pm 0.0 (4.9)	0.0	
Mo#2v2	p μ MOEA/D	17.5 \pm 0.0 (17.5)	6.8 \pm 6.5 (18.7)	1.8 \pm 0.5 (1.2)	4.3 \pm 0.1 (4.6)	0.0	0.0
	NSGA-II	17.7 \pm 0.0 (17.7)	23.4 \pm 2.4 (26.5)	9.0 \pm 0.2 (8.9)	4.9 \pm 0.0 (4.9)	0.7	
Mo#2v3	p μ MOEA/D	58.5 \pm 8.6 (73.7)	46.7 \pm 1.7 (52.5)	1.8 \pm 0.5 (1.2)	1.6 \pm 0.1 (1.8)	0.1	0.0
	NSGA-II	76.8 \pm 0.0 (76.8)	48.9 \pm 0.8 (50.0)	9.1 \pm 0.3 (8.8)	2.0 \pm 0.0 (2.0)	0.0	
Mo#2v4	p μ MOEA/D	39.0 \pm 7.1 (52.0)	53.3 \pm 0.4 (54.3)	1.8 \pm 0.5 (1.3)	3.1 \pm 0.1 (3.3)	0.3	0.0
	NSGA-II	57.4 \pm 0.2 (57.5)	58.9 \pm 0.6 (59.6)	9.1 \pm 0.2 (8.9)	3.7 \pm 0.0 (3.8)	0.0	
Mo#3v1	p μ MOEA/D	38.1 \pm 7.2 (51.9)	57.0 \pm 0.9 (58.1)	1.8 \pm 0.5 (1.4)	2.5 \pm 0.1 (2.7)	1.0	0.0
	NSGA-II	51.8 \pm 0.2 (51.9)	58.8 \pm 0.4 (59.0)	9.3 \pm 0.2 (9.1)	2.8 \pm 0.0 (2.8)	0.0	
Mo#3v2	p μ MOEA/D	40.0 \pm 5.6 (55.2)	32.8 \pm 6.8 (47.1)	1.8 \pm 0.5 (1.3)	2.3 \pm 0.1 (2.4)	0.0	0.0
	NSGA-II	55.2 \pm 0.0 (55.2)	48.3 \pm 0.3 (48.6)	9.3 \pm 0.3 (9.0)	2.5 \pm 0.0 (2.5)	0.0	
Mo#3v3	p μ MOEA/D	35.2 \pm 6.3 (51.9)	56.1 \pm 0.7 (57.5)	1.9 \pm 0.6 (1.3)	2.5 \pm 0.0 (2.6)	0.1	0.0
	NSGA-II	51.6 \pm 0.5 (51.9)	57.7 \pm 0.0 (57.7)	9.3 \pm 0.2 (8.9)	2.8 \pm 0.0 (2.8)	0.6	
Mo#3v4	p μ MOEA/D	42.0 \pm 8.6 (55.6)	38.0 \pm 4.3 (44.7)	1.9 \pm 0.5 (1.3)	2.3 \pm 0.0 (2.3)	0.9	0.0
	NSGA-II	59.1 \pm 0.4 (60.1)	48.9 \pm 1.0 (49.4)	9.4 \pm 0.3 (9.1)	2.5 \pm 0.0 (2.5)	0.1	
Mo#4v1	p μ MOEA/D	6.0 \pm 7.9 (46.1)	39.8 \pm 2.0 (45.5)	1.8 \pm 0.5 (1.3)	6.2 \pm 0.1 (6.5)	0.8	0.0
	NSGA-II	54.9 \pm 4.1 (62.9)	46.4 \pm 0.6 (47.9)	9.6 \pm 0.3 (9.2)	7.0 \pm 0.1 (7.1)	0.0	
Mo#4v2	p μ MOEA/D	34.7 \pm 5.0 (47.4)	47.0 \pm 0.5 (49.6)	2.0 \pm 0.6 (1.4)	2.2 \pm 0.1 (2.3)	0.1	0.0
	NSGA-II	48.2 \pm 0.7 (49.1)	47.0 \pm 0.2 (47.2)	10.1 \pm 0.4 (9.6)	2.6 \pm 0.0 (2.6)	0.0	
Mo#4v3	p μ MOEA/D	17.5 \pm 16.1 (58.2)	9.9 \pm 6.2 (25.6)	1.9 \pm 0.6 (1.3)	6.0 \pm 0.1 (6.2)	0.1	0.0
	NSGA-II	60.3 \pm 0.7 (60.7)	28.7 \pm 0.9 (30.4)	9.5 \pm 0.3 (9.2)	6.7 \pm 0.0 (6.7)	0.0	
Mo#4v4	p μ MOEA/D	52.5 \pm 5.9 (67.6)	51.2 \pm 0.4 (51.6)	1.9 \pm 0.5 (1.4)	1.8 \pm 0.1 (1.9)	0.7	0.0
	NSGA-II	74.6 \pm 0.0 (74.6)	51.7 \pm 0.3 (53.1)	9.7 \pm 0.3 (9.4)	2.2 \pm 0.0 (2.2)	0.0	

Tabla A.16: Tabla comparativa: $p\mu$ MOEA/D vs. NSGA-II – Instancias Montevideo.

Apéndice B

Publicaciones

A continuación, se adjuntan cuatro artículos que fueron presentados en distintas conferencias internacionales en el marco de este proyecto. La lista de artículos es la siguiente:

1. *“Online taxi sharing optimization using evolutionary algorithms”* presentado por Gabriel Fagúndez en el Simposio Latinoamericano de Investigación de Operaciones e Inteligencia Artificial de la XL Conferencia Latinoamericana en Informática (CLEI), celebrada del 15 al 19 de setiembre de 2014 en la ciudad de Montevideo, Uruguay. El artículo describe el algoritmo secuencial (*seqEA*) que resuelve la formulación monoobjetivo del problema de viajes compartidos en taxis.
2. *“A parallel micro evolutionary algorithm for taxi sharing optimization”* presentado por Renzo Massobrio en el VIII ALIO/EURO Workshop on Applied Combinatorial Optimization, celebrado entre el 8 y 10 de diciembre de 2014 en la ciudad de Montevideo, Uruguay. El artículo describe el algoritmo paralelo con micro poblaciones distribuidas (*pμEA*) que resuelve la formulación monoobjetivo del problema de viajes compartidos en taxis.
3. *“Planificación multiobjetivo de viajes compartidos en taxis utilizando un micro algoritmo evolutivo paralelo”* presentado por Sergio Nesmachnow en el X Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB) celebrado en las ciudades de Mérida y Almendralejo del 4 al 6 de febrero de 2015. El artículo describe el algoritmo paralelo por descomposición de dominio (*pμMOEA/D*) que resuelve la variante multiobjetivo del problema de viajes compartidos en taxis.
4. *“Multiobjective taxi sharing optimization using the NSGA-II evolutionary algorithm”* presentado por Sergio Nesmachnow en la sesión especial “Metaheuristics for Smart Cities” en la 11th edition of the Metaheuristics International Conference (MIC) celebrada los días 7 a 10 de junio de 2015 en la ciudad de Agadir, Marruecos. El artículo describe el algoritmo evolutivo con enfoque multiobjetivo explícito (*NSGA-II*) que resuelve la variante multiobjetivo del problema de viajes compartidos en taxis.

Online taxi sharing optimization using evolutionary algorithms

Gabriel Fagúndez, Renzo Massobrio y Sergio Nesmachnow
 Centro de Cálculo, Facultad de Ingeniería, Universidad de la República
 Herrera y Reissig 565, 11300 Montevideo, Uruguay
 Email: {gabrielf, renzom, sergion}@fing.edu.uy

Abstract—This article presents the application of evolutionary algorithms for solving the problem of distributing a group of passengers travelling from the same origin to different destinations in several taxis, with the goal of minimizing the total cost of the trips. The experimental analysis compares the quality of the solutions found using the proposed algorithm versus those computed using an intuitive greedy heuristic similar to those previously proposed in the related literature to solve the problem, showing that the evolutionary algorithm is able to reach significant improvements in the trips' total cost, outperforming the greedy heuristic in up to 41.7 % in the best case, and up to 36.4 % on average. Two applications are also presented: a web-based user interface and a mobile application for devices using the iOS operating system.

I. INTRODUCCIÓN

Los viajes compartidos en automóvil, conocidos bajo el término anglosajón de *car pooling*, han tenido gran interés público en los últimos años [1]. Los beneficios de compartir transporte afectan tanto en el plano económico como en el ecológico, a niveles individuales y colectivos. Compartir vehículos entre personas que deben transportarse a destinos cercanos permite minimizar los costos de traslado y también la cantidad de automóviles en las rutas, reduciendo la polución y contribuyendo a minimizar el impacto ambiental del transporte, que es un problema capital en las últimas décadas [2], [3] en las grandes ciudades. Adicionalmente, un menor tráfico se traduce directamente en una menor cantidad de embotellamientos, logrando un tránsito más fluido y por ende mayor eficiencia en los traslados, en especial en horas pico. Desde el punto de vista económico, al compartir el costo del viaje entre los diferentes pasajeros del vehículo, se reduce significativamente los costos asociados. En el caso particular de usuarios que utilizan automóviles con taxímetro, los viajes son más rápidos y por lo tanto más baratos.

Los beneficios de los viajes compartidos han dado lugar a diferentes iniciativas para atender el gran interés del público. Entre ellas se pueden mencionar los carriles exclusivos para viajes compartidos presentes en muchas ciudades, las campañas para compartir los viajes hacia el lugar del trabajo, y una gran variedad de aplicaciones en línea desarrolladas para encontrar compañeros de viaje.

Los automóviles con taxímetro (taxis) constituyen un medio de transporte rápido y confortable. Sin embargo, es frecuente que los taxis no se utilicen a capacidad completa. Por este motivo, los taxis podrían verse beneficiados por los

aspectos aplicables a los viajes compartidos. En la actualidad, existen algunos servicios web que proporcionan soluciones al problema de planificación de viajes compartidos, y algunos trabajos de investigación que han resuelto variantes del problema de planificar viajes en taxis utilizando métodos específicos.

El problema de planificación de viajes compartidos es NP-difícil [4]. Para resolver instancias con dimensiones realistas es necesario utilizar heurísticas o metaheurísticas [5] que permitan calcular soluciones de calidad aceptable en tiempos razonables.

En esta línea de trabajo, este artículo presenta una solución al problema de planificación de viajes compartidos en taxi, donde los usuarios parten desde un mismo origen hacia varios destinos. Se describe un algoritmo evolutivo eficiente que busca una solución que minimice el gasto total de los pasajeros. La validación experimental del algoritmo propuesto se realiza sobre escenarios reales, incluyendo datos actualizados de ubicaciones geográficas y tarifas de taxímetro. Complementariamente, se presenta un sistema que involucra dos aplicaciones para que los usuarios puedan acceder a la resolución de casos reales del problema en línea: una aplicación web y una aplicación móvil para dispositivos con sistema operativo iOS.

El artículo se estructura del modo que se describe a continuación. La siguiente sección presenta el problema de planificación de recorridos compartidos en taxis y su formulación matemática. Una reseña de los principales trabajos relacionados se presenta en la Sección III. La Sección IV introduce a los algoritmos evolutivos, y la Sección V ofrece los detalles de implementación del algoritmo evolutivo propuesto. El análisis experimental se reporta en la Sección VI, incluyendo la comparación con heurísticas tradicionales para la resolución del problema y un análisis de tiempo de ejecución. Las aplicaciones de software desarrolladas (interfaz web y una aplicación móvil) se describen en la Sección VII. Finalmente, la Sección VIII presenta las conclusiones de la investigación y las principales líneas de trabajo futuro.

II. EL PROBLEMA DE VIAJES COMPARTIDOS EN TAXIS

Esta sección presenta la descripción del problema de viajes compartidos en taxis y su formulación matemática como problema de optimización combinatoria.

II-A. Descripción y modelo del problema

El problema a resolver modela la siguiente situación: un determinado número de personas, ubicadas en un mismo lugar

de origen, deciden viajar utilizando taxis hacia diferentes destinos. El problema de optimización relacionado consiste en determinar el número apropiado de taxis y distribuir los pasajeros de forma eficiente, para minimizar el costo global de los recorridos.

La distribución de pasajeros está restringida a la cantidad máxima de personas que pueden viajar en un mismo taxi. Esta constante puede ser fácilmente parametrizable, permitiendo aplicar el problema a diferentes realidades.

El problema fue originalmente concebido para modelar el transporte en taxis en la ciudad de Montevideo, Uruguay, aunque el algoritmo no restringe su aplicabilidad a otros escenarios, tal como se demuestra en el análisis experimental del método de planificación propuesto. Los costos asociados al transporte están modelados con una función realista que considera la tarificación basada en distancias recorridas por cada taxi. Este modelo corresponde a considerar escenarios con tráfico fluido, sin grandes embotellamientos. La incorporación de datos de tráfico en tiempo real es propuesta como una de las principales líneas de trabajo futuro.

Con respecto al problema, las siguientes consideraciones deben ser tenidas en cuenta al instanciar la formulación genérica a un determinado escenario:

- Cada taxi puede trasladar a un número limitado de pasajeros, dependiendo de las reglamentaciones propias de cada ciudad. Este parámetro debe ser ingresado como dato de entrada del algoritmo.
- El número máximo de taxis para N pasajeros es N , en el caso particular de que cada pasajero viaje en un vehículo. Esta solución representa una cota superior trivial para el costo de transporte en un determinado escenario.
- El costo de un taxi está dado por la suma del costo inicial conocido popularmente como "bajada de bandera", más el costo determinado por la distancia recorrida desde el origen al destino. No se consideran otros posibles costos tales como esperas, propinas o cargos extras por equipaje.

II-B. Formulación Matemática

La formulación matemática del problema de viajes compartidos en taxis se presenta a continuación

Dados los siguientes elementos:

- Un conjunto de pasajeros $P = \{p_1, p_2, \dots, p_N\}$; que parten de un punto geográfico común O y desean trasladarse hacia un conjunto de destinos potencialmente diferentes $D = \{d_1, d_2, \dots, d_N\}$;
- Un conjunto de automóviles con taxímetro (taxis) $T = \{t_1, t_2, \dots, t_M\}$; con $M \leq N$; donde la capacidad de cada taxi está acotada por un valor C_{MAX} (número máximo de pasajeros que puede transportar) y cada taxi tiene asociada una función $C : T \rightarrow \{0, 1, \dots, C_{MAX}\}$ que indica cuántos pasajeros hacen uso de un taxi en un determinado viaje.
- Una matriz simétrica M de dimensión $(N+1) \times (N+1)$ que especifica las distancias entre cada uno de los

puntos geográficos involucrados en el problema (un punto origen y N puntos destino);

- Una función de costo del transporte c_T determinada por los parámetros B constante (costo inicial, *bajada de bandera*) y el costo por distancia $c(dist(O, d_i))$.

El problema consiste en hallar una planificación, es decir una función $f : P \rightarrow T$ para transportar los N pasajeros en K taxis ($K \leq M$) que determine la asignación de pasajeros a taxis y el orden en que serán trasladados a los respectivos destinos, minimizando la función de costo total (CT) de la asignación, dada en la Ecuación 1.

$$CT = \sum_{t_i} (B + \sum_{j=1}^{C(t_i)} c(dist(d_{j-1}, d_j))) \quad (1)$$

En la formulación presentada previamente se busca optimizar costo global del viaje completo. Diversas estrategias pueden aplicarse para calcular el costo por pasajero [6], por ejemplo: i) pagar por distancia y repartir equitativamente el costo de bajada de bandera, y cada pasajero j participante en un viaje compartido en el taxi t_i deberá pagar un valor de $B/C(t_i) + dist(d_{j-1}, d_j)$ por el transporte; ii) pagar una tarifa plana independientemente de las distancias, y cada pasajero en el taxi t_i deberá pagar $(B + \sum_{j=1}^{C(t_i)} c(dist(d_{j-1}, d_j))/C(t_i))$, etc.

El problema de planificación de viajes compartidos es NP-difícil, al ser una variante del problema de *car pooling* [4]. Por este motivo, para resolver instancias con dimensiones realistas en tiempos reducidos es necesario utilizar heurísticas o metaheurísticas [5].

III. TRABAJOS RELACIONADOS

Esta sección introduce algunos trabajos relacionados al problema propuesto. Tal como se referencia en la introducción, el problema del *car pooling* ha tenido gran interés público en los últimos años. Los beneficios que presenta han motivado diversas investigaciones en el área que aplican diversas metodologías y heurísticas.

III-A. Taxi Pooling

El problema de *taxi pooling*, una variante particular del *car pooling*, ha sido estudiado desde diferentes perspectivas en la literatura del área. Desde el punto de vista de la compañía de taxis, Xin et al. [7] estudiaron el problema de los *k*-taxis, que propone optimizar el costo total de k taxis que atienden pedidos de clientes en línea. La estrategia propuesta consiste en indicar a los dos taxis más cercanos a un determinado pedido que se dirijan hacia el mismo para intentar atenderlo. Debido a esta competencia entre taxis, se evita el problema donde un taxi siempre está ocupado mientras otros se encuentran inactivos. Los resultados muestran que el problema de los *k*-taxis, como generalización del problema de *k*-servidores [8], alcanza un cociente de k entre su desempeño y el de un algoritmo óptimo que conoce toda la secuencia de pedidos de antemano.

Tratando de balancear los intereses de las compañías y los usuarios, Ma et al. [9] diseñaron un sistema dinámico

para compartir taxis, combinando un algoritmo de búsqueda para encontrar taxis candidatos a satisfacer un pedido, y un algoritmo de planificación que inserta el viaje en el itinerario del taxi que incurre en la menor distancia adicional al atender ese pedido. El algoritmo de planificación chequea cada inserción posible para un nuevo viaje, verifica que satisfaga las restricciones temporales impuestas, y elige para insertar aquél con menor incremento de distancia asociada. Para mejorar el desempeño computacional del algoritmo se describe una estrategia perezosa que utiliza resultados previamente calculados, de forma de postergar lo más posible los cálculos de caminos más cortos hasta que sean necesarios. Usando una base de datos de trayectorias de GPS generadas por 33.000 taxis durante 3 meses en la ciudad de Beijing, el sistema descripto alcanzó un ahorro en la distancia de un 13 % respecto a los viajes individuales, además de atender un 25 % más de pedidos, en un escenario simulado donde la proporción de pedidos contra taxis era de 6 a 1.

Más relacionado con el problema que se presenta en este artículo, que se enfoca principalmente en los intereses del cliente, Tao et al. [10] propusieron dos heurísticas ávidas para optimizar los costos en viajes compartidos de taxis. Un algoritmo se aplica para el caso de un origen a muchos destinos y el otro para el escenario opuesto. Los resultados de una prueba piloto realizada en Taipei con 10 taxis y 798 pasajeros muestran en promedio un porcentaje de éxito en los emparejamientos de un 60.3 %. Sin embargo, los resultados reportados sobre la mejora en el consumo de combustible son en términos absolutos, no en porcentaje, por lo que es difícil extraer información útil para comparar. El problema de un origen a muchos destinos es el que se presenta en este artículo, por lo que es de particular interés comparar el desempeño del algoritmo evolutivo propuesto frente a la técnica ávida.

El análisis de trabajo relacionado muestra que el problema de viajes compartidos en taxis es de interés en la actualidad, debido a que impacta directamente tanto en el medioambiente como en la economía. Sin embargo, las soluciones centradas en el usuario son escasas en la literatura, por lo que hay lugar para el desarrollo y mejora de aplicaciones enfocadas en maximizar el ahorro de los clientes y reducir el tráfico de vehículos a través de los viajes compartidos en taxis.

III-B. Planificación en línea de vehículos compartidos

Mas allá de las investigaciones académicas en el área, existen en la actualidad diversas aplicaciones interactivas que presentan de forma amigable una solución al problema del *car pooling*. Entre las mas conocidas se encuentra la disponible en el sitio web Carpooling (<http://carpooling.com>). Esta solución permite encontrar compañeros de viajes para compartir gastos y ahorrar dinero. El servicio de Carpooling se orienta a personas que desean compartir autos propios, y no taxis, como en nuestra investigación. De todas formas, existen analogías importantes con nuestra propuesta, como la limitación en la cantidad de pasajeros por vehículo.

En cuanto al *taxis pooling*, existen también soluciones similares para usuarios finales en línea. Una de ellas es la disponible en el sitio web Carpling (<http://carpling.com>). La idea detrás de este servicio es similar a la de Carpooling, pero con un enfoque mayoritario en los taxis. A grandes rasgos,

busca encontrar personas que desean viajar a lugares cercanos en taxi, y compartir gastos. Esta solución en línea se diferencia de la que se presenta en este artículo, ya que se limita a insertar viajes que estén completamente incluidos en otros, sin explorar otras posibles soluciones que de todas formas minimizan el costo total. Adicionalmente, permite evaluar a los usuarios, formando un ranking de los usuarios más activos y confiables.

En cuanto al problema presentado en este informe, no existen en la actualidad servicios en línea similares. Existen aplicaciones web con ciertos puntos en común, aunque no con las particularidades de la presente investigación.

IV. ALGORITMOS EVOLUTIVOS

Los Algoritmos Evolutivos (AE) son técnicas estocásticas que emulan el proceso de evolución natural de las especies para resolver problemas de optimización, búsqueda y aprendizaje [11]. En los últimos 30 años, los AE han sido exitosamente aplicados para resolver problemas de optimización subyacentes a problemas complejos del mundo real en múltiples áreas de aplicación [5].

Un AE es una técnica iterativa (cada iteración se denomina *generación*) basada en emular el proceso de evolución natural de los seres vivos. En cada generación se aplican operadores estocásticos sobre un conjunto de individuos (la *población*). Inicialmente, la población se genera aplicando un procedimiento aleatorio o utilizando una heurística específica para el problema a resolver. Cada individuo en la población codifica una solución tentativa al problema estudiado, y tiene un valor de *fitness*, que es una medida relacionada con la función objetivo del problema de optimización, dado por una función de evaluación que determina su adecuación para resolver el problema. El objetivo del AE es mejorar el fitness de los individuos en la población. Este objetivo se logra mediante la aplicación iterativa de *operadores evolutivos*, como la *recombinación* de partes de dos individuos y la *mutación* aleatoria de su codificación, que son aplicados a individuos seleccionados según su fitness, guiando al AE a soluciones tentativas de mayor calidad. El Algoritmo 1 presenta el esquema genérico de un AE que trabaja sobre una población P .

Algorithm 1 Esquema de un Algoritmo Evolutivo.

```

1: inicializar( $P(0)$ )
2:  $t \leftarrow 0$  {contador de generación}
3: mientras no se cumpla el criterio de parada hacer
4:   evaluar( $P(t)$ )
5:   padres  $\leftarrow$  selección( $P(t)$ )
6:   hijos  $\leftarrow$  operadores de variación(padres)
7:   nueva población  $\leftarrow$  reemplazo(hijos,  $P(t)$ )
8:    $t++$ 
9:    $P(t) \leftarrow$  nueva población
10: fin mientras
11: retornar mejor individuo hallado

```

El criterio de parada de la iteración usualmente involucra un número determinado de generaciones, una cota de calidad sobre el mejor valor de fitness hallado, o la detección de una situación de convergencia. Políticas específicas se utilizan para seleccionar el grupo de individuos para participar en la recombinación (la *selección*) y para determinar cuáles nuevos

individuos serán insertados en la población en cada nueva generación (el *reemplazo*). Finalmente, el AE retorna la mejor solución hallada en el proceso iterativo, tomando en cuenta la función de fitness considerada para el problema.

Los detalles de diseño e implementación del AE desarrollado en este trabajo para resolver el problema de planificación de viajes compartidos de taxis se presentan en la Sección V.

V. DISEÑO E IMPLEMENTACIÓN DEL AE PROUESTO

Esta sección presenta los detalles de diseño e implementación del AE.

V-A. La biblioteca Mallba

El AE propuesto para el problema de planificación de viajes compartidos en taxis fue desarrollado utilizando Malva [12], una implementación actualizada de la biblioteca Mallba [13].

Mallba es una biblioteca para optimización combinatoria desarrollada en C++ que incluye métodos exactos, técnicas heurísticas y metaheurísticas. Todos los algoritmos en Mallba son implementados en base a *esqueletos*, similares a un patrón genérico de nombre *strategy*. Un *esqueleto* es un algoritmo que implementa un algoritmo de optimización (por ejemplo, AE) en forma de *template*, que debe ser completado para aplicarse a un problema concreto.

El diseño de un esqueleto se basa en la separación de dos conceptos: las *características del problema* (incluyendo la función a optimizar y detalles sobre cómo manipular las soluciones tentativas) que son proporcionadas por el usuario, y las *técnicas de resolución*, que son provistas de modo abstracto por la biblioteca. Los esqueletos se implementan como un conjunto de clases requeridas y clases provistas:

- Las *clases provistas* implementan aspectos internos de los algoritmos, de modo independiente a los problemas a resolver, en los archivos con extensión *.pro.cc*. Las clases provistas más importantes son *Solver* (el algoritmo) y *SetUpParams* (para fijar los parámetros del método de resolución).
- Las *clases requeridas* especifican información relacionada al problema, implementadas en los archivos con extensión *.req.cc*. Cada esqueleto incluye las clases requeridas *Problem* y *Solution* que encapsulan las entidades necesarias para resolver el problema. Otras clases pueden ser requeridas dependiendo del método de resolución empleado.

El AE presentado en este trabajo fue desarrollado extendiendo el esqueleto *newGA* de Mallba.

V-B. Representación de soluciones

Las soluciones del problema son representadas como tuplas de largo $2N-1$ donde N es el número de pasajeros de la instancia del problema a resolver. Las tuplas contienen los enteros del 1 al N , que representan a cada uno de los pasajeros a ser distribuidos en cada uno de los taxis, y $N-1$ separadores, para distinguir grupos de pasajeros asignados a diferentes taxis. En nuestro caso en particular, los separadores son *ceros*, aunque potencialmente podrían utilizarse cualquier carácter

que no sea un número natural. Cada grupo de dígitos distintos de cero representan por lo tanto a un taxi, que transportará a su destino a cada uno de los pasajeros, respetando el orden en que aparecen en la secuencia de izquierda a derecha. La Figura 1 muestra un ejemplo de representación de solución para una instancia con $N = 5$.

En este caso en particular, poseemos la siguiente representación: **0 3 1 0 5 0 2 4 0**. La misma, tal como puede apreciarse en la imagen referenciada, se traduce en un taxi que se trasladará del origen, en primer lugar, hacia el destino del pasajero 3, y luego al destino del pasajero 1. Luego, otro taxi partirá del origen, directamente al destino 5. Apreciamos como el 0 entre ambos taxis actúa de separador. Finalmente, un tercer vehículo saldrá desde el origen, hacia el destino 2 y finalizará en el destino 4. Un detalle importante es que todos los ceros que se encuentren en el inicio de la secuencia, así como aquellos que se encuentren al final de la misma, serán ignorados.

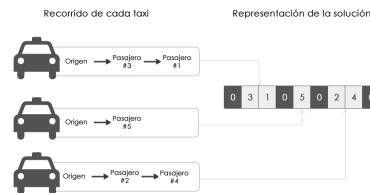


Figura 1. Ejemplo de representación de una solución para el problema de planificación de viajes compartidos en taxis.

Dada la representación elegida y la realidad del problema, surgen las siguientes restricciones para cada tupla:

- El número de dígitos consecutivos mayores que cero está limitado por la máxima cantidad de pasajeros permitidos por taxi.
- Cada número mayor que cero debe aparecer una y sólo una vez, para asegurar que cada pasajero está asignado a un único taxi.
- Finalmente, se observa que dos o más ceros consecutivos cumplen el mismo rol que un único cero.

V-C. Generación de la población inicial

La población inicial del AE es generada mediante un algoritmo constructivo descrito en el Algoritmo 2.

Algoritmo 2 Inicialización aleatoria de las soluciones del AE

```

desde j = 1 a 2N-1 hacer
    sol[j] = 0; {inicializar solución con ceros}
fin desde
desde i = 1 a N hacer
    repetir
        posicion = aleatorio(2N-1);
        hasta (sol[posicion]==0) {sortear una posición libre}
        sol[posicion] = i;
    fin desde
    retornar sol;
```

La generación aleatoria aplicando el Algoritmo 2 puede violar alguna de las restricciones impuestas por el mecanismo de representación. Para resolver este inconveniente se aplica un procedimiento de corrección a las soluciones generadas (Algoritmo 3), desplazando ceros consecutivos de forma de romper, en un punto aleatorio, las secuencias de dígitos mayores que cero que no cumplen con el largo máximo permitido.

Algorithm 3 Procedimiento de corrección de soluciones

```

seguir = verdadero;
mientras seguir hacer
    contador = 0;
    i = 0;
    mientras contador<= CMAX and i < N hacer
        si sol[i] ≠ 0 entonces
            contador++;
        en otro caso
            contador = 0;
        fin si
        i++;
    fin mientras
    si cantidad> CMAX entonces
        m = random((i-CMAX)+1,i-1);
        p = encontrarPrimerCeroConsecutivo();
        intercambiar(p,m);
    en otro caso
        seguir=falso;
    fin si
fin mientras

```

V-D. Función de fitness

El objetivo del problema es minimizar el gasto total de los pasajeros, las soluciones con menor costo total serán las más adecuadas para resolver el problema. Para transformar el problema de minimización de costo en un problema de maximización de fitness, se utiliza el inverso de la función de costo total de una solución, tal como fue presentada en la Ecuación 1.

V-E. Operadores

Debido a las restricciones en la representación explicadas en la Sección V-B, los operadores usados por el AE deben ser adaptados para aplicarse al problema.

1) *Selección*: Se utilizó el operador de *selección proporcional*, que asigna a cada individuo una probabilidad de selección dada por el cociente entre su valor de fitness y la suma del fitness de los individuos en la población (Ecuación 2)

$$P_i = \frac{\text{fitness}(i)}{\sum_{j=1}^N \text{fitness}(j)} \quad (2)$$

2) *Recombinación*: Se utilizó una versión modificada del operador de *Cruzamiento Basado en la Posición (Position Based Crossover - PBX)* (Algoritmo 4).

Para evitar violar las restricciones de codificación, se aplica el procedimiento de corrección al completar el cruzamiento PBX, de forma de asegurar que se cumplan las restricciones impuestas. La Figura 2 muestra un ejemplo del PBX para soluciones con 5 pasajeros y un máximo de 4 por taxi.

Algorithm 4 Cruzamiento Basado en la Posición - PBX

- 1: Seleccionar aleatoriamente varias posiciones del padre 1.
- 2: generar parcialmente el hijo 1, copiando los valores (*alelos*) de las posiciones elegidas del padre 1.
- 3: Marcar los alelos del padre 2 que ya fueron seleccionados en el padre 1.
- 4: Desde el inicio del padre 2, seleccionar secuencialmente el siguiente alelo que no haya sido marcado, y ponerlo en la primer posición libre del hijo 1, desde el comienzo.

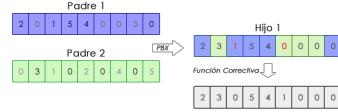


Figura 2. Ejemplo de cruzamiento PBX aplicado al problema.

3) *Mutación*: Se utilizó el operador de *Mutación por Intercambio (Exchange Mutation - EM)*. Es un operador muy simple de implementar, que sorteaa aleatoriamente dos posiciones en la solución y las intercambia. De ser necesario, el procedimiento de corrección se aplica posteriormente a la EM, para corregir posibles conflictos en las soluciones mutadas.

VI. RESULTADOS EXPERIMENTALES

Esta sección reporta el análisis experimental del algoritmo evolutivo propuesto sobre un conjunto de instancias realistas del problema.

VI-A. Plataforma de desarrollo y ejecución

El AE se desarrolló en C++, utilizando la biblioteca Mallba y el compilador g++ versión 4.7.2 (Ubuntu/Linaro 4.7.2-2ubuntu1). El análisis experimental se llevó a cabo en un equipo con procesador Intel Core i5, CPU M 430 a 2.27GHz, 4GB de memoria RAM, y sistema operativo Ubuntu Linux 12.04 de 64 bits.

VI-B. Instancias del problema

Para evaluar el AE propuesto, se generó un conjunto de instancias realistas del problema de planificación de viajes compartidos en taxis. Se utilizó un enfoque metodológico específico para la generación de instancias realistas del problema, siguiendo los lineamientos presentados en los trabajos relacionados y utilizando servicios disponibles para recabar información sobre pedidos de taxi, mapas, y tarifas, incluyendo:

- *Generador de Pedidos de Taxis (Taxi Query Generator-TQG)*, una herramienta que utiliza información de una base de datos de trayectorias de taxis, obtenidas a través de los dispositivos GPS instalados durante una semana en 10.357 taxis en la ciudad de Beijing, para generar pedidos de taxis realistas. Los datos son una muestra de los utilizados por Ma *et al.* [9] tal como se mencionó en la Sección III. TQG genera una lista con las coordenadas de origen y destino de viajes individuales. Para obtener instancias aplicables al problema presentado en este artículo, se

implementó un script que agrupa viajes que comienzan en orígenes cercanos obtenidos a través del TQG, y devuelve instancias del problema de un origen a muchos destinos.

- *QGIS Desktop*, un sistema de información geográfica gratuito y de código abierto, utilizado para crear, editar, visualizar y publicar material geoespacial [14]. QGIS se utilizó para transformar la lista de coordenadas obtenidas a la salida del script de agrupación en un archivo KML (Keyhole Markup Language), que permite ser visualizado utilizando Google Maps/Google Earth [15], de forma de tener una representación visual del origen y destinos de la instancia en un mapa.
- La interfaz para aplicaciones de *TaxiFareFinder* [16], que fue utilizada para obtener la matriz de costos de cada instancia del problema generada, junto a los precios correspondientes a la bajada de bandera. Cada par de coordenadas de una determinada instancia del problema es enviado a la interfaz de *TaxiFareFinder* para obtener el costo entre cada punto.

Siguiendo el enfoque propuesto, se diseñó un benchmark compuesto por seis instancias del problema, utilizando información realista.

VI-C. Ajuste paramétrico

Debido a la cualidad estocástica de los AEs, es necesario ajustar sus parámetros previamente al análisis experimental. Para los experimentos de configuración paramétrica se utilizaron tres instancias del problema con diferentes características, fijando el máximo número de pasajeros permitidos en un mismo taxi en 4. El ajuste se focalizó en estudiar tres parámetros del AE: tamaño de la población ($\#P$), probabilidad de recombinación (pc) y probabilidad de mutación (pm). Para cada parámetro fueron estudiados los siguientes valores candidatos: $\#P$: 150, 200, 250; pc : 0.6, 0.75, 0.95; y pm : 0.001, 0.01, 0.1.

Se estudiaron todas las combinaciones de valores sobre las tres instancias de calibración, realizando 20 ejecuciones independientes del AE en cada caso, con 2000 generaciones en cada ejecución, para resolver cada problema.

Los resultados se muestran gráficamente en la Figura 3, reportando el costo promedio obtenido en las tres instancias para cada una de las combinaciones de parámetros. Las tablas de resultados completos se reportan en <http://www.fing.edu.uy/ino/grupos/cecal/hpc/AG-Taxi>.

Recordando que se trata de un problema de minimización, aquellas configuraciones de parámetros que alcancen valores menores de costo serán preferidas. Para los 3 tests se observa que los mejores resultados se obtienen con el máximo tamaño de población (250), la mínima probabilidad de recombinación (0.6), y la máxima probabilidad de mutación (0.1).

VI-D. Evaluación experimental

El análisis experimental se enfocó en evaluar el desempeño del AE y la calidad de las soluciones alcanzadas. Se utilizaron seis instancias del problema (diferentes a las utilizadas en los experimentos de calibración, para evitar sesgos en los resultados), que corresponden a escenarios de entre 25 y 50

pasajeros. Estas dimensiones son razonablemente mayores a lo que se puede esperar en instancias generadas por usuarios finales, evaluando el AE sobre instancias complejas.

1) Resultados numéricos: La Tabla I reporta los resultados del AE para las seis instancias del problema estudiadas. Las ejecuciones fueron realizadas con la configuración paramétrica determinada en los experimentos de calibración.

Debido a que la aplicación tiene un fuerte enfoque hacia el usuario final, es importante mantener un balance entre la calidad de las soluciones y el tiempo de respuesta. De acuerdo al trabajo de Cantú-Paz y Goldberg [17], generalmente es preferible una única ejecución con un número mayor de generaciones, frente a múltiples ejecuciones más pequeñas. En consecuencia, el AE fue evaluado usando diferentes criterios de parada en términos del número de generaciones (10000, 25000, 50000 y 75000).

Se realizaron 20 ejecuciones independientes del AE con cada uno de los distintos criterios de parada, para cada una de las instancias del problema. Para analizar las soluciones obtenidas se aplicó el test de Shapiro-Wilk (S-W) [18] para establecer si los valores de costo obtenidos seguían o no una distribución normal y poder comparar apropiadamente las medias/medianas. Shapiro-Wilk ha demostrado un mejor desempeño para el chequeo de normalidad entre los tests estadísticos de normalidad más utilizados [19].

La Tabla I reporta los siguientes valores: mejor valor de costo alcanzado ($min(CT)$); promedio de los mejores valores de costo alcanzados (\bar{CT}); varianza de los mejores valores de costo alcanzados ($\sigma^2(CT)$); p-valor del test de Shapiro-Wilk sobre los valores de costo ($p\text{-valor S-W}$); generación en que se alcanzó la mejor solución ($min(gen)$); generación promedio en que se alcanzó la mejor solución (\bar{gen}); tiempo de ejecución mínimo ($min(T)$); tiempo de ejecución promedio (\bar{T}); y varianza del tiempo de ejecución ($\sigma^2(T)$). Todos los tiempos reportados se miden en segundos.

Los resultados numéricos reportados en la Tabla I demuestran que para un nivel de significación de $\alpha = 0.05$ en el test de Shapiro-Wilk para normalidad sobre los valores de costo, todas las ejecuciones cumplen $p\text{-valor} > \alpha$, de donde es posible concluir que no puede descartarse la hipótesis nula del test de Shapiro-Wilk, que propone que los valores de costo obtenidos siguen una distribución normal.

Observando los promedios de los mejores valores de costo alcanzados, es claro que ejecuciones con un mayor número de generaciones llevan a mejores soluciones. Sin embargo, la mejora alcanzada depende de las características propias de la instancia que se resuelve. Por ejemplo, para la instancia #3 se observa un 22.5 % de mejora en promedio entre las soluciones obtenidas con 75.000 generaciones y las obtenidas con 10.000 generaciones, mientras que para la instancia #6 esta mejora es de un 10.6 % en promedio.

2) Comparación contra un algoritmo ávido que resuelve el problema: Un algoritmo ávido (o *greedy*) es un método de construcción de soluciones que toma la decisión localmente óptima en cada paso, con la esperanza de llegar a un óptimo global del problema. Como se mencionó en III, es relevante comparar el desempeño del AE frente a un algoritmo ávido, en términos de la calidad de las soluciones alcanzadas y de

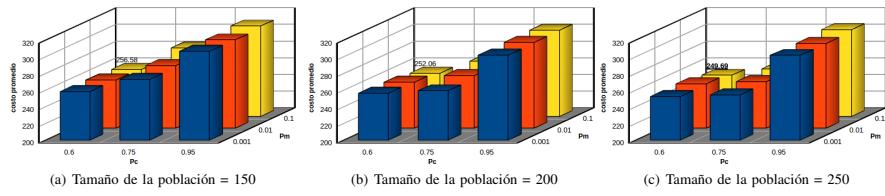


Figura 3. Costo promedio obtenido en las instancias de calibración con diferentes configuraciones de parámetros (los valores más bajos se prefieren).

Tabla I. EVALUACIÓN EXPERIMENTAL: DESEMPEÑO DEL AE

<i>test</i>	#gen.	<i>min(CT)</i>	\overline{CT}	$\sigma^2(CT)$	<i>p-valor S-W</i>	<i>min(gen)</i>	\overline{gen}	<i>min(T)</i>	\overline{T}	$\sigma^2(T)$
(32 pasajeros)	10000	421.0	471.6	1343.6	0.4	982.0	7877.1	13.5	13.6	7.1e-03
	25000	378.6	411.5	283.8	0.3	12230.0	2.1e+04	33.6	34.3	0.2
	50000	372.4	397.3	187.4	0.6	23271.0	4.3e+04	68.1	70.0	2.1
	75000	366.8	388.1	112.5	0.1	20836.0	5.7e+04	101.3	102.7	1.3
(26 pasajeros)	10000	118.1	134.9	139.6	0.1	2214.0	7605.6	10.3	10.5	1.1e-02
	25000	113.1	128.7	58.8	0.5	10448.0	1.8e+04	25.8	26.5	0.2
	50000	113.3	124.2	40.4	0.7	11971.0	3.2e+04	52.9	56.6	27.7
	75000	115.2	123.2	29.2	0.7	11825.0	5.0e+04	76.8	78.9	0.8
(35 pasajeros)	10000	367.5	464.4	2994.0	0.1	2750.0	7070.5	15.2	15.4	1.1e-02
	25000	372.6	427.8	1703.1	0.2	1292.0	1.6e+04	38.1	38.4	5.2e-02
	50000	336.0	367.2	322.9	0.6	19436.0	4.2e+04	76.6	77.5	0.5
	75000	335.7	360.0	183.6	0.6	41013.0	6.5e+04	115.1	119.4	9.8
(41 pasajeros)	10000	233.9	258.4	237.2	0.4	4463.0	7882.6	18.7	19.1	5.5e-02
	25000	209.3	240.9	273.9	0.7	16049.0	2.1e+04	46.1	47.2	0.2
	50000	211.4	228.6	94.9	0.7	22717.0	4.2e+04	92.1	96.9	22.0
	75000	199.4	222.9	133.8	1.0	33033.0	6.4e+04	140.0	142.0	0.8
(41 pasajeros)	10000	521.9	613.9	3928.9	0.6	226.0	7029.2	18.7	19.0	5.1e-02
	25000	468.6	530.0	1764.0	0.1	4259.0	2.0e+04	46.8	51.3	32.6
	50000	464.5	491.9	417.9	0.2	16940.0	4.0e+04	93.7	95.1	0.7
	75000	454.9	483.0	252.6	0.2	39926.0	6.0e+04	141.7	143.2	4.7
(35 pasajeros)	10000	401.6	445.8	586.0	0.8	3728.0	8679.0	15.2	15.6	7.7e-02
	25000	397.1	417.4	198.6	0.2	11685.0	20066.0	38.1	38.7	0.4
	50000	368.7	403.8	246.2	0.7	15776.0	4.0e+04	76.9	78.1	1.9
	75000	380.0	398.4	98.0	0.4	42089.0	6.3e+04	115.7	119.8	5.7

Tabla II. EVALUACIÓN EXPERIMENTAL: COMPARACIÓN ENTRE AE Y ALGORITMO ÁVIDO

<i>test</i>	<i>algoritmo</i>	<i>min(CT)</i>	\overline{CT}	<i>min(T)</i>	\overline{T}	<i>p-valor M-W</i>	mejora sobre alg. ávido (max)	mejora sobre alg. ávido (prom.)
(32 pasajeros)	<i>AE</i> 10000	421.0	471.6	13.4	13.6	1.5e-05	19.6 %	10.0 %
	<i>AE</i> 25000	378.6	411.5	33.6	34.3	0.0	27.8 %	21.5 %
	<i>AE</i> 50000	372.4	397.3	68.1	70.0	0.0	28.9 %	24.2 %
	<i>AE</i> 75000	366.8	388.1	101.3	102.7	0.0	30.0 %	25.9 %
	alg. ávido	524.0	524.0	0.0	0.0	-	-	-
(26 pasajeros)	<i>AE</i> 10000	118.1	134.9	10.3	10.5	0.0	39.1 %	30.4 %
	<i>AE</i> 25000	113.1	128.7	25.8	26.5	0.0	41.7 %	33.6 %
	<i>AE</i> 50000	113.3	124.2	52.9	56.6	0.0	41.6 %	36.0 %
	<i>AE</i> 75000	115.2	123.2	76.8	78.9	0.0	40.6 %	36.4 %
	alg. ávido	193.9	193.9	0.0	0.0	-	-	-
(35 pasajeros)	<i>AE</i> 10000	367.5	464.4	15.2	15.4	1.5e-05	29.6 %	11.0 %
	<i>AE</i> 25000	372.6	427.8	38.1	38.4	0.0	28.6 %	18.0 %
	<i>AE</i> 50000	336.0	367.2	76.6	77.5	0.0	35.6 %	29.6 %
	<i>AE</i> 75000	335.7	360.0	115.1	119.4	0.0	35.7 %	31.0 %
	alg. ávido	522.0	522.0	0.0	0.0	-	-	-
(41 pasajeros)	<i>AE</i> 10000	233.9	258.4	18.7	19.1	1.5e-05	14.1 %	5.1 %
	<i>AE</i> 25000	209.3	240.9	46.1	47.2	0.0	23.1 %	11.5 %
	<i>AE</i> 50000	211.4	228.6	92.1	96.9	0.0	22.4 %	16.0 %
	<i>AE</i> 75000	199.4	222.9	140.0	142.0	0.0	26.7 %	18.1 %
	alg. ávido	272.2	272.2	0.0	0.0	-	-	-
(35 pasajeros)	<i>AE</i> 10000	521.9	613.9	18.7	19.0	3.0e-02	9.6 %	-6.3 %
	<i>AE</i> 25000	468.6	530.0	46.8	51.3	1.5e-04	18.8 %	8.2 %
	<i>AE</i> 50000	464.5	491.9	93.7	95.1	0.0	19.5 %	14.8 %
	<i>AE</i> 75000	454.9	483.0	141.7	143.2	0.0	21.2 %	16.4 %
	alg. ávido	577.3	577.3	0.0	0.0	-	-	-
(35 pasajeros)	<i>AE</i> 10000	401.6	445.8	15.2	15.6	3.0e-02	12.7 %	3.1 %
	<i>AE</i> 25000	397.1	417.4	38.1	38.7	0.0	13.7 %	9.3 %
	<i>AE</i> 50000	368.7	403.8	76.9	78.1	0.0	19.9 %	12.3 %
	<i>AE</i> 75000	380.0	398.4	115.7	119.8	0.0	17.4 %	13.4 %
	alg. ávido	460.2	460.2	0.0	0.0	-	-	-

la eficiencia computacional. Con este propósito, se implementó un algoritmo ávido sencillo para resolver el problema de planificación de viajes compartidos abordado en este artículo.

El algoritmo ávido implementado se basa en aplicar una técnica de agrupamiento simple e intuitiva. Toma un taxi, y agrega el destino más cercano al taxi actual hasta completarlo (en cuyo caso forma un nuevo taxi), o hasta que todos los pasajeros hayan sido asignados. Una excepción ocurre cuando el costo de añadir un nuevo destino al taxi actual es mayor al costo de asignar un nuevo taxi que transporte únicamente a ese pasajero. En este caso, se forma un nuevo taxi y se "cierra" el anterior. Este comportamiento emula una estrategia intuitiva para resolver el problema con un enfoque de agregación de estrategias de decisión localmente óptimas, y se aproxima a lo que un grupo de usuarios humanos pueden idear de manera simple para resolver el problema. La estructura básica del algoritmo ávido implementado para la comparación de los resultados obtenidos por el AE se muestra en el Algoritmo 5.

Algorithm 5 Algoritmo ávido para compartir taxis

```

M = CMAX;
taxi = 1;
contador = 0;
mientras pasajerosSinAsignar() hacer
    si (contador == 0) entonces
        p1 = destinoMasCercanoDesdeOrigen();
        agregarPasajero(p1,taxi,solucion);
        contador++;
    en otro caso si (contador ≥ M) entonces
        contador = 0; {El taxi actual está completo}
        taxi++;
    en otro caso
        p2 = vecinoMasCercano(p1);
        si (costo(p1,p2) ≤ costo(origen,p2) + B) entonces
            agregarPasajero(p2,taxi,solucion);
            p1 = p2;
            contador++;
    en otro caso
        taxi++; {Comenzar con un nuevo taxi}
        contador = 0;
    fin si
fin mientras
retornar solucion;
```

La Tabla II reporta el análisis comparativo entre los resultados calculados por el AE y el algoritmo ávido, para las instancias de la evaluación experimental. Para analizar la significancia estadística de la comparación, se aplicó el test bilateral U de Mann-Whitney [20] para establecer si los resultados obtenidos con el AE tienen una diferencia significativa con los calculados por el algoritmo ávido. La columna *p*-valor *M-W* en la Tabla II reporta el *p*-valor del test de Mann-Whitney sobre los valores de costo calculados por el AE. Las restantes columnas corresponden a las explicadas para la tabla previa. Todos los tiempos reportados se miden en segundos.

El test U de Mann-Whitney aplicado sobre las distribuciones de resultados muestra que existe una diferencia significativa entre las soluciones obtenidas por el AE y aquellas alcanzadas por el algoritmo ávido. Para todos los casos,

excepto para la instancia #5, una única ejecución de 25.000 generaciones fue suficiente para obtener una mejora sobre la solución obtenida por el algoritmo ávido. Para la instancia #2, ejecutando por solo 25.000 generaciones, se alcanzó un **33.6 %** de mejora en promedio y **41.7 %** en el mejor caso. Para la instancia #3 estos porcentajes son de **18.0 %** en promedio y **28.6 %** en el mejor caso. Sin embargo, permitiendo al AE evolucionar por 75.000 generaciones, la mejora sube a **31.0 %** en promedio y **35.7 %** en el mejor caso.

La Figura 4 resume los promedios de los mejores valores y valores promedios de mejora (diferencia porcentual) con respecto al algoritmo ávido, encontrados por el AE utilizando 75.000 generaciones.

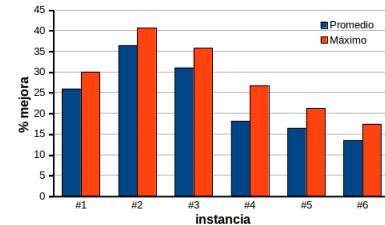


Figura 4. Porcentajes de mejora con respecto al algoritmo ávido.

Dado que el algoritmo ávido ejecuta en un tiempo negligible, es importante evaluar el tiempo requerido por el AE para calcular mejores resultados. Este tiempo de respuesta se verá reflejado cuando el usuario final utilice el AE en una aplicación web o móvil. Para este fin, se diseñaron experimentos de 20 ejecuciones independientes, sobre las mismas 6 instancias anteriores, programando el AE con el siguiente criterio de parada: si supera al algoritmo ávido en un 20 %, si no mejora la solución por 20.000 generaciones (se asume que la búsqueda se estancó), o al cabo de 100.000 generaciones.

Las Figuras 5(a)–5(f) reportan el tiempo requerido por el AE para alcanzar soluciones de igual calidad que el algoritmo ávido, y para superarlo en un 10 % y en un 20 %, para las seis instancias del problema abordadas en el análisis comparativo. Sobre cada barra se indica el porcentaje de mejora que alcanzó el AE sobre el algoritmo ávido.

Los resultados muestran que el AE es capaz de obtener resultados similares al ávido al ejecutar por un tiempo en el entorno de los 10 segundos, y mejorarlo en más del 10 % en la mayoría de los casos, al ejecutar por un tiempo mayor. Este tiempo depende fuertemente de la instancia que se resuelve y varía desde menos de 30 segundos hasta un par de minutos.

VII. APLICACIONES WEB Y MÓVIL

Para la resolución de instancias reales del problema se desarrolló un sistema formado por una aplicación web y una aplicación móvil, al cual llamaremos de ahora en más *Planificador de Viajes Compartidos en Línea*.

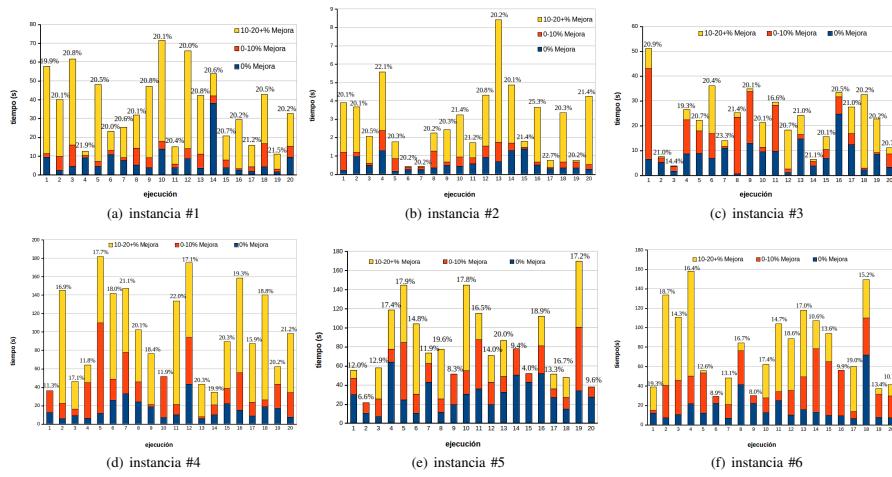


Figura 5. Tiempo requerido por el AE para alcanzar/mejorar al algoritmo ávido.

VII-A. Arquitectura

Una de las principales decisiones de diseño fue seleccionar una arquitectura acorde a los requerimientos del sistema. Los mismos contemplaban los siguientes puntos:

- Ejecución de un programa de alto peso computacional para el cálculo de la solución mediante un AE.
- Fuertes conexiones con los servicios de Google Maps para obtener las distancias entre los puntos.
- Gran volumen de procesamiento remoto, involucrando operaciones de lectura/escritura en archivos.
- Conexiones temporales con aplicaciones móviles para el cálculo de la solución.
- Necesidad de un desarrollo ágil.

En base a los puntos mencionados, se desarrolló un sistema con una arquitectura simple pero escalable, utilizando tecnologías particulares en cada uno de los componentes. Los detalles se presentan en la Figura 6.

1) *Servidor Central*: Se trata del componente principal del *Planificador de Viajes Compartidos*. Su rol central lo ubica como un componente imprescindible para que el sistema se comporte correctamente y sea funcional. El servidor central es el encargado de recibir las solicitudes de los usuarios web, y retornar las páginas web que el usuario tendrá en su navegador como resultado de utilizar el sistema. Por otra parte, brinda una *Interfaz de Programación de Aplicaciones (API)* para la comunicación y posterior ejecución de instancias en aplicaciones ejecutando en teléfonos móviles. En cuanto a particularidades técnicas, el servidor central está alojado en la infraestructura gratuita de *Heroku* [21] y está implementado en el lenguaje de programación dinámico *Ruby* [22], usando el framework de desarrollo *Rails* [23].

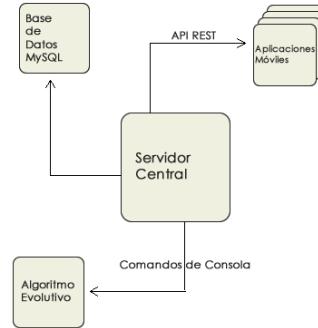


Figura 6. Arquitectura del Sistema Planificador de Viajes Compartidos.

2) *Aplicación Móvil*: Este componente permite utilizar el *Planificador de Viajes Compartidos* desde teléfonos móviles. Particularmente, fue desarrollado para teléfonos inteligentes con sistema operativo *iOS*.

3) *Base de datos*: Con el fin de optimizar las consultas realizadas al sistema y persistirlas en el tiempo, se hace uso de una base de datos MySQL relacional. En la implementación actual la base de datos se encuentra alojada en el mismo servidor que el Servidor Central, aunque es posible migrarla a otro equipo si es necesario mejorar el desempeño computacional del *Planificador de Viajes Compartidos* como sistema.

La base de datos implementada para el *Planificador de Viajes Compartidos* cuenta con cuatro tablas principales: i) *locations*, ii) *queries*, iii) *users* y iv) *roles*, para modelar

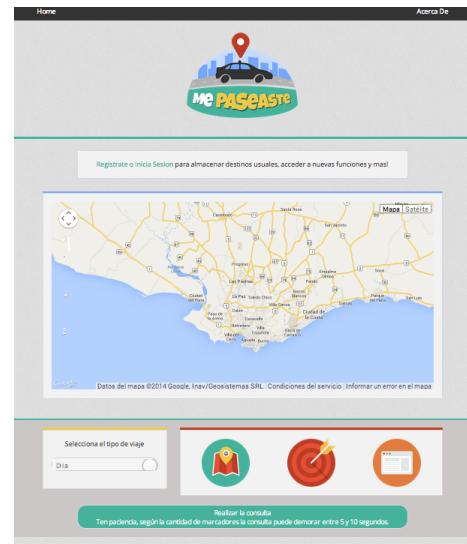


Figura 7. Interfaz de la aplicación web para el problema de compartir taxis.

VII-B. Aplicación Web

Con el fin de acelerar el proceso de desarrollo, luego de investigar las opciones disponibles para implementar el sistema, se decidió desarrollar el Servidor Central basándose en el lenguaje de programación Ruby y el framework Rails. La aplicación web se encuentra disponible en línea en la dirección www.mepaseaste.uy. La interfaz de usuario de la aplicación desarrollada se presenta en la Figura 7.

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos. Se trata de un lenguaje interpretado en una sola pasada y su implementación oficial es distribuida bajo una licencia de software libre. Rails, por su parte, es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC).

Uno de las ventajas más importantes que tiene Ruby On Rails sobre los demás entornos que estuvieron en discusión, fue la facilidad de ejecutar programas externos de consola. Complementariamente, permite publicar de forma simple una API para la comunicación externa de dispositivos móviles.

1) Modelos: En las aplicaciones web orientadas a objetos sobre bases de datos, el Modelo consiste en las clases que representan las tablas de la base de datos. Las definiciones de las clases también detallan las relaciones entre clases con sentencias de mapeo objeto relacional. Por ejemplo, si la clase *Consulta* tiene una definición *has_many:ubicaciones*, y existe una instancia de Consulta llamada *a*, entonces *a.ubicaciones* devolverá un array con todos los objetos *Ubicaciones* cuya columna *consulta_id* (en la tabla *comentarios*) sea igual a *a.id*.

Dos modelos importantes implementados en el Servidor Central son *Consulta* y *Ubicacion*. *Consulta* modela las consultas del usuario, con una serie de atributos particulares, por ejemplo si la consulta es pública o no, el número de ubicaciones, entre otros, mientras que *Ubicacion* modela las ubicaciones ingresadas, identificadas por su latitud y longitud. Además, se implementaron modelos para los usuarios y roles, que brindan funcionalidades extra a la aplicación web, tales como el inicio de sesión y el manejo de una sección de administración simple.

2) Vistas: En las aplicaciones MVC existe un componente fundamental, llamado *Vista*, que representa los datos obtenidos por el Controlador en el navegador del usuario. Usualmente, en las aplicaciones web la vista consiste en una cantidad mínima de código incluido en HTML. En el caso de la aplicación web desarrollada para la planificación de taxis, existen vistas para ingresar los puntos, y vistas para visualizar los resultados. Asimismo, también existen vistas orientadas a mejorar la experiencia de usuario, incluyendo un perfil de usuario y páginas de inicio de sesión, entre otros.

3) Controladores: En MVC, las clases del Controlador responden a la interacción del usuario e invocan a la lógica de la aplicación, que a su vez manipula los datos de las clases del Modelo y muestra los resultados usando las Vistas. En las aplicaciones web basadas en MVC, los métodos del controlador son invocados por el usuario usando el navegador web. En la aplicación web implementada, existen controladores para los resultados y para la gestión de los puntos ingresados.

4) Procesamiento en el navegador: Un punto no menor, es que la aplicación tiene gran cantidad de procesamiento en el navegador del cliente. Cada vez que el usuario inserta un punto en el mapa, se ejecutan una serie de métodos asíncronos en el navegador del usuario, utilizando la tecnología Javascript/AJAX, para obtener la distancia del punto ingresado a todos los demás previamente añadidos. La secuencia de ejecución es la siguiente:

- Cuando el usuario realiza un click sobre el mapa, se almacena su latitud y su longitud en un arreglo.
- Para cada uno de los puntos ingresados anteriormente, se realiza una solicitud a la API de Google Maps con las coordenadas del punto al cual se quiere calcular la distancia. El resultado se almacena en una matriz dinámica.
- Cuando el usuario presiona el botón de enviar, la matriz de distancias se envía al servidor, quien se encarga de calcular los costos y ejecutar el AE.

Por otra parte, al mostrar un resultado, también se hace uso de la API de Google Maps para graficar las rutas que debe recorrer cada taxi.

5) Costo de los Taxis: Las tarifas de taxis de las distintas ciudades son obtenidas a través de la API del servicio Taxi Fare Finder [16]. Una invocación a dicha API con las coordenadas de dos puntos devuelve la tarifa asociada a dicho viaje, con información detallada acerca del costo de bajada de bandera, de la distancia recorrida y otros costos adicionales.

VII-C. Aplicación Móvil

La decisión de implementar una aplicación nativa para un conjunto reducido de móviles en lugar de una versión móvil basada en un comportamiento web, pero adaptada a pantallas pequeñas, estuvo fuertemente basada en las razones de experiencia de usuario. La aplicación de planificación de viajes compartidos en taxis tiene como particularidad que despliega información en mapas, y este procesamiento consume una gran cantidad de recursos. El acceso a la aplicación web a través de dispositivos móviles genéricos es posible, pero el comportamiento de la aplicación dista de ser atractivo.

Por este motivo, se optó por desarrollar una versión de la aplicación para teléfonos inteligentes con el sistema operativo iOS de Apple. Entre los modelos soportados se encuentran los iPhones 3G, 3GS, 4, 4S, 5 y 5S. Se espera que la versión para iOS sea la primera de una serie de implementaciones para diferentes plataformas móviles.

1) Prototipo en PhoneGap: Al comenzar el desarrollo, se implementó un prototipo basado en la tecnología *PhoneGap* [24]. PhoneGap es un framework para el desarrollo de aplicaciones móviles producido por Nitobi que permite a los programadores desarrollar aplicaciones para dispositivos móviles utilizando herramientas genéricas tales como JavaScript, HTML5 y CSS3, que en nuestro caso fueron ya utilizadas para el desarrollo de la versión web. Las aplicaciones resultantes son híbridas. Por un lado, no son realmente aplicaciones nativas al dispositivo, ya que la generación de las vistas se realiza mediante un proceso análogo al realizado en los navegadores web y no con interfaces gráficas específicas de cada sistema. Por otro lado, tampoco son aplicaciones web,

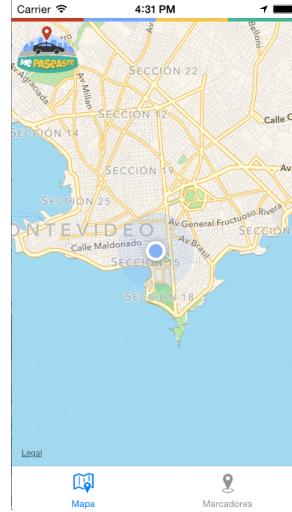


Figura 8. Interfaz de la aplicación móvil para el problema de compartir taxis.

teniendo en cuenta que las aplicaciones son empaquetadas para poder ser desplegadas en el dispositivo incluso trabajando con el API del sistema nativo.

Al evaluar el prototipo desarrollado en PhoneGap, se pudo verificar que, ante las características que el sistema demandaba, una aplicación de este estilo brindaría una mala experiencia de usuario. Como ejemplos de la baja aplicabilidad práctica del prototipo se pudo verificar que la aplicación tardaba varios segundos en responder al ingreso de un único punto, y no respondía correctamente a gestos sobre el mapa.

Los pobres resultados de usabilidad y desempeño del prototipo desarrollado orientaron al desarrollo nativo como la mejor alternativa para la aplicación. Esta decisión acota el número de dispositivos para los cuales es posible utilizar el servicio, pero brinda una aplicación con una mejor experiencia de uso y un desempeño muy superior.

2) Desarrollo Nativo: La aplicación nativa desarrollada, a diferencia del prototipo, fue implementada y evaluada en un conjunto específico de dispositivos. La interfaz de usuario de la aplicación móvil desarrollada se presenta en la Figura 8. Un punto importante es que la aplicación móvil solamente funciona como intermediario entre el usuario móvil y el Servidor Central. El cálculo de la asignación de pasajeros y recorridos de los taxis, al igual que en la versión web, se realiza en el Servidor Central.

VIII. CONCLUSIONES

Este trabajo ha presentado el diseño e implementación de un algoritmo evolutivo para la resolución del problema de planificación en línea de viajes compartidos en taxis desde un origen a múltiples destinos.

El algoritmo evolutivo propuesto conforma una precisa y eficiente solución para el problema abordado. El análisis experimental realizado usando un conjunto de instancias del problema construidas en base a datos reales tomados de GPS de taxis de la ciudad de Beijing, mostró que los resultados del algoritmo evolutivo permiten mejorar significativamente—hasta un **41.7%** en el mejor caso y **36.4%** en el caso promedio—los resultados sobre los calculados empleando un algoritmo ávido que simula el comportamiento intuitivo de un grupo de usuarios humanos al enfrentarse al problema. Los resultados se obtienen en tiempos de ejecución reducidos, permitiendo su aplicación para la resolución en línea del problema.

Adicionalmente, se presentó el desarrollo de un sistema que incluye una aplicación web y una aplicación móvil para dispositivos iOS, que permiten a los usuarios finales resolver instancias reales del problema de una forma simple y amigable. Dado que las instancias utilizadas en el análisis experimental del método propuesto tienen dimensiones mayores a las que usualmente se proponen resolver en la práctica por usuarios reales a través de una aplicación web o móvil, es de esperar que el algoritmo evolutivo mantenga, o incluso supere, el muy buen desempeño obtenido para las instancias abordadas en este trabajo.

Las principales líneas de trabajo futuro se orientan a mejorar el desempeño del algoritmo evolutivo propuesto, para mejorar su aplicabilidad a casos realistas de alta demanda. En este sentido, el uso de modelos paralelos que permitan obtener soluciones de calidad reduciendo el tiempo de ejecución, surge como una de las principales alternativas a explorar en esta línea de trabajo. Adicionalmente, la posibilidad de estudiar alternativas del problema considerado puede resultar de gran interés. Entre las variantes en las que se está trabajando actualmente se puede mencionar las que incorporan restricciones temporales al problema, preferencias personales de los pasajeros, y también las que plantean incorporar datos realistas del tráfico de las ciudades. Por otra parte, la línea de trabajo marcada también considera problemas multi origen, donde se tienen múltiples vehículos que pueden partir de diversos órigenes. La combinación entre las restricciones temporales y el problema multi origen presenta un desafío interesante que puede ser afrontado en futuras investigaciones. Por último, otras metaheurísticas podrían ser estudiadas con el objetivo de analizar su adaptabilidad a las propiedades del problema, para potencialmente encontrar mejores soluciones, junto con respectivos análisis de desempeño computacional entre las alternativas.

En la página web del proyecto <http://www.fing.edu.uy/ince/grupos/cecal/hpc/AG-Taxi> se encuentran publicados los siguientes recursos: i) ejecutable del algoritmo evolutivo y del algoritmo ávido con instrucciones; ii) generador de instancias del problema; iii) instancias del problema y resultados numéricos detallados de los experimentos de calibración y evaluación del algoritmo evolutivo propuesto; y iv) link a la interfaz web y a la aplicación móvil.

AGRADECIMIENTOS

Las actividades de investigación reportadas en este artículo han sido parcialmente financiadas por ANII y PEDECIBA, Uruguay.

REFERENCIAS

- [1] N. Fellows and D. Pittfield, "An economic and operational evaluation of urban car-sharing," *Transportation Research Part D: Transport and Environment*, vol. 5, no. 1, pp. 1–10, 2000.
- [2] E. Ferrari, R. Manzini, A. Pareschi, A. Persona, and A. Regattieri, "The car pooling problem: Heuristic algorithms based on savings functions," *Journal of Advanced Transportation*, vol. 37, pp. 243–272, 2003.
- [3] R. Katzev, "Car sharing: A new approach to urban transportation problems," *Analyses of Social Issues and Public Policy*, vol. 3, no. 1, pp. 65–86, 2003.
- [4] A. Letchford, R. Eggle, and J. Lysgaard, "Multistars, partial multistars and the capacitated vehicle routing problem," *Mathematical Programming*, vol. 94, no. 1, pp. 21–40, 2002.
- [5] S. Nesmachnow, "Metaheuristics as soft computing techniques for efficient optimization," in *Encyclopedia of Information Science and Technology*. M. Khosrow-Pour, Ed., IGI Global, 2014, pp. 1–10.
- [6] "How to Split a Shared Cab Ride? Very Carefully, Say Economists," *The Wall Street Journal*, 8 de diciembre, 2005, [Online] <http://online.wsj.com/news/articles/SB113279169439805647>; accessed 07-04-2014.
- [7] C.-L. Xin and W.-M. Ma, "Scheduling for on-line taxi problem on a real line and competitive algorithms," in *Proceedings of the International Conference on Machine Learning and Cybernetics*, vol. 5, Aug 2004, pp. 3078–3083.
- [8] E. Kotsoupias, "The k-server problem," *Computer Science Review*, vol. 3, no. 2, pp. 105–118, 2009.
- [9] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 2013, pp. 410–421.
- [10] C.-C. Tao and C.-Y. Chen, "Heuristic algorithms for the dynamic taxipooling problem based on intelligent transportation system technologies," in *4th International Conference on Fuzzy Systems and Knowledge Discovery*, vol. 3, Aug 2007, pp. 590–595.
- [11] T. Bäck, D. Fogel, and Z. Michalewicz, Eds., *Handbook of evolutionary computation*. Oxford University Press, 1997.
- [12] "The Malva Project: A framework for computational intelligence in C++," [Online] <https://github.com/themalvaproject>, accessed 04-04-2014.
- [13] E. Alba, G. Luque, J. Garcia-Nieto, G. Ordóñez, and G. Leguizamón, "Maliba: a software library to design efficient optimisation algorithms," *International Journal of Innovative Computing Applications*, vol. 1, no. 1, pp. 74–85, 2007.
- [14] C. M. Schweik, M. T. Fernandez, M. P. Hamel, P. Kashwan, Q. Lewis, and A. Stepanov, "Reflections of an online geographic information systems course based on open source software," *Social Sciences Computing Review*, vol. 27, no. 1, pp. 118–129, Feb. 2009.
- [15] "Google Earth," [Online] <http://www.google.com/intl/en/earth/>, accessed 19-04-2014.
- [16] "TaxiFareFinder API (beta)," [Online] <http://www.taxifarefinder.com/api.php>, accessed 19-04-2014.
- [17] E. Cantu-Paz and D. E. Goldberg, "Are multiple runs of genetic algorithms better than one?" in *Proceedings of the Genetic and Evolutionary Computation Conference*. Springer Verlag, 2003, pp. 801–812.
- [18] S. Shapiro and M. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.
- [19] N. Razali and Y. Wah, "Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests," *Statistics and Computing*, vol. 2, no. 3, pp. 117–119, 2011.
- [20] H. Mann and D. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.
- [21] N. Middleton and R. Schneeman, *Heroku: Up and Running*. O'Reilly Media, Inc., 2013.
- [22] B. Tate and C. Hibbs, *Ruby on Rails: Up and Running*. O'Reilly Media, Inc., 2006.
- [23] D. Flanagan and Y. Matsumoto, *The Ruby Programming Language*. O'Reilly, 2008.
- [24] J. Wargo, *PhoneGap Essentials: Building Cross-Platform Mobile Apps*. Addison-Wesley Professional, 2012.

A parallel micro evolutionary algorithm for taxi sharing optimization

Renzo Massobrio, Gabriel Fagúndez and Sergio Nesmachnow
Centro de Cálculo, Facultad de Ingeniería, Universidad de la República
Herrera y Reissig 565, 11300 Montevideo, Uruguay
{renzom, gabrielf, sergion}@fing.edu.uy

Abstract—This article presents the application of a parallel micro evolutionary algorithm to the problem of distributing passengers traveling from the same origin to different destinations in several taxis, with the goal of minimizing the total cost of the trips. The proposed method is designed to provide an accurate and efficient way to solve the problem. The experimental analysis compares the solutions found using the proposed algorithm versus those computed using a sequential evolutionary algorithm and an intuitive greedy heuristic. The results show that the parallel evolutionary algorithm is able to efficiently reach significant improvements in the total cost, outperforming the greedy heuristic in up to 36.1% (18.2% on average), and the sequential evolutionary algorithm in up to 8.5% (4.3% on average).

Keywords—evolutionary algorithms; vehicle sharing

I. INTRODUCTION

Car pooling is the concept of sharing car journeys, and it has gained massive public attention in recent years [1]. Both economical and environmental benefits (at individual and collective levels) are obtained by sharing car trips, as it minimizes travel expenses as well as the amount of vehicles on the streets, thus reducing pollution. This contributes to minimize the impact of transportation in the environment, which is a major concern nowadays, especially for big cities [2] [3]. Additionally, less traffic leads to fewer traffic jams, resulting in a more fluent, thus more efficient, trip. From an economical perspective, sharing trips between multiple passengers significantly reduces the transportation costs.

The aforementioned benefits have led to initiatives to attend the public concern on this topic. Exclusive car-pool lanes, campaigns to promote car sharing, and a plethora of mobile applications to find car pooling mates [4], are some of the many examples that illustrate the importance of this subject.

Taxis are a fast and reliable mean of transportation. However, they rarely run at full capacity and could therefore benefit from the car pooling idea. There are some web platforms that provide solutions for ride-sharing scheduling, and some research works on different variants of the taxi-pooling problem. The taxi sharing problem is NP-hard [5]. Thus, heuristics and metaheuristics [6] are needed to find high-quality solutions in reasonable execution times for realistic problem instances.

This article presents a parallel micro evolutionary algorithm to solve the one-origin-multiple-destinations variant of the taxi sharing problem. The experimental evaluation over real-world scenarios demonstrates that the proposed method is an accurate and efficient tool to solve the problem, which can be easily integrated in on-line (web, mobile) applications.

The article is organized as follows. Section II introduces the taxi sharing problem and reviews related work. Section III introduces evolutionary algorithms (EAs), and the proposed micro EA to solve the problem. Section IV reports the experimental evaluation, including a comparison against a sequential EA from [12] and a greedy heuristic to solve the problem. Finally, Section V formulates the conclusions and the main lines for future work.

II. THE TAXI SHARING PROBLEM

This section introduces the taxi sharing problem and reviews related works on the topic.

A. Problem model and formulation

The problem models the reality of a group of people (the *passengers*) willing to share a taxi from the same origin to different destinations. Passengers are interested in knowing the appropriate number of taxis needed and how to visit their destinations in order to minimize the total cost.

Passenger distribution is restricted to the maximum number of people allowed by safety regulations to travel in each taxi. The transportation costs model realistic data in a fluid traffic scenario (including a model for delays and traffic congestion is proposed as future work). The cost for each taxi includes the cost to hire the taxi (*minimum fare*), and the cost for traveling from the origin to the final destination; additional costs related to baggage, tips, or waiting times are not considered.

The mathematical formulation of the taxi sharing problem considers the following elements:

- A set of passengers $P = \{p_1, p_2, \dots, p_N\}$; travelling from the same origin point O to a set of (potentially different) destination points $D = \{d_1, d_2, \dots, d_N\}$;
- A set of taxis $T = \{t_1, t_2, \dots, t_M\}$, $M \leq N$; the maximum number of passengers in the taxi is C_{MAX} , and a function $C:T \rightarrow \{0, 1, \dots, C_{MAX}\}$ indicates how many passengers use the taxi in a trip;
- A symmetric matrix M (dimension $(N+1) \times (N+1)$) with the distances between each one of the geographic points in the problem (one origin and N destinations);
- A cost function c_T given by the constant MF (*minimum fare*) and the cost by distance $c(\text{dist}(O, d_i))$.

The problem goal is to find a planning function $f:P \rightarrow T$ to transport the N passengers in K taxis ($K \leq M$), determining both the passengers-to-taxis assignment and the order to visit the destinations, minimizing the total cost (TC , Eq. 1).

$$TC = \sum_{t_i} (MF + \sum_{j=1}^{C(t_i)} c(\text{dist}(d_{j-1}, d_j))) \quad (1)$$

The proposed formulation minimizes the total cost. Different options can be applied to compute the cost for each passenger [7], including: i) paying for distance and equally dividing the minimum fare, each passenger j in a shared taxi t_i pays $MF/C(t_i) + \text{dist}(d_{j-1}, d_j)$; ii) paying a flat fare, disregarding distances, $(MF + \sum_{j=1}^{C(t_i)} c(\text{dist}(d_{j-1}, d_j))) / C(t_i)$, etc.

The taxi sharing problem is NP-hard, as it is a variant of the *car pooling* problem [5]. Thus, when dealing with realistic problem instances, heuristics and metaheuristics [6] are the most useful methods to find high-quality solutions in reasonable execution times.

B. Related work

The *taxi pooling* problem is a variant of the *car pooling* problem, which has been studied from different perspectives in the related literature. From the point of view of the taxi companies, Xin *et al.* [8] studied the optimization of the total cost of k taxis serving clients on-line, applying a heuristic that sends the two nearest taxis to attend a request, promoting competition and avoiding underutilization. The results showed that the problem, as a generalization of the k -servers problem [9], holds a competitive ratio of k against an optimal algorithm that knows the entire sequence of requests beforehand.

Balancing the interest of both taxi owners and users, Ma *et al.* [10] proposed a dynamic system for taxi sharing, by combining a search and a planning method to find taxis/assign passengers. A lazy strategy is applied to improve execution times, using previously computed results to delay the shortest path calculation as much as possible. Using a database of 33.000 GPS taxi trajectories from Beijing, the dynamic system reduced the travelled distances up to 13%, while serving 25% more requests in a simulated scenario with 6 requests per taxi.

Closer to the problem we tackle in this article, which focuses mostly on the interests of the customers, Tao *et al.* [11] proposed two heuristic algorithms to optimize taxi ride-sharing costs based on greedy strategies. One algorithm is applied to the one-origin-to-many-destinations problem, and the other one is designed for the many-to-one scenario. The results of a field test of taxi-pooling at Taipei with 10 taxis and 798 passengers show an average matching success rate of 60.3%. However, the fuel savings results are shown only in absolute terms, making it difficult to extract meaningful information to compare with other techniques. The one-to-many problem is the one tackled in this paper, so it is of particular interest to see the performance of the greedy method against the parallel method evolutionary algorithm we propose here.

Besides the academic proposals, there are several on-line applications to solve different version of the car pooling problem, including Carpooling (<http://carpooling.com>), which allows finding trip partners to share costs when people use their own cars. Carpling (<http://carpling.com>) focuses on taxi pooling from the point of view of the companies, finding users with nearby destinations. The solutions computed by Carpling are restricted to trips totally contained in other trips, without

considering different trip combinations. There are no proposals of applications solving the taxi sharing problem by explicitly computing routes as proposed in this article.

The analysis of related works indicates that the taxi sharing problem is interesting for the scientific community, having a significant impact on environment protection and economy. There are few user-oriented solutions in literature, so there is room to contribute in this line of research, by proposing efficient methods for planning and reducing vehicular traffic.

In our previous work [12], we proposed a simple EA for taxi sharing optimization that showed a good capability of solving realistic medium-size problem instances, but requiring large execution times. The parallel micro EA we propose in this article is conceived to improve both the quality of results and the computational efficiency of the search.

III. A PARALLEL MICRO EA FOR TAXI SHARING

This section introduces EAs and describes the proposed parallel micro-EA for taxi sharing.

A. Evolutionary algorithms

EAs are non-deterministic methods that emulate the evolution of species in nature to solve optimization, search, and learning problems [13]. In the last twenty-five years, EAs have been successfully applied for solving optimization problems underlying many real applications of high complexity.

Parallel models are a popular option to improve the efficiency and the efficacy of EAs. By splitting the population into several computing elements, parallel evolutionary algorithms (PEAs) allow reaching high quality results in a reasonable execution time even for hard-to-solve optimization problems. The parallel EA proposed in this work is categorized within the *distributed subpopulations* model [14]: the population is split in several subpopulations (*demes*). Each deme runs a serial EA, and the individuals are able to interact only with other individuals in the deme. An additional *migration* operator is defined: occasionally some individuals are exchanged among demes, introducing a new source of diversity in the EA.

Micro-EAs provide an efficient alternative to reduce the execution times when solving on-line optimization problems, by using small populations and restarting procedures. In our research group, micro-EAs have been applied successfully to solve combinatorial optimization problems [15] [16]. The main features of the proposed parallel micro-EA to solve the taxi sharing problem are presented next.

B. Implementation details: encoding and fitness function

Solutions are represented as tuples containing integers between 1 and N , representing the passengers, and $N-1$ zeros to separate passengers assigned to different taxis. The order for visiting the destinations is the one specified in the sequence. Fig. 1 shows an example of the solution encoding for an instance with $N = 5$.

The solution encoding has some restrictions/features: i) the number of consecutive (non-zero) integers is limited to C_{MAX} ; ii) each integer must appear only once in the encoding; iii) consecutive zeros mean the same than a single one.

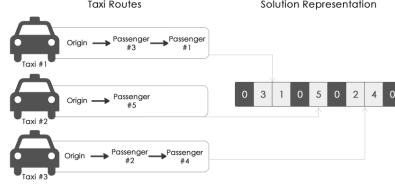


Fig. 1. Example of solution representation for the taxi sharing problem.

The fitness function accounts for the problem objective (cost of the trips). In order to transform the cost minimization problem to a maximization one, we define the fitness function as the inverse of the total cost (defined in Eq. 1).

C. Evolutionary operators

The proposed encoding has specific features and restrictions, so we apply ad-hoc search operators.

Population initialization: the population is initialized using two alternative methods. The *random initialization* uses a constructive algorithm that randomly places the numbers from 1 to N into a tuple initially created with $2N-1$ zeroes. The *greedy initialization* (see next section) applies a random number of perturbations, *i.e.* swaps two elements, over the solution found by a greedy algorithm to solve the problem. Both initialization methods are studied in the experimental evaluation.

Feasibility check and correction process: both initialization methods might violate some of the constraints defined for the solution encoding. Thus, a corrective function is applied to guarantee solution feasibility. The method searches for sequences of non-zero digits larger than the maximum number of passengers allowed per taxi. Then, the correction algorithm locates the first consecutive couple of zeroes, and moves the first zero to a random place at the non-zero sequence, in order to break the invalid group. The search for invalid sequences continues until the end of the solution; at that point, the sequence fulfills all the constraints.

Selection: a *tournament selection* is applied to provide an appropriate selection pressure for the micro populations. Initial experiments confirmed that the standard proportional selection technique does not provide enough diversity to avoid premature dominance and convergence in solutions far from the optimum.

Recombination: we apply an ad-hoc variant of the *Position Based Crossover* (PBX) operator, explained in Algorithm 1. In order to avoid violating the constraints imposed by the solution encoding, the same corrective function used after the initialization is applied on the offspring produced using the PBX crossover. Fig. 2 shows an example of the PBX crossover for two individuals with 5 passengers and $C_{MAX} = 4$.

Mutation: we apply the *Exchange Mutation* operator, which randomly selects and exchanges two positions in the solution encoding. Afterward, the same corrective function mentioned above is applied to fix invalid solutions, if necessary.

Migration: The migration operator is applied asynchronously, considering the demes connected in an unidirectional ring topology. Immigrants are selected applying a

Algorithm 1 Ad-hoc PBX for the taxi sharing problem

- 1: Randomly select several positions in parent 1.
- 2: Partially generate the offspring, copying the selected values from parent 1.
- 3: Mark in parent 2 the positions already selected in parent 1.
- 4: Select the next non-marked value in parent 2, sequentially from the beginning, and copy it in the first free position in offspring.

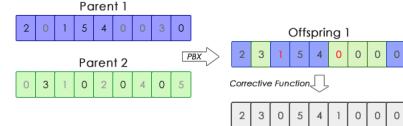


Fig. 2. PBX applied to the taxi sharing problem.

tournament selection and they replace the worst individuals in the destination deme.

IV. EXPERIMENTAL ANALYSIS

This section reports the experimental analysis of the proposed parallel micro EA to solve a set of realistic instances of the taxi sharing problem.

A. Development and execution platform

The proposed EA was implemented in the C++ programming language, using Malva [17]. The experimental evaluation was performed on a Dell Power Edge server, Quad-core Xeon E5430 processor at 2.66GHz, 8 GB RAM and Gigabit Ethernet, from the Cluster FING high performance computing facility (Universidad de la República, Uruguay, website <http://www.fing.edu.uy/cluster>) [18].

B. Problem instances

In order to evaluate the proposed EA, a group of real instances were generated. A specific methodological approach for the generation of realistic instances of the problem was used, taking into account the problem restrictions and using services available to gather information about taxi demands, maps, and fares, including:

- *Taxi Query Generator (TQG)*, a tool that uses information from a database of taxi trajectories obtained from GPS devices installed for a week in 10,357 taxis in the city of Beijing, to generate realistic taxi demands. The data are a subset of those used by Ma et al. [10]. TQG produces a list of origin/destination coordinates for individual trips. In order to design useful instances for the problem addressed in this article, we developed a script that groups those trips that originate in nearby locations, thus outputs are according to the one origin-to-many destinations problem formulation.
- The *TaxiFareFinder* interface [19], which was used to obtain the cost matrix for each generated instance of the problem, along with the prices corresponding to the minimum fare. Each pair of coordinates for a

given instance of the problem is sent to the interface TaxiFareFinder to get the cost between each point.

Following the proposed approach, we created a benchmark set of **24** realistic instances of the taxi sharing problem, with different dimensions: *small* (10–25 taxi requests), medium (25–40 requests), large (40–55 requests), and very large (55–70 requests), to use in the experimental evaluation.

C. Parameter tuning

EAs are stochastic search methods, thus a parameter setting analysis is mandatory. For this purpose, a set of 5 medium-size problem instances (different to the evaluation ones, to avoid biased results) was generated using the aforementioned method. After an initial evaluation, the micro-population size was set to 15. The migration operator was set to select 2 individuals from each deme using a tournament selection and send them to the next deme, where they replace the worst individuals. The migration operator is applied every 500 generations. The parameter tuning focused on two key aspects of the EA: crossover (p_C) and mutation probability (p_M). Three candidate values were picked for each one: $p_C \in \{0.6, 0.75, 0.95\}$ and $p_M \in \{0.001, 0.01, 0.1\}$.

For all 9 combinations of candidate values, 20 independent executions were performed for each problem instance, with 100.000 generations in each run. Representative results are shown in Figure 3, reporting the average cost obtained for each combination of candidate values. Because this is a minimization problem, those settings that achieve lower cost values are preferred. The parameter setting results suggest that using $p_C = 0.75$ and $p_M = 0.1$ allows computing the best results for the problem instances solved.

D. Experimental evaluation

The experimental analysis focused on both the quality of the solutions and the performance of the proposed parallel EA.

1) Comparison of initialization strategies: the two strategies used to initialize the population were comparatively studied on the whole set of problem instances, by analyzing the cost values obtained after 100.000 generations using 24 cores to decide which initialization performs the best. We applied the Kolmogorov-Smirnov statistical test with a significance level of 0.05 and we found that the results distributions do not follow a normal distribution. Afterward, the Kruskal-Wallis test was applied to analyze the results distributions for each initialization method.

Table I reports the best, average, and standard deviation cost values obtained when using the greedy and the random initialization in 20 independent executions performed for each problem instance. Overall, both initialization methods computed the same average cost for two small problem instances, the greedy method found the best average in 11 instances, and the random one in 9 instances. Instances where the greedy initialization performed the best, with 95% statistical confidence ($p_{value} < 0.05$ on the Kruskal-Wallis test) are marked in bold.

From the results in Table I, we can assess with statistical confidence that the greedy initialization allows computing better solutions than the random method, especially for large problem instances. Thus, this method was selected for the rest of the experimental evaluation.

TABLE I. INITIALIZATION STRATEGIES: COMPARATIVE RESULTS

instance	greedy initialization		random initialization	
	best	avg. \pm std	best	avg. \pm std
(10–25)	#1 125.5	125.5 \pm 0.0	125.5	125.5 \pm 0.0
	#2 168.8	168.8 \pm 0.0	168.8	168.8 \pm 0.0
	#3 191.0	191.2 \pm 0.5	191.0	191.0 \pm 0.0
	#4 215.5	215.6 \pm 0.1	215.5	215.5 \pm 0.0
	#5 297.5	298.4 \pm 0.3	297.5	300.9 \pm 3.5
	#6 246.1	252.2 \pm 2.4	244.1	245.9 \pm 2.1
(25–40)	#1 336.5	344.5 \pm 5.6	336.5	340.5 \pm 3.9
	#2 320.2	323.1 \pm 3.4	319.1	324.3 \pm 3.6
	#3 795.5	801.2 \pm 4.2	796.1	803.7 \pm 4.8
	#4 351.2	357.4 \pm 3.4	351.8	358.6 \pm 3.0
	#5 436.8	443.7 \pm 3.8	435.6	443.3 \pm 5.7
	#6 359.3	367.0 \pm 5.0	360.9	375.2 \pm 6.1
(40–55)	#1 415.0	429.9 \pm 7.1	412.6	420.0 \pm 3.7
	#2 304.9	319.8 \pm 7.5	306.4	322.0 \pm 6.5
	#3 416.2	425.0 \pm 4.3	417.2	431.2 \pm 7.1
	#4 362.1	367.7 \pm 3.2	361.6	367.0 \pm 3.3
	#5 440.3	446.3 \pm 2.6	437.6	445.2 \pm 4.8
	#6 553.5	562.1 \pm 4.3	554.2	566.5 \pm 7.6
(55–70)	#1 626.3	637.7 \pm 3.6	630.7	643. \pm 6.4
	#2 517.2	524.8 \pm 4.5	509.0	520.8 \pm 7.7
	#3 490.0	498.4 \pm 3.5	487.3	501.6 \pm 7.8
	#4 736.8	744.9 \pm 6.1	729.0	742.1 \pm 8.0
	#5 548.7	560.6 \pm 4.7	551.9	564.3 \pm 7.0
	#6 1412.6	1424.6 \pm 6.1	1418.7	1430.0 \pm 6.1

2) Comparison against the sequential EA: Table II reports the average costs obtained for each instance dimension, using the p μ EA—with 8, 16, and 24 cores—and the sequential EA. From the results, it is clear that using more generations leads to better solutions. In addition, p μ EA has a good scalability behavior, as cost results improve when using additional cores, up to 1.6% when comparing p μ EA–24 cores vs. p μ EA–8 cores on a large problem instance.

The results in Table II demonstrate that p μ EA outperforms the sequential EA on every instance in the test set, with significant cost improvements (according to the Kruskal-Wallis test). Over all instances, the average cost improvements computed by p μ EA over the sequential EA were 4.3%. In the best case, instance #1 in the set of large problem instances, an improvement of up to 8.5% was achieved against the average cost obtained by the sequential EA. The full experimental results are available at www.fing.edu.uy/inco/grupos/cecal/hpc/AG-Taxi.

TABLE II. COMPARISON OF P μ EA USING DIFFERENT NUMBERS OF SUBPOPULATIONS AND THE SEQUENTIAL EA

instances	algorithm	generations				
		10000	25000	50000	75000	100000
(10–25)	p μ EA–8 cores	212.2	211.2	210.1	209.7	209.5
	p μ EA–16 cores	210.6	209.7	209.2	208.9	208.8
	p μ EA–24 cores	209.9	209.3	208.9	208.7	208.6
	sequential EA	219.3	217.6	216.5	215.6	215.2
(25–40)	p μ EA–8 cores	454.9	449.3	447.5	446.9	446.5
	p μ EA–16 cores	449.5	444.0	442.2	441.6	440.7
	p μ EA–24 cores	448.0	442.7	441.0	440.0	439.5
	sequential EA	475.9	468.7	466.3	465.2	464.6
(40–55)	p μ EA–8 cores	454.2	438.8	433.6	432.2	431.4
	p μ EA–16 cores	450.9	435.4	428.9	427.6	426.9
	p μ EA–24 cores	448.7	432.4	427.6	425.8	425.1
	sequential EA	475.7	456.6	449.3	447.2	446.6
(55–70)	p μ EA–8 cores	788.2	759.3	745.8	741.4	739.5
	p μ EA–16 cores	781.5	752.8	741.0	736.7	734.9
	p μ EA–24 cores	777.9	750.9	737.9	733.6	731.8
	sequential EA	815.0	782.2	766.4	762.7	761.1

3) Comparison against an intuitive greedy algorithm: A greedy strategy was implemented in order to compare the performance of p μ EA against a traditional intuitive heuristic.

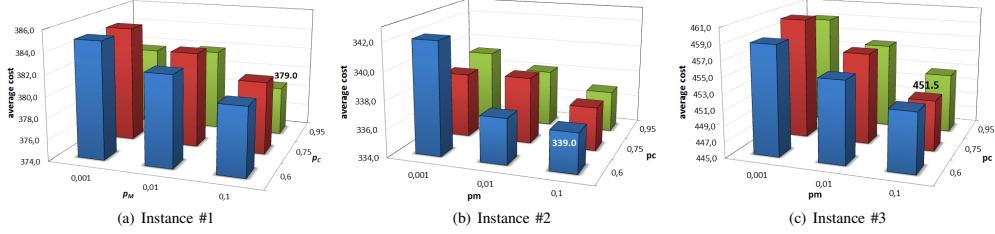


Fig. 3. Average cost obtained with different parameter configurations on three different problem instances

The greedy method (Algorithm 2) sequentially adds the passenger whose destination is closer to the origin in a taxi, until the taxi is full. In the case where the cost of adding one passenger to the current taxi is greater than the one obtained by assigning a new taxi to serve that passenger request, the current taxi is full, and a new one is formed. The algorithm ends when every passenger is assigned to a taxi. This method emulates an intuitive approach to solve the problem by taking local optimum decisions [11]. A similar strategy can be expected from a group of human users trying to solve the problem.

Algorithm 2 Intuitive greedy algorithm for taxi sharing

```

taxi = 1; counter = 0; {Init}
while passengers to assign do
    if (counter == 0) then
        p1 = nearestDestinationfromOrigin();
        addPassenger(p1,taxi,solution);
        counter++;
    else if (counter  $\geq C_{MAX}$ ) then
        counter = 0; {Current taxi is full}
        taxi++;
    else
        p2 = nearestLocation(p1);
        if (cost(p1,p2)  $\leq$  cost(origin,p2) + MF) then
            addPassenger(p2,taxi,solution);
            p1 = p2;
            counter++;
        else
            taxi++; {Start a new taxi}
            counter = 0;
        end if
    end if
end while
return solution;

```

Since the greedy algorithm is very fast, it is important to evaluate how long it takes the $p\mu$ EA to outperform it. Figure 4 reports the average time $p\mu$ EA need to outperform the greedy solution by 5%, 10%, 15% and 20% (in the best case for each instance). Additionally, the average overall improvement and average time to reach 100.000 generations is displayed for each problem instance (upper row, in red).

The greedy algorithm executes in a negligible amount of time, but the results in Figure 4 show that $p\mu$ EA is able to improve the greedy by 5% almost instantaneously, and improvements of about 10% are computed in most small/medium/large cases in a few seconds. $p\mu$ EA also computes solutions with larger improvements over greedy (up to 36.1%), in execution times that strongly depends on the instance and dimension.

In the best case for each problem dimension, $p\mu$ EA improved the results computed using the greedy algorithm by **36.1%** in small instance #2, **27.4%** in medium instance #5, **27.2%** in large instance #4, and **25.0%** in very large instance #3. The full results of the comparison are available at www.fing.edu.uy/inco/grupos/cecal/hpc/AG-Taxi

The previous results demonstrate that the proposed $p\mu$ EA is an accurate and very efficient tool for cost optimization in the taxi sharing problem. The reduced execution times to compute significant improvements over traditional techniques make $p\mu$ EA an appropriate method for on-line optimization, to be used in application-oriented websites such as the one we are currently deploying in <http://www.mepaseaste.uy>.

V. CONCLUSIONS AND FUTURE WORK

This work has presented the design and implementation of a parallel micro evolutionary algorithm for the one-origin-to-many-destinations taxi pooling problem.

Vehicle sharing is an interesting topic nowadays, mainly because it impacts in economic cost and environmental protection, and the scientific community has been studying diverse variants of the car pooling problem in the last years. In this work, we study a specific variant of the taxi sharing problem and propose a parallel micro EA to solve it.

The proposed algorithm was conceived to provide accurate and efficient solutions for the problem, in order to improve the previous results computed by the sequential EA presented in [12]. By using a distributed model and micro populations, the proposed $p\mu$ EA provides an improved search pattern that allows solving the planning problem in online mode.

Indeed, in the experimental analysis performed over a benchmark set of 24 realistic problem instances generated using real GPS data from taxis in the city of Beijing, $p\mu$ EA was able to compute significant improvements over both the sequential EA and an intuitive greedy algorithm to solve the problem. Regarding the cost of solutions, improvements up to 8.5% (4.3% in average) were achieved over the sequential EA and up to 36.1% (18.2% in average) over the greedy method. Furthermore, significant improvements are computed in very fast execution times, making the proposed optimization technique an appropriate tool for online taxi sharing planning

The main lines for future work are focused on including other user-oriented information in the problem formulation, such as online traffic data and taxi availability, in order

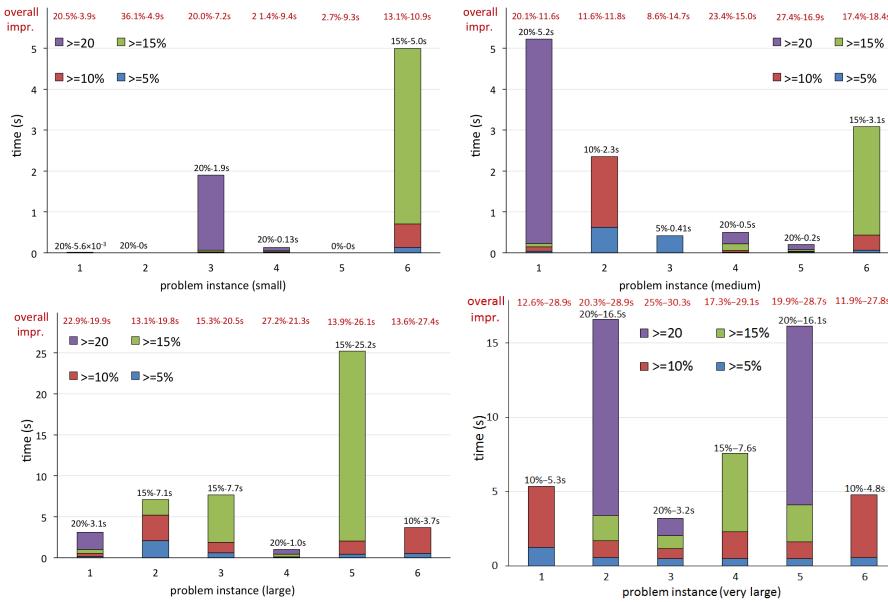


Fig. 4. pμEA comparison against the greedy algorithm: best and average cost improvements and average execution times.

to provide a more realistic planning. The proposed pμEA will be incorporated in our website for taxi planning. Other optimization techniques, including multiobjective EAs should also be evaluated to solve the problem.

ACKNOWLEDGEMENT

The research reported in this article was partly supported by ANII and PEDECIBA, Uruguay.

REFERENCES

- [1] N. Fellows and D. Pitfield, "An economic and operational evaluation of urban car-sharing," *Transportation Research Part D: Transport and Environment*, vol. 5, no. 1, pp. 1–10, 2000.
- [2] E. Ferrari, R. Manzini, A. Pareschi, A. Persona, and A. Regattieri, "The car pooling problem: Heuristic algorithms based on savings functions," *Journal of Advanced Transportation*, vol. 37, pp. 243–272, 2003.
- [3] R. Katzev, "Car sharing: A new approach to urban transportation problems," *Analyses of Social Issues and Public Policy*, vol. 3, no. 1, pp. 65–86, 2003.
- [4] "Ride-Sharing Services Grow Popular in Europe," The New York Times, By Eric Pfanner, Published: Oct. 1, 2012, [Online 02-2014] <http://www.nytimes.com/2012/10/01/technology/ride-sharing-services-grow-popular-in-europe.html>.
- [5] A. Letchford, R. Eggle, and J. Lysgaard, "Multistars, partial multistars and the capacitated vehicle routing problem," *Mathematical Programming*, vol. 94, no. 1, pp. 21–40, 2002.
- [6] S. Nesmachnow, "Metaheuristics as soft computing techniques for efficient optimization," in *Encyclopedia of Information Science and Technology*, M. Khosrow-Pour, Ed., IGI Global, 2014, pp. 1–10.
- [7] "How to Split a Shared Cab Ride? Very Carefully, Say Economists," The Wall Street Journal, Dec. 8, 2005, [Online 06-2014] <http://online.wsj.com/news/articles/SB113279169439805647>.
- [8] C.-L. Xin and W.-M. Ma, "Scheduling for on-line taxi problem on a real line and competitive algorithms," in *Proc. of the Int. Conf. on Machine Learning and Cybernetics*, 2004, pp. 3078–3083.
- [9] E. Koutsoupias, "The k-server problem," *Computer Science Review*, vol. 3, no. 2, pp. 105–118, 2009.
- [10] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *IEEE 29th Int. Conf. on Data Engineering*, 2013, pp. 410–421.
- [11] C.-C. Tan and C.-Y. Chen, "Heuristic algorithms for the dynamic taxipooling problem based on intelligent transportation system technologies," in *4th Int. Conf. on Fuzzy Systems and Knowledge Discovery*, 2007, pp. 590–595.
- [12] G. Fagundez, R. Massobrio, and S. Nesmachnow, "Resolución en línea del problema de viajes compartidos en taxis usando algoritmos evolutivos," in *Conferencia Latinoamericana de Informática*, 2014.
- [13] T. Bäck, D. Fogel, and Z. Michalewicz, Eds., *Handbook of evolutionary computation*. Oxford University Press, 1997.
- [14] E. Alba, G. Luque, and S. Nesmachnow, "Parallel metaheuristics: Recent advances and new trends," *Int. Transactions in Operational Research*, vol. 20, no. 1, pp. 1–48, 2013.
- [15] S. Nesmachnow, H. Cancela, and E. Alba, "A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling," *Applied Soft Computing*, vol. 12, no. 2, pp. 626–639, 2012.
- [16] S. Nesmachnow and S. Iturriaga, "Multiobjective grid scheduling using a domain decomposition based parallel micro evolutionary algorithm," *Int. Journal of Grid Computing*, vol. 4, pp. 70–84, 2013.
- [17] "The Malva Project: A framework for computational intelligence in C++," [Online 05-2014] <https://github.com/themalvaproject>.
- [18] S. Nesmachnow, "Computación científica de alto desempeño en la Facultad de Ingeniería, Universidad de la República," *Revista de la Asociación de Ingenieros del Uruguay*, vol. 61, pp. 12–15, 2010.
- [19] "TaxiFareFinder API," [Online 05-2014] <http://www.taxifarefinder.com>.

Planificación multiobjetivo de viajes compartidos en taxis utilizando un micro algoritmo evolutivo paralelo

Renzo Massobrio, Gabriel Fagúndez y Sergio Nesmachnow

Resumen—Este trabajo presenta la aplicación de un micro algoritmo evolutivo paralelo para resolver una variante multiobjetivo del problema de planificación de viajes compartidos en taxis. Los objetivos considerados en la variante abordada del problema son el costo total del viaje y la demora en llegar a destino por parte de cada pasajero. Esta versión del problema de viajes compartidos en taxis contempla situaciones realistas con alta aplicabilidad en la práctica, y toma en cuenta los criterios más considerados por los usuarios. El problema se resuelve con un micro algoritmo evolutivo paralelo que utiliza el modelo de subpoblaciones distribuidas y una estrategia de descomposición de dominio para contemplar las diferentes ponderaciones de los objetivos. El análisis experimental realizado sobre un conjunto de instancias reales del problema muestra que el algoritmo propuesto es capaz de encontrar significativas mejoras en ambos objetivos cuando se lo compara con estrategias ávidas intuitivas para resolver el problema.

Palabras clave—planificación multiobjetivo, viajes compartidos, algoritmos evolutivos paralelos

I. INTRODUCCIÓN

Los viajes compartidos en automóvil, conocidos bajo el término anglosajón *car pooling*, han captado el interés del público en los últimos años [1]. Compartir vehículos con personas que se dirigen a destinos cercanos, genera beneficios no solo en el plano económico, sino también en el ecológico, a niveles individuales y colectivos. Por otra parte, el *car pooling* permite minimizar los costos de traslado y la cantidad de automóviles en las rutas, reduciendo así la polución y contribuyendo a minimizar el impacto ambiental del transporte, un problema capital en las últimas décadas en las grandes ciudades [2, 3].

Los automóviles con taxímetro (taxis) constituyen un medio de transporte rápido y confortable. Sin embargo, es frecuente que los taxis no se utilicen a capacidad completa. Por este motivo, se puede aplicar el concepto de viaje compartido a los taxis de forma de obtener similares beneficios. Desde el punto de vista de los usuarios, minimizar los costos es un claro objetivo al compartir vehículos con otras personas, aunque puede no ser el único. En ciertas situaciones, minimizar la demora puede ser más gravitante que únicamente minimizar el costo de traslado. Costo y demora son objetivos contrapuestos y por tal motivo resulta importante atacar el problema multiobjetivo, hallando soluciones con diferentes valores de compromiso entre costo y demora.

E-mail: {renzom,gabrielf,sergion}@fing.edu.uy .

En la actualidad existen servicios web que proporcionan soluciones al problema de planificación de viajes compartidos utilizando heurísticas simples (sin contemplar la inclusión de técnicas de inteligencia computacional), y algunos trabajos de investigación que han resuelto variantes del problema de planificar viajes en taxis utilizando métodos específicos. El problema de planificación de viajes compartidos es NP-difícil [4]. Para resolver instancias con dimensiones realistas es necesario utilizar heurísticas o metaheurísticas [5] que permitan hallar soluciones de calidad aceptable en tiempos razonables, especialmente si se contempla la posibilidad de calcular resultados en tiempo real para ofrecer el servicio a los usuarios.

En esta línea de trabajo, este artículo presenta un micro algoritmo evolutivo paralelo aplicado a resolver la variante multiobjetivo del problema de planificación de viajes compartidos en taxis que considera la minimización simultánea del costo del viaje y la demora en llegar a destino por parte de cada pasajero. Las principales contribuciones del trabajo consisten en: i) la implementación de un micro algoritmo evolutivo paralelo que utiliza el modelo de subpoblaciones distribuidas y una estrategia de descomposición de dominio para contemplar las diferentes ponderaciones de los objetivos, pensado para resolver eficientemente el problema y ii) el análisis experimental realizado sobre un conjunto de instancias realistas del problema, que demuestra que el algoritmo propuesto es capaz de encontrar planificaciones con significativas mejoras en ambos objetivos cuando se lo compara con estrategias ávidas intuitivas para resolver el problema.

El artículo se organiza del modo que se describe a continuación. La Sección II presenta la versión multiobjetivo del problema de planificación de viajes compartidos en taxis y su formulación matemática. El micro algoritmo evolutivo paralelo ideado para resolver el problema se introduce en la Sección III. Una revisión de los principales trabajos relacionados se ofrece en la Sección IV. Los detalles de diseño e implementación del algoritmo propuesto se describen en la Sección V. La Sección VI reporta el análisis experimental realizado sobre escenarios realistas del problema y una discusión sobre los principales resultados obtenidos. Finalmente, la Sección VII presenta las conclusiones del trabajo y formula las principales líneas de trabajo actual y futuro.

II. EL PROBLEMA DE PLANIFICACIÓN DE VIAJES COMPARTIDOS EN TAXIS (VERSIÓN MULTIOBJETIVO)

Esta sección presenta una descripción del problema de viajes compartidos en taxis, y su formulación matemática como problema de optimización multiobjetivo.

A. Descripción del problema

El problema a resolver (*taxi pooling*) modela una realidad donde un determinado número de personas, ubicadas en un mismo lugar de origen, deciden viajar hacia diferentes destinos utilizando taxis. El problema de optimización relacionado consiste en determinar el número apropiado de taxis y la distribución adecuada de pasajeros, buscando minimizar dos objetivos relevantes para los usuarios: el *costo del viaje* y la *demora percibida*. La distribución de pasajeros está restringida a la cantidad máxima de personas que pueden viajar en un mismo taxi. Se contemplan diferentes tamaños de vehículos, y una cantidad fija de vehículos de cada una de las capacidades.

Aunque el problema fue originalmente concebido para modelar el transporte en taxis en la ciudad de Montevideo, Uruguay, ni su formulación ni el algoritmo propuesto para su resolución restringen su aplicabilidad a otros escenarios, tal como se demuestra en el análisis experimental reportado en la Sección VI. Los costos asociados al transporte están modelados con una función realista que considera la tarificación basada en distancias recorridas por cada taxi. Este modelo corresponde a considerar escenarios con tráfico fluido, sin grandes embotellamientos. La incorporación de datos de tráfico en tiempo real está propuesta como una de las principales líneas de trabajo actual y futuro.

Respecto al problema, las siguientes consideraciones deben ser tenidas en cuenta:

- Cada taxi puede trasladar a un número limitado de pasajeros dependiendo de su capacidad. La cantidad de taxis disponibles de cada tipo es dada como entrada del algoritmo.
- El número máximo de taxis a utilizar para un escenario con N pasajeros es N , en el caso particular de que cada pasajero viaje en un vehículo distinto.
- El costo de un taxi está dado por la suma del costo inicial (conocido popularmente como “bajada de bandera”), más el costo determinado por la distancia recorrida desde el origen al destino. No se consideran otros posibles costos tales como esperas, propinas o cargos extras por equipaje.
- Cada pasajero tiene asociada una tolerancia que indica el tiempo que éste está dispuesto a demorar por sobre el tiempo que le demandaría un viaje directo desde el origen a su destino. La tolerancia de cada pasajero también es dada como entrada al algoritmo.

B. Formulación matemática

La formulación matemática del problema de viajes compartidos en taxis se presenta a continuación.

Dados los siguientes elementos:

- Un conjunto de pasajeros $P = \{p_1, p_2, \dots, p_N\}$; que parten de un punto geográfico común O y desean trasladarse hacia un conjunto de destinos potencialmente diferentes $D = \{d_1, d_2, \dots, d_N\}$.
- Un conjunto de taxis $T = \{t_1, t_2, \dots, t_M\}$; con $M \leq N$, con capacidades (pasajeros que pueden transportar) $K = \{K_1, K_2, \dots, K_M\}$.
- Una función C que determina cuántos pasajeros hacen uso de un taxi en un determinado viaje, siendo $C_i \leq K_i$.
- Una matriz MC de dimensión $(N+1) \times (N+1)$ que especifica el costo entre cada uno de los puntos geográficos involucrados en el problema (un punto origen y N puntos destino).
- Una matriz MT de dimensión $(N+1) \times (N+1)$ que especifica los tiempos de demora para un viaje entre cada uno de los puntos geográficos involucrados en el problema.
- Un vector T de largo N donde T_i es el tiempo extra que el usuario i tolera por encima del tiempo que tarda en ir directamente del origen a su destino.
- Una función de *costo total*, determinada por el valor B constante (costo inicial, “bajada de bandera”) y la suma para todos los taxis de los costos entre el origen, los destinos intermedios y el destino final:

$$CT = \sum_{t_i} (B + \sum_{j=1}^{C(t_i)} MC[d_{j-1}, d_j]) \quad (1)$$

- Una función de *demora total*, determinada por la diferencia entre el tiempo tolerado y el tiempo real que cada pasajero tarda en llegar a su destino:

$$DT = \sum_{t_i} \left(\sum_{j=1}^{C(t_i)} \left(T_i + MT[0, d_j] - \left(\sum_{h=1}^j (MT[h-1, h]) \right) \right) \right) \quad (2)$$

El problema consiste en hallar una *planificación de viajes*, es decir una función $f : P \rightarrow T$ para transportar los N pasajeros en L taxis ($L \leq M$) que determine cómo asignar pasajeros a taxis y el orden en que serán trasladados a los respectivos destinos, minimizando la función objetivo definida por la agregación lineal ponderada de costo y demora $FO = W_C \times CT + W_D \times DT$, siendo $W_C = 1 - W_D$.

III. COMPUTACIÓN EVOLUTIVA

A. Algoritmos evolutivos

Los Algoritmos Evolutivos (AE) son técnicas estocásticas que emulan el proceso de evolución natural de las especies para resolver problemas de optimización, búsqueda y aprendizaje [6]. En los últimos

30 años, los AE han sido exitosamente aplicados para resolver problemas de optimización subyacentes a problemas complejos del mundo real en múltiples áreas de aplicación [5].

Un AE es una técnica iterativa (cada iteración se denomina *generación*) que aplica operadores estocásticos sobre un conjunto de individuos (la *población*). Inicialmente, la población se genera aleatoriamente, o aplicando una heurística específica para el problema a resolver. Cada individuo en la población codifica una solución tentativa al problema estudiado, y tiene un valor de *fitness*, dado por una función de evaluación que determina la adecuación del individuo para resolver el problema. El AE busca mejorar el fitness de los individuos en la población, mediante la aplicación de *operadores evolutivos*, como la *recombinación* de partes de dos individuos y la *mutación* aleatoria de su codificación, guiando al AE a soluciones de mayor calidad.

El criterio de parada usualmente involucra un número determinado de generaciones, una cota de calidad sobre el mejor valor de fitness hallado, o la detección de una situación de convergencia. Políticas específicas se utilizan para seleccionar los individuos que participan en la recombinación (la *selección*) y para determinar cuáles nuevos individuos serán insertados en la población en cada nueva generación (el *reemplazo*). El AE retorna la mejor solución hallada en el proceso iterativo, tomando en cuenta la función de fitness considerada para el problema.

B. Algoritmos evolutivos paralelos

Las implementaciones paralelas se han popularizado como un mecanismo para mejorar el desempeño de los AE. Dividiendo la población o el cálculo de fitness entre varios elementos de procesamiento, los AE paralelos permiten abordar problemas de grandes dimensiones y difíciles de resolver, hallando resultados de buena calidad en tiempos razonables.

Los modelos paralelos de los AE utilizados en este estudio corresponden a la categoría de subpoblaciones distribuidas [7]. La población original se divide en varias subpoblaciones (islas), cada una de las cuales ejecuta un AE secuencial donde los individuos solo son capaces de interactuar con otros en su misma isla. Se define un operador adicional de *Migración* que permite el intercambio ocasional de individuos entre islas, introduciendo una nueva fuente de diversidad. El Algoritmo 1 presenta un esquema paralelo para un AE, donde *emigrantes* denota al conjunto de individuos a intercambiar con otra isla, seleccionados de acuerdo a una política determinada por *seleccMigracion*. El operador de migración intercambia los individuos entre islas de acuerdo a una topología de interconexión, que generalmente es un anillo unidireccional. La *CondicionMigracion* determina cuándo se lleva a cabo el intercambio de individuos.

Algorithm 1 Esquema de un AE Paralelo.

```

1: inicializar( $P(0)$ )
2:  $t \leftarrow 0$  {contador de generación}
3: evaluar( $P(0)$ )
4: mientras no se cumple CriterioParada hacer
5:   padres  $\leftarrow$  selección( $P(t)$ )
6:   hijos  $\leftarrow$  operadores de variación(padres)
7:   nueva población  $\leftarrow$  reemplazo(hijos,  $P(t)$ )
8:   evaluar( $P(t)$ )
9:    $t++$ 
10:   $P(t) \leftarrow$  nueva población
11:  si CondicionMigración entonces
12:    emigrantes  $\leftarrow$  seleccMigracion( $P(t)$ )
13:    inmigrantes  $\leftarrow$  migración(emigrantes)
14:    insertar(inmigrantes,  $P(t)$ )
15:  fin si
16:  fin mientras
17: retornar mejor solución hallada

```

C. El algoritmo μ -MOEA/D

Los AE paralelos mejoran su eficiencia computacional al trabajar con subpoblaciones que limitan las interacciones entre individuos. Sin embargo, los AE suelen perder diversidad cuando trabajan con poblaciones pequeñas, convergiendo prematuramente a soluciones lejanas al óptimo del problema.

Varias alternativas se han propuesto para mitigar el efecto de la pérdida de diversidad en los AE. En este trabajo se aplica un micro algoritmo paralelo (μ -MOEA/D) para la resolución del problema de viajes compartidos en taxis. La idea de un micro algoritmo genético fue presentada por Coello y Pulido [8], basándose en los estudios teóricos de Goldberg sobre la convergencia al óptimo de un problema de optimización cuando se utiliza un AE con tan solo tres individuos en la población, en caso de contar con un operador apropiado para introducir diversidad ante convergencia prematura.

El algoritmo μ -MOEA/D propuesto en este trabajo combina dos características de diseño: (1) un modelo paralelo de subpoblaciones distribuidas utilizando micropoblaciones y migración, que se utiliza como operador para proveer diversidad a la búsqueda y (2) un enfoque de descomposición de dominio similar al aplicado en el algoritmo MOEA/D [9] para resolver un problema de optimización con dos objetivos. Cada subpoblación en μ -MOEA/D se enfoca en resolver un problema de optimización específico, aplicando un esquema de agregación lineal para los objetivos del problema, pero utilizando diferentes pares de pesos para ponderar las funciones costo y demora. Este enfoque permite muestrear el frente de Pareto del problema, a pesar de que el algoritmo no utilice un esquema explícito de asignación de fitness basado en dominancia de Pareto. El enfoque ha sido aplicado previamente al algoritmo CHC [10].

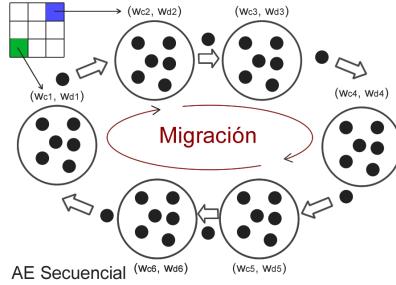


Fig. 1. Esquema del algoritmo $p\mu$ -MOEA/D

IV. TRABAJOS RELACIONADOS

Diferentes variantes del *taxi-pooling* han sido estudiadas en la literatura. Ma et al. [11] diseñaron un sistema dinámico para compartir taxis, combinando un método de búsqueda para encontrar taxis candidatos a satisfacer un pedido, y un algoritmo de planificación que inserta el viaje en el itinerario del taxi que incurre en la menor distancia adicional al atender ese pedido. Usando una base de datos de trayectorias GPS generadas por 33.000 taxis durante 3 meses en la ciudad de Beijing, el sistema alcanzó un ahorro en la distancia de un 13 %.

Contemplando el objetivo de costo, Tao et al. [12] propusieron dos heurísticas ávidas para optimizar los costos en viajes compartidos de taxis. Una prueba piloto realizada en Taipei con 10 taxis y 798 pasajeros mostró resultados promedio de 60.3 % en el porcentaje de éxito en los emparejamientos. Los resultados sobre la mejora en el consumo de combustible se reportan en términos absolutos, siendo difícil utilizarlos como base para una comparación.

Abordando el problema multiobjetivo, Yeqian et al. [13] propusieron un algoritmo de recocido simulado enfocado en minimizar el costo operativo de los taxis y maximizar la satisfacción de los usuarios. La formulación del problema tiene en cuenta la distancia real recorrida, el tiempo de recorrido extra de los taxis y el tiempo de espera de los clientes. El algoritmo propuesto redujo la distancia recorrida en 19 % y aumentó 66 % la cantidad de taxis disponibles respecto a una situación en que no se comparten taxis.

En nuestro grupo de trabajo se ha abordado la resolución de la variante mono-objetivo del problema de planificación de viajes compartidos en taxis, utilizando AE secuenciales [14] y AE paralelos [15].

El análisis de trabajos relacionados muestra que el problema de viajes compartidos en taxis es de interés en la actualidad, debido a su impacto en el medioambiente y en la economía. Sin embargo, las soluciones centradas en el usuario son escasas en la literatura, en especial aquellas que buscan optimizar más de un objetivo simultáneamente. Por lo tanto, es posible contribuir desarrollando aplicaciones que apliquen inteligencia computacional enfocadas en maximizar

el ahorro y la satisfacción de los clientes que hacen uso de viajes compartidos en taxis.

V. DISEÑO E IMPLEMENTACIÓN DE $p\mu$ -MOEA/D

Esta sección presenta los detalles de diseño e implementación de $p\mu$ -MOEA/D aplicado al problema.

A. Codificación y función de fitness

Las soluciones se representan como tuplas que contienen a los enteros entre 1 y N , representando a los pasajeros, y $N - 1$ ceros utilizados para separar pasajeros asignados a diferentes taxis. El orden en que se visitan los destinos es el especificado en la secuencia. La Figura 2 muestra un ejemplo de codificación de la solución para una instancia con 5 pasajeros.



Fig. 2. Ejemplo de la codificación utilizada.

El esquema de codificación tiene las siguientes restricciones: i) cada entero debe aparecer una única vez en la tupla, ii) los ceros consecutivos tienen el mismo significado que un único cero, iii) las secuencias de dígitos distintos de cero deben ser de largo menor o igual a la máxima capacidad de vehículo disponible hasta el momento, de acuerdo al vector de capacidades de la flota de vehículos.

El fitness queda definido por la función objetivo descripta en II-B, utilizando $W_c = [0; \frac{1}{\#islas}; 1]$ y $W_d = 1 - W_c$.

B. Operadores evolutivos

Dado que la codificación utilizada tiene características y restricciones particulares, se utilizan operadores evolutivos adaptados al problema.

Inicialización de la población: Se utilizaron dos métodos de inicialización. La *inicialización aleatoria* usa un algoritmo constructivo que ubica aleatoriamente los números del 1 al N en una tupla creada con $2N - 1$ ceros. La *inicialización ávida* inserta en la población dos individuos generados con las soluciones halladas por algoritmos ávidos para costo y demora (ver Sección VI), y completa la población con individuos generados aplicando un número aleatorio de perturbaciones (i.e. intercambios de dos elementos), a las soluciones obtenidas por ambos algoritmos ávidos. Ambas estrategias de inicialización se estudian en la evaluación experimental.

Proceso de corrección: Ambas estrategias de inicialización pueden violar alguna de las restricciones definidas para la codificación de las soluciones, por lo cual se aplica una función correctiva para garantizar que las soluciones generadas sean válidas. El

algoritmo de corrección busca secuencias de dígitos distintos de cero de largo mayor a la máxima capacidad de vehículo disponible hasta el momento. Al encontrar una, se busca la primer pareja de ceros consecutivos y se mueve el primer cero hacia una posición aleatoria dentro de la secuencia encontrada, de forma de romper con el grupo inválido de dígitos distintos de cero. Este proceso continúa hasta que se recorre la solución por completo, garantizando que la misma cumple con todas las restricciones.

Selección: se utilizó el operador de *selección por torneo* para otorgar una apropiada presión selectiva a las micropoblaciones. Experimentos iniciales mostraron que la selección proporcional no es capaz de ofrecer suficiente diversidad, conduciendo a convergencia prematura.

Recombinación: se aplicó una variante del operador *Position Based Crossover (PBX)*, explicado en el Algoritmo 2.

Algorithm 2 Esquema de la recombinación PBX

- 1: Seleccionar aleatoriamente varias posiciones del padre 1.
- 2: generar parcialmente el hijo 1, copiando los valores de las posiciones elegidas del padre 1.
- 3: Marcar las posiciones del padre 2 que ya fueron seleccionados en el padre 1.
- 4: Desde el inicio del padre 2, seleccionar secuencialmente el siguiente valor no marcado, e insertarlo en la primera posición libre del hijo.

De forma de cumplir con las restricciones impuestas por la codificación de la solución, se aplica la misma función correctiva utilizada en la inicialización sobre el hijo resultante del cruzamiento PBX. La Figura 3 muestra un ejemplo del cruzamiento PBX para una instancia con 5 pasajeros.

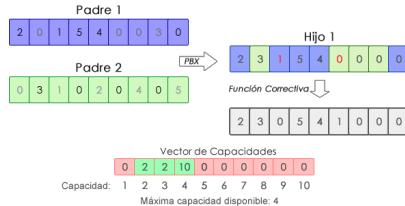


Fig. 3. Ejemplo del operador PBX aplicado.

Mutación: Se aplicó el operador de mutación por intercambio (*Exchange Mutation, EM*), que selecciona e intercambia dos posiciones de la solución y luego aplica la función correctiva para corregir las potenciales soluciones inválidas.

Migración: El operador de migración se aplica asincrónicamente, considerando a las islas conectadas en una topología de anillo unidireccional. Los emigrantes se seleccionan mediante torneo y reemplazan a los peores individuos de la isla destino. La

migración contribuye a mejorar la diversidad, evitando la convergencia prematura del algoritmo.

VI. ANÁLISIS EXPERIMENTAL

Esta sección reporta el análisis experimental de μ -MOEA/D sobre un conjunto de instancias realistas del problema.

A. Plataforma de desarrollo y ejecución

El algoritmo μ -MOEA/D se desarrolló en C++, utilizando Malva [16]. La evaluación experimental se realizó en un HP Proliant DL585, AMD Opteron 6272 a 2.09GHz, 48GB de RAM y Gigabit Ethernet, en la plataforma de cómputo de alto desempeño Cluster FING (Universidad de la República, Uruguay, <http://www.fing.edu.uy/cluster>) [17].

B. Instancias del problema

Para evaluar el algoritmo propuesto, se generó un conjunto de instancias realistas del problema de planificación de viajes compartidos en taxis. Se utilizó un enfoque metodológico específico para generar los escenarios, siguiendo los lineamientos presentados en los trabajos relacionados y utilizando servicios disponibles para recabar información sobre pedidos de taxi, mapas, y tarifas, incluyendo:

- *Generador de Pedidos de Taxis (Taxi Query Generator, TQG)*, una herramienta que utiliza información de una base de datos de trayectorias de taxis, obtenidas con dispositivos GPS en 10.357 taxis de la ciudad de Beijing, para generar pedidos de taxis realistas. Los datos son una muestra de los utilizados por Ma *et al.* [11]. Para obtener instancias aplicables al problema estudiado en este artículo, se implementó un script que agrupa viajes de TQG que comienzan en orígenes cercanos y devuelve instancias del problema de un origen a muchos destinos.
- La interfaz para aplicaciones de *TaxiFareFinder* [18], que se utilizó para obtener la matriz de costos de cada instancia del problema generada, junto a los precios correspondientes a la bajada de bandera. Cada par de coordenadas de una determinada instancia del problema es enviado a la interfaz de TaxiFareFinder para obtener el costo entre cada punto.

Además, se generó un conjunto de instancias para la ciudad de Montevideo, Uruguay, considerando un origen ubicado en un punto de interés de la ciudad y un conjunto de destinos aleatorios en la ciudad. Los orígenes se corresponden con cuatro situaciones particulares: un local bailable, un evento deportivo, un centro de estudios terciario y un centro comercial.

A partir del mapa de cada instancia, se generaron distintas variantes al combinar diferentes valores para el nivel de tolerancia en la demora de cada pasajero, y las capacidades de vehículos disponibles.

C. Ajuste paramétrico

Debido a la cualidad estocástica de los AEs, es necesario ajustar sus parámetros previamente al análisis experimental. Para la configuración paramétrica

se utilizaron cuatro instancias del problema (distintas de las utilizadas en la evaluación experimental para evitar sesgo). Luego de una evaluación inicial, el tamaño de la micro población se fijó en 15 individuos. El operador de migración se fijó de forma de seleccionar 2 individuos de cada isla mediante una selección por torneo, y enviarlos a la isla vecina, donde reemplazan a los peores individuos. El operador de migración se aplica cada 1000 generaciones.

El ajuste paramétrico se focalizó en estudiar las probabilidades de recombinación (p_C , valores candidato 0.6, 0.75 y 0.95) y de mutación (p_M , valores candidato 0.001, 0.01 y 0.1). Se estudiaron todas las combinaciones de valores sobre las cuatro instancias de calibración, realizando 30 ejecuciones independientes de 20000 generaciones en cada caso.

La Figura 4 muestra los valores promedio para las funciones de costo y demora obtenidos en la instancia #2, para cada combinación de parámetros.

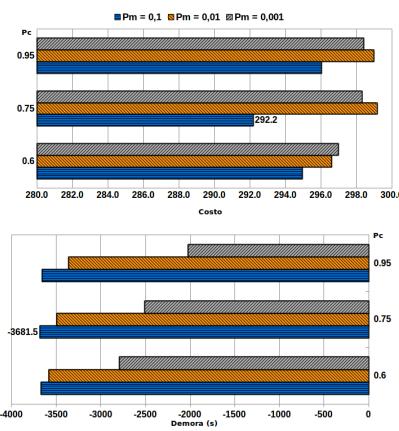


Fig. 4. Configuración paramétrica (instancia #2)

Los resultados del análisis paramétrico indican que los mejores resultados para las instancias estudiadas se obtienen utilizando $p_C=0.75$ y $p_M=0.1$.

D. Evaluación experimental

El análisis experimental se enfocó en evaluar el desempeño de μ -MOEA/D y la calidad de las soluciones alcanzadas. Se utilizaron 39 instancias del problema (27 ubicadas en Beijing y 12 en Montevideo) que corresponden a escenarios de entre 9 y 50 pasajeros. Estas dimensiones son razonablemente mayores a lo que se puede esperar en instancias generadas por usuarios finales de un sistema de planificación en la vida real. Se realizaron 30 ejecuciones independientes de cada instancia, con un criterio de parada de 20000 generaciones.

D.1 Comparación contra dos algoritmos ávidos que resuelven el problema

Un algoritmo ávido es un método de construcción de soluciones que toma decisiones localmente óptimas en cada paso. Es relevante comparar el desempeño de μ -MOEA/D frente a un algoritmo ávido en términos de la calidad de las soluciones alcanzadas y de la eficiencia computacional. Con este propósito, se implementaron dos algoritmos ávidos para resolver el problema de planificación de viajes compartidos en taxis, contemplando los dos objetivos considerados en la función de fitness.

El algoritmo ávido que minimiza el costo se basa en aplicar una técnica de agrupamiento simple e intuitiva. Toma un taxi, y agrega el destino más cercano al taxi actual hasta completarlo (en cuyo caso forma un nuevo taxi), o hasta que todos los pasajeros hayan sido asignados. Una excepción ocurre cuando el costo de añadir un nuevo destino al taxi actual es mayor al costo de asignar un nuevo taxi que transporte únicamente a ese pasajero. En este caso, se forma un nuevo taxi y se “cierra” el anterior. El algoritmo emula una estrategia intuitiva para resolver el problema con un enfoque de agregación de estrategias de decisión localmente óptimas, y se aproxima a lo que un grupo de usuarios humanos pueden idear de manera simple para resolver el problema.

El algoritmo ávido que minimiza la demora de los usuarios aplica una técnica simple e intuitiva que toma los pasajeros más apurados y los ubica en un nuevo taxi, en la primera ubicación. Luego, toma los pasajeros en orden según su urgencia, selecciona el taxi que minimice su demora tomando en cuenta el último pasajero agregado, y añade al pasajero en la última ubicación. Si el taxi alcanza el máximo de pasajeros, el mismo se considera como “cerrado”. Al cerrar un taxi, existe la chance de que alguno de los ya formados pero no cerrados, supere la nueva cota superior de pasajeros según el vector de capacidades. Se utiliza una corrección que en caso de detectar que un taxi supera la nueva cota máxima de pasajeros, lo elimina y crea dos nuevos taxis de capacidad uno con sus dos primeros pasajeros.

E. Resultados y discusión

La Tabla I reporta los valores de costo y demora alcanzados en cada conjunto de instancias por el μ -MOEA/D con las dos inicializaciones estudiadas y por los algoritmos ávidos. Los resultados muestran que el μ -MOEA/D es poco sensible a la estrategia de inicialización elegida, lo que sugiere robustez en la búsqueda. Para el resto de la evaluación experimental se elige la inicialización ávida por alcanzar mejores resultados en instancias de dimensiones grandes.

La Tabla II reporta las mejoras en costo y demora obtenidas por el μ -MOEA/D con inicialización ávida, sobre los algoritmos ávidos para costo y demora. Adicionalmente, se indica entre paréntesis el tiempo necesario en segundos para alcanzar dicha mejora.

Se observa que en el mejor caso se alcanzan mejoras de hasta un 69.0% en el costo y 135.9% en la demora. Adicionalmente, las mejoras se alcanzan en tiempos muy cortos, de apenas un par de segundos en la mayoría de los casos, lo que permite que $p\mu$ -MOEA/D sea útil para resolver instancias reales del problema en línea, como en el caso de la aplicación web instalada en mepaseaste.uy.

La Figura 5 muestra las soluciones no dominadas alcanzadas a las 0, 10000, y 20000 generaciones por $p\mu$ -MOEA/D, en 30 ejecuciones independientes, para una instancia representativa de cada grupo de instancias de prueba. Como referencia, se incluyen las soluciones calculadas por los algoritmos ávidos.

Se puede apreciar que $p\mu$ -MOEA/D permite aproximar correctamente un hipotético frente de pareto real para las distintas instancias de prueba. Es de destacar que, además de alcanzar soluciones de mejor calidad con el correr de las generaciones, $p\mu$ -MOEA/D mantiene bien muestrado tanto los extremos del frente como las soluciones cercanas al vector ideal. Los resultados alcanzados muestran que el modelo de descomposición de datos elegido permite mantener un buen nivel de diversidad a lo largo de la ejecución del algoritmo.

VII. CONCLUSIONES

Este trabajo presentó el diseño e implementación de un micro AE paralelo para resolver el problema multiobjetivo de planificación de viajes compartidos en taxis, desde un origen a múltiples destinos. Los objetivos considerados son el costo total del viaje y la demora percibida por los pasajeros al compartir su viaje (con respecto a viajar en un único taxi).

El micro algoritmo evolutivo paralelo propuesto conforma una precisa solución para el problema abordado. El análisis experimental realizado sobre un conjunto de 39 instancias del problema construidas en base a datos reales tomados de GPS de taxis de la ciudad de Beijing y adaptadas a la ciudad de Montevideo, mostró que los resultados del algoritmo propuesto permiten mejorar significativamente los resultados—hasta un 69.0% en costo y hasta un 135.9% en demora—sobre los calculados empleando algoritmos ávidos que simulan el comportamiento intuitivo de un grupo de usuarios humanos al enfrentarse al problema. Los resultados se obtienen en tiempos de ejecución reducidos, permitiendo su aplicación para la resolución en línea del problema.

Las principales líneas de trabajo futuro se orientan a incorporar datos del tráfico en tiempo real, que permitan estimar con mayor precisión las tarifas y elegir rutas alternativas según corresponda, así como también datos acerca de la disponibilidad de vehículos y sus distintas capacidades. Asimismo, se propone estudiar AE que utilicen un enfoque multiobjetivo explícito para la resolución del problema.

En la página web del proyecto www.fing.edu.uy/inco/grupos/cecal/hpc/AG-Taxi se encuen-

tran publicados los siguientes recursos: i) código fuente de $p\mu$ -MOEA/D y de los algoritmos ávidos; ii) generador de instancias del problema; iii) instancias del problema utilizadas y resultados numéricos detallados de los experimentos realizados.

AGRADECIMIENTOS

Esta investigación ha sido parcialmente financiada por ANII y PEDECIBA, Uruguay.

REFERENCIAS

- [1] N. Fellows and D. Pittfield, “An economic and operational evaluation of urban car-sharing,” *Transportation Research Part D: Transport and Environment*, vol. 5, no. 1, pp. 1–10, 2000.
- [2] E. Ferrari, R. Manzini, A. Pareschi, A. Persona, and A. Regattieri, “The car pooling problem: Heuristic algorithms based on savings functions,” *Journal of Advanced Transportation*, vol. 37, pp. 243–272, 2003.
- [3] R. Katzev, “Car sharing: A new approach to urban transportation problems,” *Analyses of Social Issues and Public Policy*, vol. 3, no. 1, pp. 65–86, 2003.
- [4] A. Letchford, R. Eglese, and J. Lynggaard, “Multistars, partial multistars and the capacitated vehicle routing problem,” *Mathematical Programming*, vol. 94, no. 1, pp. 21–40, 2002.
- [5] S. Nesmachnow, “Metaheuristics as soft computing techniques for efficient optimization,” in *Encyclopedia of Information Science and Technology*, M. Khosrow-Pour, Ed., pp. 1–10, IGI Global, 2014.
- [6] T. Bäck, D. Fogel, and Z. Michalewicz, Eds., *Handbook of evolutionary computation*, Oxford Univ. Press, 1997.
- [7] E. Alba, G. Luque, and S. Nesmachnow, “Parallel metaheuristics: Recent advances and new trends,” *International Transactions in Operational Research*, vol. 20, no. 1, pp. 1–48, 2013.
- [8] C. Coello and G. Pulido, “A micro-genetic algorithm for multiobjective optimization,” in *Proc. of the 1st Int. Conf. on Evolutionary Multi-Criterion Optimization*, London, UK, 2001, pp. 126–140.
- [9] Q. Zhang and H. Li, “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition,” *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, 2007.
- [10] S. Nesmachnow and S. Iturriaga, “Multiobjective grid scheduling using a domain decomposition based parallel micro evolutionary algorithm,” *Int. Journal of Grid and Utility Computing*, vol. 4, pp. 70–84, 2013.
- [11] Y.Zheng S. Ma and O.Wolfson, “T-share: A large-scale dynamic taxi ridesharing service,” in *IEEE 29th Int. Conf. on Data Engineering*, 2013, pp. 410–421.
- [12] C.Tao and C. Chen, “Heuristic algorithms for the dynamic taxipoooling problem based on intelligent transportation system technologies,” in *4th Int. Conf. on Fuzzy Systems and Knowledge Discovery*, 2007, pp. 590–595.
- [13] Li F. Qiu H. Xu Y. Lin, W., “Research on optimization of vehicle routing problem for ride-sharing taxi,” in *8th Int. Conf. on Traffic and Transportation Studies Changsha, China*, 2012.
- [14] R. Massobrio, G. Fagúndez, and S. Nesmachnow, “A parallel micro evolutionary algorithm for taxi sharing optimization,” in *VII ALIO/EURO Workshop on Applied Combinatorial Optimization*, Montevideo, Uruguay, 2014.
- [15] G. Fagúndez, R. Massobrio, and S. Nesmachnow, “Resolución en línea del problema de viajes compartidos en taxis usando algoritmos evolutivos,” in *Conf. Latinoamericana de Informática*, Montevideo, Uruguay, 2014.
- [16] “The Malva Project: A framework for computational intelligence in C++,” [Online 11-2014] <https://github.com/themalvaproject>.
- [17] S. Nesmachnow, “Computación científica de alto desempeño en la Facultad de Ingeniería, Universidad de la República,” *Revista de la Asociación de Ingenieros del Uruguay*, vol. 61, pp. 12–15, 2010.
- [18] “TaxiFareFinder API,” [Online 11-2014] <http://www.taxifarefinder.com>.

TABLA I
RESULTADOS ALCANZADOS CON CADA TÉCNICA

instancias	métrica	p μ -MOEA/D (R)	p μ -MOEA/D (G)	ávido
chicas	costo	min	105.42	131.35
	costo	avg	312.34	215.08
	demora	min	-2877.00	6039.30
	demora	avg	2115.48	12210.34
medianas	costo	min	283.22	296.80
	costo	avg	840.46	612.49
	demora	min	-12406.20	11613.10
	demora	avg	5286.01	23586.89
grandes	costo	min	369.51	328.88
	costo	avg	768.76	586.53
	demora	min	-3562.70	-3581.70
	demora	avg	5945.52	5886.05
Montevideo	costo	min	146.31	149.33
	costo	avg	361.06	300.72
	demora	min	-2028.85	-2028.85
	demora	avg	1307.22	5288.68

TABLA II
MEJORA ALCANZADA SOBRE EL ALGORITMO ÁVIDO

instancias	métrica	generaciones			
		10000	15000	20000	
chicas	max. mejora ávido.c	max	36.0 % (0.6s)	36.0 % (0.9s)	36.0 % (1.2s)
		avg	21.0 % (1.0s)	21.0 % (1.4s)	21.0 % (1.8s)
		std	12.2	12.2	12.2
	max. mejora ávido.d	max	122.4 % (1.2s)	122.4 % (1.8s)	122.4 % (2.4s)
		avg	111.1 % (0.8s)	111.1 % (1.2s)	111.1 % (1.6s)
		std	5.0	5.0	5.0
medianas	max. mejora ávido.c	max	31.6 % (1.3s)	32.5 % (2.0s)	32.5 % (2.6s)
		avg	19.2 % (1.9s)	19.7 % (2.8s)	19.9 % (3.7s)
		std	9.5	9.4	9.3
	max. mejora ávido.d	max	135.9 % (1.8s)	135.9 % (2.7s)	135.9 % (3.6s)
		avg	120.4 % (1.8s)	120.4 % (2.7s)	120.4 % (3.6s)
		std	8.8	8.8	8.8
grandes	max. mejora ávido.c	max	67.3 % (2.5s)	68.1 % (3.7s)	69.0 % (5.0s)
		avg	15.5 % (2.9s)	16.8 % (4.3s)	17.2 % (5.7s)
		std	19.3	19.3	19.5
	max. mejora ávido.d	max	128.5 % (3.1s)	130.4 % (4.6s)	131.3 % (6.1s)
		avg	115.1 % (2.8s)	115.7 % (4.1s)	116.0 % (5.5s)
		std	6.6	6.9	7.1
Montevideo	max. mejora ávido.c	max	33.9 % (0.9s)	33.9 % (1.3s)	33.9 % (1.8s)
		avg	14.3 % (1.1s)	14.4 % (1.6s)	14.4 % (2.2s)
		std	8.9	8.9	8.9
	max. mejora ávido.d	max	123.5 % (0.5s)	123.5 % (0.7s)	123.5 % (0.9s)
		avg	116.3 % (1.0s)	116.3 % (1.5s)	116.3 % (2.0s)
		std	4.7	4.7	4.7

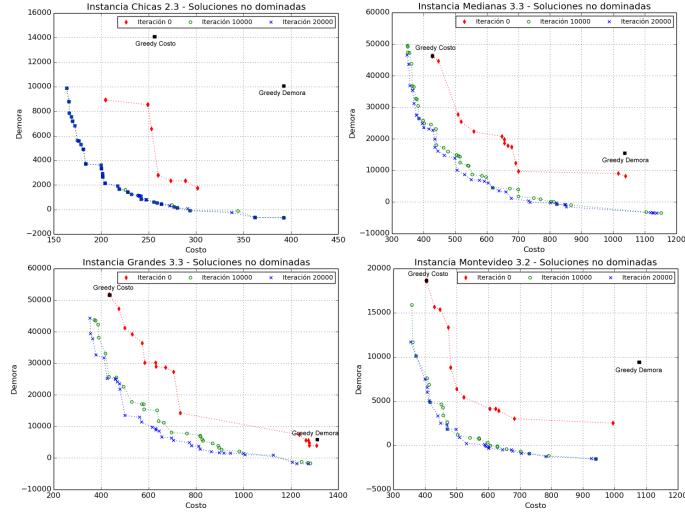


Fig. 5. Frentes de Pareto para una instancia de cada conjunto.

Multiobjective taxi sharing optimization using the NSGA-II evolutionary algorithm

Renzo Massobrio, Sergio Nesmachnow, Gabriel Fagúndez

Universidad de la República, Uruguay
 {renzom,sergion,gabrielf}@fing.edu.uy

Abstract

This article presents the application of the NSGA-II multiobjective evolutionary algorithm to the problem of distributing passengers traveling from the same origin to different destinations in several taxis. A new problem formulation is presented, accounting for two quality of service metrics from the point of view of the users: minimize the total cost of the trips and the time of travel for each passenger from the origin to its destination. The proposed method follows a fully multiobjective approach, and it is designed to provide an accurate and efficient way to solve realistic instances of the problem with high practical applicability. The experimental analysis compares the solutions found using the proposed algorithm versus those computed using a previous parallel micro evolutionary algorithm (following a linear aggregation approach to combine the problem objectives) and two greedy heuristics. The results show that the multiobjective evolutionary algorithm is able to efficiently reach significant improvements in both problem objectives in short execution times.

1 Introduction

Car pooling is the concept of sharing vehicles among people with similar travel needs. The idea has gained massive public attention in recent years [6]. Using shared vehicles has both economical and environmental benefits, at individual and collective levels. Having less vehicles on the streets leads to fewer traffic jams, resulting in a more fluent, thus more efficient traffic. This contributes to minimizing the impact of transportation in the environment, which is a major concern nowadays, specially for big cities [7] [8]. From an economical perspective, sharing trips between multiple passengers significantly reduces transportation costs.

Taxis are a fast and reliable mean of transportation. However, they rarely run at full capacity, and they could therefore benefit from the car pooling idea. There are some web platforms that provide solutions for ride-sharing scheduling, and some research works on different variants of the *taxis pooling* problem. The taxi sharing problem is NP-hard [9]. Thus, heuristics and metaheuristics [16] are needed to find high-quality solutions in reasonable execution times for realistic problem instances.

This article presents a multiobjective evolutionary algorithm (MOEA) to solve the multi-objective one-origin-multiple-destinations variant of the taxi sharing problem. The experimental evaluation over real-world scenarios demonstrates that the proposed method is an accurate and efficient tool to solve the problem, which can be easily integrated in on-line (web, mobile) applications.

The article is organized as follows. Section 2 introduces the multiobjective version of the taxi sharing problem and reviews related work on the topic. Section 3 introduces evolutionary algorithms (EAs) and their multiobjective variants, and describes the proposed MOEA to solve the problem. Section 4 reports the experimental evaluation, including a comparison against a previous EA applying domain decomposition and linear aggregation of objectives from [14] and two greedy heuristics to solve the problem. Finally, Section 5 formulates the conclusions and the main lines for future work.

2 The multiobjective taxi sharing problem

This section introduces the taxi sharing problem and its mathematical formulation as a multiobjective optimization problem, and reviews related works on the topic.

Agadir, June 7-10, 2015

2.1 Problem description and model

The *taxi pooling* problem models a reality where a number of people located in the same origin, decide to travel to different destinations using taxis. The optimization problem proposes finding an appropriate number of taxis and the distribution of passengers, with the goal of simultaneously minimizing two important objectives for users: the cost of the travel and the perceived delay to reach their destinations. This model considers traffic flow scenarios without major traffic jams. Including traffic data in real time is proposed as one of the main lines of current and future work.

Regarding the problem, the following considerations should be taken into account:

- Each taxi can transport a limited number of passengers depending on its capacity. The amount of taxis available for each capacity is given as input to the algorithm.
- The maximum number of taxis that can be used in a scenario with N passengers is N , in the case where every passenger travels in a separate vehicle.
- The cost for each taxi includes the cost to hire the taxi (minimum fare), and the cost for traveling from the origin to the final destination. No other costs are considered such as waiting-times, tips or extra baggage fees.
- Each passenger has an associated tolerance indicating the additional time they are willing to spend due to taxi-sharing, compared to the time it would require a direct journey from the origin to their destinations. The tolerance of each passenger is also given as an input to the algorithm.

2.2 Mathematical formulation

The mathematical formulation is presented below.

Given the following elements:

- A set of passengers $P = \{p_1, p_2, \dots, p_N\}$ starting from a common geographic point O and having a set of potentially different destinations $D = \{d_1, d_2, \dots, d_N\}$.
- A set of taxis $T = \{t_1, t_2, \dots, t_M\}$, $M \leq N$, with capacities (passengers that each taxi can carry) $K = \{K_1, K_2, \dots, K_M\}$.
- A function C that determines how many passengers use a taxi on a given trip, where $C_i \leq K_i$.
- A matrix CM of dimension $(N+1) \times (N+1)$ specifying the cost between each of the geographic locations involved in the problem (an origin point and destination points N).
- A matrix TM of dimension $(N+1) \times (N+1)$ that specifies the traveling time between each of the geographic locations.
- A vector T of length N where T_i is the extra time that user i tolerates over the time it takes to go directly from the origin O to destination d_i .
- A *total cost function*, determined by the minimum fare (MF) and the distance cost from the origin to the final destination:

$$TC = \sum_{t_i} (MF + \sum_{j=1}^{C(t_i)} CM[d_{j-1}, d_j]) \quad (1)$$

- A *total delay function*, determined by the difference between the time tolerated and the actual time that each passenger takes to reach its destination:

$$TD = \sum_{t_i} \left(\sum_{j=1}^{C(t_i)} \left(T_i + TM[0, d_j] - \left(\sum_{h=1}^j (TM[h-1, h]) \right) \right) \right) \quad (2)$$

The problem aims to find a planning function $f: P \rightarrow T$ to transport the N passengers in K taxis ($K \leq M$), determining both the assignment of passengers to taxis and the order to visit the destinations, simultaneously minimizing the total cost (Eq. 1) and the total delay (Eq. 2).

Agadir, June 7-10, 2015

2.3 Related work

The *taxi pooling* problem is a variant of the *car pooling* problem, which has been studied from different perspectives in the related literature.

From the point of view of taxi companies, Xin *et al.* [18] optimized the cost of k taxis serving clients on-line, applying a heuristic that sends the two nearest taxis to attend a request, promoting competition and avoiding underutilization. The results showed that the problem holds a competitive ratio of k against an optimal algorithm that knows the entire sequence of requests beforehand.

Balancing the interest of taxi owners and users, Ma *et al.* [12] proposed a dynamic system for taxi sharing, by combining a search and a planning method to find taxis and assign passengers. A lazy strategy is applied to improve execution times, delaying the shortest path calculation as much as possible. Using a database of 33.000 GPS taxi trajectories from Beijing, the dynamic system reduced the travelled distances up to 13%, while serving 25% more requests in a simulated scenario with 6 requests per taxi.

Closer to the problem being tackled which focuses on customers' interests, Tao *et al.* [17] proposed two heuristic algorithms to optimize taxi ride-sharing costs, based on greedy strategies for the one-origin-to-many-destinations and for the many-to-one scenarios. The results of a field test of taxi-pooling at Taipei with 10 taxis and 798 passengers showed an average matching success rate of 60.3%. However, the fuel savings results are shown only in absolute terms, making it difficult to extract meaningful information to compare with other techniques. We address the one-to-many problem variant, so it is of particular interest to see the performance of the greedy method against our proposed MOEA.

From the point of view of multiobjective optimization, Lin et al. [10] proposed a Simulated Annealing approach to minimize the operational cost for a taxi fleet and maximize the user satisfaction. The problem formulation takes into account the distance traveled by each vehicle and the waiting time of clients. The proposed algorithm was able to reduce the distances in 19% and increasing by 66% the number of taxis available, when compared with a traditional scenario that does not apply taxi sharing.

The analysis of related works indicates that the taxi sharing problem is currently interesting for the scientific community, having a significant impact on both environment protection and economy. However, there are few user-oriented solutions in literature, so there is still room to contribute in this line of research, by proposing efficient optimization methods for planning and reducing vehicular traffic.

Our previous work [5] proposed a simple EA for taxi sharing optimization that showed a good applicability for realistic medium-size problem instances, but requiring large execution times. After that, a parallel EA was implemented to notably improve the results quality and the computational efficiency of the search [13]. Our first multiobjective approach to the problem applied a parallel micro EA using a linear aggregation approach to combine both objectives [14]. Following this line of work, the MOEA we introduce in this article is the first proposal for an explicit multiobjective algorithm to solve the problem.

3 A multiobjective evolutionary algorithm for taxi sharing

This section presents details of the proposed multiobjective evolutionary algorithm for taxi sharing optimization and the greedy heuristics used as a baseline for the results comparison.

3.1 Evolutionary algorithms and NSGA-II

Evolutionary algorithms are non-deterministic metaheuristic methods that emulate the evolution of species in nature to solve optimization, search, and learning problems [2]. In the past 30 years, evolutionary algorithms have been applied to solve many highly complex optimization problems.

MOEAs [3, 4] have obtained accurate results when used to solve difficult real-life optimization problems in many research areas. Unlike many traditional methods for multiobjective optimization, MOEAs find a set with several solutions in a single execution, since they work with a population of solutions in each generation. MOEAs are designed taking into account two goals at the same time: i) approximating the Pareto front, and ii) maintaining diversity instead of converging to a reduced section of the Pareto

front. A Pareto-based evolutionary search leads to the first goal, while the second is accomplished using specific techniques that are also used in multi-modal function optimization (e.g. sharing, crowding).

In this work, we apply the non-dominated sorting genetic algorithm (NSGA), version II [4], a state-of-the-art MOEA that has been successfully applied in many areas. NSGA-II has an improved evolutionary search function compared with its predecessor, NSGA, based on three features: i) a non-dominated, elitist ordering that diminishes the complexity of the dominance check; ii) a crowding technique for diversity preservation; and iii) a fitness assignment method that considers crowding distance values. The algorithm was developed using ECJ [11], a Java-based evolutionary computation research system.

3.2 Implementation details: encoding and objective functions

Solutions are represented as arrays of integers between 1 and N , representing the passengers, and zeros to separate passengers assigned to different taxis. The order for visiting the destinations is the one specified in the sequence. Figure 1 shows an example of the solution encoding for an instance with $N = 5$.

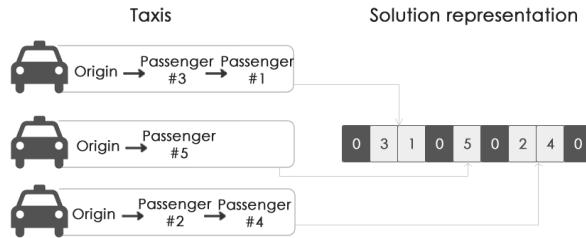


Figure 1: Example of solution representation for the taxi sharing problem.

The solution encoding has some restrictions/features: i) the number of consecutive (non-zero) integers is limited by the available taxi capacities as described in Section 2.2; ii) each integer must appear only once in the encoding; iii) consecutive zeros mean the same than a single one.

The objective functions to optimize according to a fully multiobjective approach are the total cost of the trips and the perceived delay as described in Eq. (1) and Eq. (2).

3.3 Evolutionary Operators

The proposed encoding has specific features and restrictions, so we apply ad-hoc search operators.

Population initialization: the initial population consists of two individuals representing the Greedy solutions for cost and delay (see Section 4.4.1) and the rest of the population is generated by applying a random number of perturbations (*i.e.* swapping two elements) over copies of these two individuals.

Feasibility check and correction process: the initialization method might violate some of the constraints defined for the solution encoding. Thus, a corrective function is applied to guarantee solution feasibility. The method searches for sequences of non-zero digits larger than the maximum vehicle capacity at the moment. Then, the correction algorithm locates the first consecutive couple of zeroes, and moves the first zero to a random place at the non-zero sequence, in order to break the invalid sequence. The search for invalid sequences continues until the end of the solution; at that point, the individual fulfills all problem constraints.

Selection: a tournament selection (tournament size: 2 individuals) is applied to provide an appropriate selection pressure. Initial experiments confirmed that the standard proportional selection technique did not provide enough diversity, leading to premature convergence.

Recombination: we apply an ad-hoc variant of the *Position Based Crossover* (PBX) operator, explained in Algorithm 1. In order to avoid violating the constraints imposed by the solution encoding, the same corrective function used after the initialization is applied to the resulting offsprings. Figure 2 shows an example of the PBX crossover for an instance with 5 passengers.

Algorithm 1 Ad-hoc PBX for the taxi sharing problem

- 1: Randomly select several positions in parent 1.
- 2: Partially generate the offspring, copying the selected values from parent 1.
- 3: Mark in parent 2 the positions already selected in parent 1.
- 4: Select the next non-marked value in parent 2, sequentially from the beginning, and copy it in the first free position in offspring.

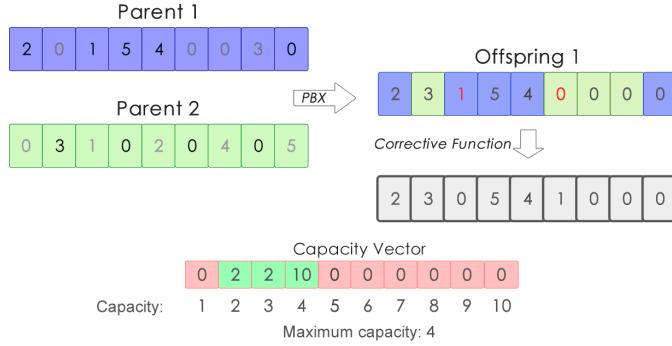


Figure 2: PBX applied to the taxi sharing problem.

Mutation: we apply the *Exchange Mutation* operator, which randomly selects and exchanges two positions in the solution encoding. Afterward, the same corrective function mentioned above is applied to fix potential invalid solutions.

4 Experimental analysis

This section reports the experimental analysis of the proposed multiobjective EA to solve a set of realistic instances of the taxi sharing problem.

4.1 Problem instances

A specific methodological approach for the generation of realistic problem instances was used, taking into account the problem restrictions and using services available to gather information about taxi demands, maps, and fares, including:

- *Taxi Query Generator (TQG)*, a tool that uses information from a database of taxi trajectories obtained from GPS devices installed for a week in 10,357 taxis in the city of Beijing, to generate realistic taxi demands. The data are a subset of those used by Ma et al. [12]. TQG produces a list of origin/destination coordinates for individual trips. In order to design useful instances for the problem addressed in this article, we developed a script that groups trips that originate in nearby locations and generates one origin-to-many destinations problem instances.
- The *TaxiFareFinder* interface [1], which was used to obtain the cost matrix for each generated instance of the problem, along with the prices corresponding to the minimum fare. Each pair of coordinates for a given instance of the problem is sent to the interface TaxiFareFinder to get the cost between each point.

Following the proposed approach, we created a benchmark set of **41** realistic instances of the taxi sharing problem, with different dimensions: *small* (9–24 passengers), *medium* (26–37 passengers), *large*

Agadir, June 7-10, 2015

(42–53 passengers) and a set of instances for the city of Montevideo, Uruguay (9–44 passengers), considering the origins in different points of interest in the city and choosing random destinations. For each instance, we defined different passengers tolerances and vehicles capacities.

4.2 Multiobjective optimization metrics

A large number of metrics have been proposed in the literature to evaluate MOEAs [3, 4]. In this work, we apply several of them to evaluate the results obtained from the NSGA-II algorithm, convergence and correct sampling of the set of non-dominated solutions of the problem:

- The number of (different) non-dominated solutions (ND).
- Generational Distance (GD): the (normalized) sum of the distances between the non-dominated solutions in the Pareto front computed by the algorithm (solutions v in the approximated Pareto front P^*) and a set of uniformly distributed points in the true Pareto front (Eq. 3). Smaller values of GD mean a better approximation to the Pareto front.
- Spacing: evaluates the dispersion of non-dominated solutions in the calculated Pareto front (Eq. 4).
- Spread (s): evaluates the dispersion of non-dominated solutions in the computed Pareto front, including the distance from the extreme points of the true Pareto front (Eq. 5). Smaller values of spread mean a better distribution of non-dominated solutions.
- Hypervolume: the volume (in the objective functions space) covered by the computed Pareto front.
- Relative hypervolume (RHV): the ratio between the volumes (in the objective functions space) covered by the computed Pareto front and the true Pareto front. The ideal RHV value is 1.

$$GD = \frac{\sum_{v \in P^*} d(v, P)}{|P^*|} \quad (3)$$

$$\sqrt{\frac{\sum_{i=1}^{ND} (\bar{d} - d_i)^2}{ND - 1}} \quad (4)$$

$$\frac{\sum_{h=1}^k d_h^e + \sum_{i=1}^{ND} |\bar{d} - d_i|}{\sum_{h=1}^k d_h^e + ND \times \bar{d}} \quad (5)$$

In Eq. 4 and 5, d_i is the distance between the i -th solution in the computed Pareto front and its nearest neighbor, \bar{d} is the average of all d_i and d_h^e is the distance between the extreme of the h -th objective function in the true Pareto front and the closest point in the computed Pareto front.

The true Pareto front—which is unknown for the problem instances studied—is approximated by the set of non-dominated solutions found for each instance, in each execution of the proposed MOEA.

4.3 Parameter tuning

Given the stochastic nature of EAs a parameter setting analysis is mandatory prior to experimental analysis. For this purpose, a set of 4 medium-sized problem instance (different from those used in the experimental analysis in order to avoid bias) was generated following the method detailed in Section 4.1.

After initial experiments the population size was set to 80 individuals. The parameter tuning focused on the crossover probability (p_c , candidate values 0.6, 0.75 and 0.95) and the mutation probability (p_m , candidate values 0.001, 0.01 and 0.1). For each combination of candidate values, 30 independent executions were performed for each problem instance, with 5000 generations on each run.

The parameter settings results suggest that using $p_c = 0.75$ and $p_m = 0.1$ allows computing the best results for the problem instances used.

4.4 Experimental evaluation

The experimental analysis focused on both the quality of the solutions and the performance of the proposed NSGA-II algorithm. For all 41 problem instances 30 independent executions of 5000 generations each were done. The experimental analysis was performed on an Intel Core i7-4500U CPU @ 1.80GHz, 8GB of RAM with 64 bit Ubuntu 14.10.

Agadir, June 7-10, 2015

4.4.1 Comparison against two greedy algorithms

A greedy algorithm is a method that constructs solutions by making locally optimal decisions in each step. In order to compare the results achieved by the NSGA-II, two greedy algorithms were developed for each of the objective functions: cost and delay. Similar strategies can be expected from a group of human users trying to solve the problem.

The greedy method that minimizes cost sequentially adds the passenger whose destination is closer to the origin in a taxi, until that taxi is full. In the case where the cost of adding one passenger to the current taxi is greater than the one obtained by assigning a new taxi to serve that passenger request, the current taxi is “closed”, and a new one is formed. The algorithm ends when every passenger is assigned to a taxi.

The greedy strategy for optimizing the delay takes the most hurried passengers (those with smaller tolerance for delay) and assigns each of them to a new taxi. It then takes the rest of the passengers in order of hurriedness and assigns them to the last location of the taxi that minimizes their delay. When a taxi has as many passengers as the maximum capacity at the moment, it is considered as “closed”. If one taxi has more passengers assigned than the maximum capacity available it is deleted, and the passengers of that taxi are assigned to two new taxis.

4.4.2 Results and discussion

The experimental results of the NSGA-II were compared to those achieved by the $p\mu$ -MOEA/D previously developed by our research group [14]. Table 1 shows the improvement against the greedy strategies, the execution time in seconds and the hypervolume for both algorithms.

Results show that the NSGA-II is able to improve the greedy algorithm for cost in up to 69.9% in the best case (21.2% on average) for large instances and the greedy algorithm for delay in up to 135.9% (120.2% on average) for medium instances.

Furthermore, the NSGA-II is able to get better improvements than the $p\mu$ -MOEA/D in shorter execution times. It is important to highlight that the $p\mu$ -MOEA/D was executed at the Cluster Fing high performance computing facility [15] using 24 cores. Moreover, the NSGA-II achieved better hypervolumes values (both in best case and average) indicating a good approximation to the ideal pareto front while keeping good diversity.

Table 2 shows the multiobjective metrics values for the NSGA-II as described in Section 4.2. The number of non-dominated points on the final generation is equal to the population size for all instances. The values of generational distance are small, indicating good convergence to the pareto front. The small spread values obtained are evidence of good diversity in the solutions, and the relative hypervolumes values close to 1 indicate good coverage and convergence to the pareto front.

Figure 3 shows the results achieved by the NSGA-II, the $p\mu$ -MOEA/D and the greedy algorithms for one representative instance of each class. The NSGA-II outperforms the $p\mu$ -MOEA/D and greedy algorithms, and is able to reach a greater number of non-dominated solutions closer to the pareto front.

5 Conclusions and future work

This article studied the multiobjective taxi sharing problem, from the point of view of taxi users. A specific problem formulation is presented, considering the total cost and total delay objectives, and the NSGA-II multiobjective evolutionary algorithm is applied to find accurate solutions for a set of realistic problem instances.

Unlike our previous work [14], the problem is solved following an explicit multiobjective approach, instead of using a linear aggregation function and weights for the problem objectives.

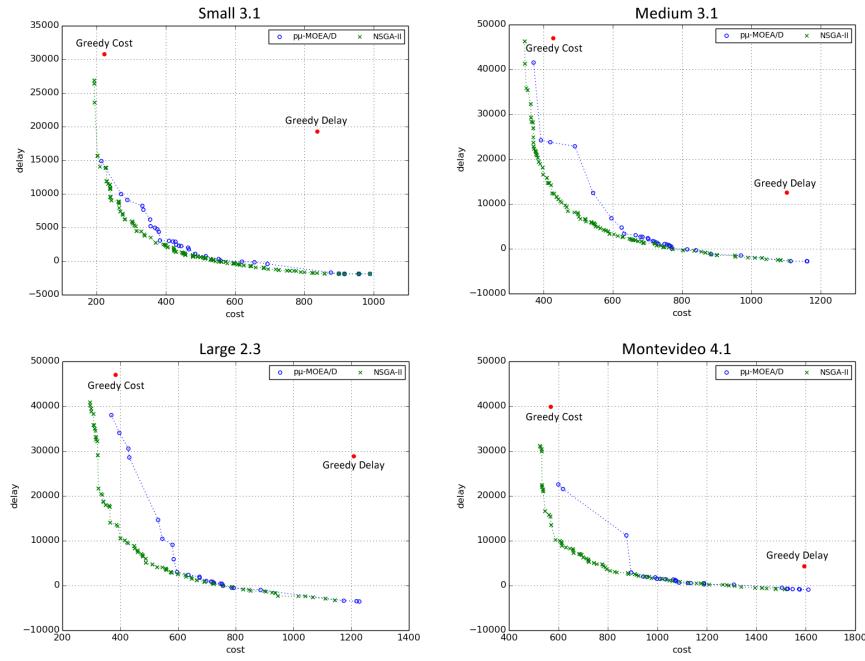
The experimental analysis demonstrates the applicability of the studied MOEA to find a wide set of trade-off solutions for the problem, considering different values for the problem objectives. The numerical results report that the NSGA-II is able to outperform a greedy heuristic for cost in up to 69.9% in

Table 1: Comparison of NSGA-II and $p\mu$ -MOEA/D for each problem instance class

instance class	metric	NSGA-II	$p\mu$ -MOEA/D [14]			
small (9–24 passengers)	improvement over greedy (%)	cost	max. mean σ	36.0 21.6 11.7	36.0 21.0 12.2	
		delay	max. mean σ	122.4 111.1 5.0	122.4 111.1 5.0	
		execution time (s)	min. mean σ	1.5 1.8 0.3	2.1 5.1 2.8	
	hypervolume		max.	2.0 × 10⁷	$1.9 × 10^7$	
			mean	5.6 × 10⁶	$5.3 × 10^6$	
			σ	$6.4 × 10^6$	$6.0 × 10^6$	
	medium (26–37 passengers)	improvement over greedy (%)	cost	max. mean σ	35.4 21.5 10.1	32.5 19.9 9.3
			delay	max. mean σ	135.9 120.2 8.8	135.9 120.4 8.8
			execution time (s)	min. mean σ	2.3 2.8 0.3	4.2 10.6 4.1
		hypervolume	max.	1.2 × 10⁸	$1.1 × 10^8$	
			mean	4.9 × 10⁷	$4.4 × 10^7$	
			σ	$3.8 × 10^7$	$3.4 × 10^7$	
	large (42–53 passengers)	improvement over greedy (%)	cost	max. mean σ	69.9 21.2 18.5	69.0 17.2 19.5
			delay	max. mean σ	123.7 113.3 4.9	131.3 116.0 7.1
			execution time (s)	min. mean σ	3.3 3.7 0.3	7.2 16.6 6.0
		hypervolume	max.	5.0 × 10⁷	$4.3 × 10^7$	
			mean	3.4 × 10⁷	$2.9 × 10^7$	
			σ	$9.3 × 10^6$	$8.5 × 10^6$	
	Montevideo (9–44 passengers)	improvement over greedy (%)	cost	max. mean σ	35.2 14.6 8.5	33.9 14.4 8.9
			delay	max. mean σ	123.5 116.4 4.5	123.5 116.3 4.7
			execution time (s)	min. mean σ	1.5 2.3 0.7	1.8 8.4 5.7
		hypervolume	max.	3.1 × 10⁷	$2.5 × 10^7$	
			mean	9.6 × 10⁶	$8.2 × 10^6$	
			σ	$1.0 × 10^7$	$8.3 × 10^6$	

Table 2: Multiobjective optimization metrics computed for NSGA-II

<i>metric</i>		<i>small</i>	<i>medium</i>	<i>large</i>	<i>Montevideo</i>
non-dominated points	min.	80.0	80.0	80.0	80.0
	mean	80.0	80.0	80.0	80.0
	σ	0.0	0.0	0.0	0.0
generational distance	min.	0.0	0.7	0.7	0.0
	mean	0.2	1.1	1.1	0.5
	σ	0.3	0.3	0.2	0.4
spacing	min.	63.9	72.2	114.8	39.0
	mean	155.7	138.9	144.2	69.5
	σ	107.3	42.9	30.6	20.5
spread	min.	0.3	0.2	0.3	0.3
	mean	0.8	0.3	0.4	0.5
	σ	0.2	0.1	0.1	0.2
RHV	max.	1.0	0.99	0.99	1.00
	mean	1.00	0.98	0.98	0.99
	σ	0.005	0.005	0.006	0.012

Figure 3: NSGA-II, $p\mu$ -MOEA/D [14], and Greedy results for one problem instance of each class.

the best case (21.2% on average) and a greedy algorithm for delay in up to 135.9% (120.2% on average). The NSGA-II reached better results in shorter execution times than our previous $p\mu$ -MOEA/D [14].

The main lines of future work include adding real-time traffic information and taxi availability to the system, in order to consider delays and alternative routes in the solutions.

Problem instances and detailed results are available at www.fing.edu.uy/inco/grupos/cecal/hpc/AG-Taxi/.

References

- [1] TaxiFareFinder API. [Online 03-2015] <http://www.taxifarefinder.com>.
- [2] T. Bäck, D. Fogel, and Z. Michalewicz, editors. *Handbook of evolutionary computation*. Oxford Univ. Press, 1997.
- [3] C. Coello, D. Van Veldhuizen, and G. Lamont. *Evolutionary algorithms for solving multi-objective problems*. Kluwer, New York, 2002.
- [4] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. J. Wiley & Sons, Chichester, 2001.
- [5] G. Fagúndez, R. Massobrio, and S. Nesmachnow. Resolución en línea del problema de viajes compartidos en taxis usando algoritmos evolutivos. In *Proceedings of the 2014 Latin American Computing Conference*, 2014.
- [6] N. Fellows and D. Pitfield. An economic and operational evaluation of urban car-sharing. *Transportation Research Part D: Transport and Environment*, 5(1):1–10, 2000.
- [7] E. Ferrari, R. Manzini, A. Pareschi, A. Persona, and A. Regattieri. The car pooling problem: Heuristic algorithms based on savings functions. *Journal of Advanced Transportation*, 37:243–272, 2003.
- [8] R. Katzev. Car sharing: A new approach to urban transportation problems. *Analyses of Social Issues and Public Policy*, 3(1):65–86, 2003.
- [9] A. Letchford, R. Egglese, and J. Lysgaard. Multistars, partial multistars and the capacitated vehicle routing problem. *Mathematical Programming*, 94(1):21–40, 2002.
- [10] Y. Lin, W. Li, F. Qiu, and H. Xu. Research on optimization of vehicle routing problem for ride-sharing taxi. In *8th Int. Conf. on Traffic and Transportation Studies Changsha, China*, 2012.
- [11] S. Luke. A java-based evolutionary computation research system. <http://cs.gmu.edu/~eclab/projects/ecj/>. Accessed February 2015.
- [12] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *IEEE 29th Int. Conf. on Data Engineering*, pages 410–421, 2013.
- [13] R. Massobrio, G. Fagúndez, and S. Nesmachnow. A parallel micro evolutionary algorithm for taxi sharing optimization. In *VII ALIO/EURO Workshop on Applied Combinatorial Optimization*, Montevideo, Uruguay, 2014.
- [14] R. Massobrio, G. Fagúndez, and S. Nesmachnow. Planificación multiobjetivo de viajes compartidos en taxis utilizando un micro algoritmo evolutivo paralelo. In *X Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, 2015.
- [15] S. Nesmachnow. Computación científica de alto desempeño en la Facultad de Ingeniería, Universidad de la República. *Revista de la Asociación de Ingenieros del Uruguay*, 61:12–15, 2010.
- [16] S. Nesmachnow. Metaheuristics as soft computing techniques for efficient optimization. In M. Khosrow-Pour, editor, *Encyclopedia of Information Science and Technology*, pages 1–10. IGI Global, 2014.
- [17] C-C. Tao and C-Y Chen. Heuristic algorithms for the dynamic taxipooling problem based on intelligent transportation system technologies. In *4th Int. Conf. on Fuzzy Systems and Knowledge Discovery*, pages 590–595, 2007.
- [18] C-L. Xin and W-M. Ma. Scheduling for on-line taxi problem on a real line and competitive algorithms. In *Proc. of the Int. Conf. on Machine Learning and Cybernetics*, pages 3078–3083, 2004.

Agadir, June 7-10, 2015