Artículo Original / Original Research

Ciencias Matemáticas, Vol. 37, No. 1, 2024, Pag. 15-29
Recibido (*Received*): 21-06-2024, Revisado (*Reviewed*): 17-07-2024
Aceptado (*Accepted*): 18-10-2024, En línea (*Online*): 09-12-2024

Grafo para la evaluación automática de soluciones vecinas en problemas de enrutamiento de vehículos

Graph for the automatic evaluation of neighbor solutions in vehicle routing problems

Fernando Raul Rodriguez Flores [1], José Jorge Rodríguez Salgado[2]*

Resumen En este trabajo se presenta el concepto de grafo de evaluación para una solución de un Problema de Enrutamiento de Vehículos. A partir de este grafo es posible obtener el costo de una solución vecina de una manera eficiente y automática. Que el costo se obtenga de manera eficiente significa que para calcularlo se realizan la menor cantidad de operaciones posible. Que sea automático significa que no es necesario programar un código para calcularlo, solo es necesario programar cómo se evalúa una solución cualquiera. Para evaluar la factibilidad de usar esta propuesta, se comparan los tiempos de ejecución de dos variantes de un algoritmo para resolver el problema de Enrutamiento de Vehículos con Restricciones de Capacidad. En la primera variante, el costo de los vecinos se calcula usando la propuesta presentada en este trabajo, mientras que en la segunda variante este costo se calcula usando un método en el que se usan todas las optimizaciones posibles para ese problema específico. Los tiempos de ejecución de la propuesta realizada están entre 1,72 y 4,17 veces el tiempo del algoritmo optimizado para ese problema específico, y tiene la ventaja de que no es necesario programar un método que calcule el costo de los vecinos, ya que este se obtiene a partir del grafo de evaluación.

Palabras Clave: grafo de evaluación, evaluación automática de soluciones vecinas, problema de enrutamiento de vehículos.

Abstract *In this work we present the concept of evaluation graph for a Vehicle Routing Problem solution. Using this graph it is possible to compute the cost of any neighbor solution in an efficient and automatic way. By efficient we mean that the neighbor's cost is computed using the smallest number of operations and it is automatic because it is not necessary to implement the code to compute this cost: the only requirement is the source code to evaluate a solution. To test the feasibility of this proposal we compare the runtime of two algorithms to solve the Capacitated Vehicle Routing Problem. In the first algorithm the cost of the neighbors is computed using the evaluation graph and, in the second one the cost is computed using an algorithm optimized for this specific problem. When the evaluation graph is used, the runtime is 1.72 to 4.17 times slower that the optimized version. However, it has the advantage that the user does not need to write the code to evaluate the neighbor's cost because it can be computed from the graph.*

Keywords: *evaluation graph, automatic evaluation of neighbor solutions, vehicle routing problem.*

Mathematics Subject Classification: 90B06, 90C59, 68T20, 90C27.

[1]Departamento Matemática Aplicada, Facultad de Matemática y Computación, Universidad de La Habana, La Habana, Cuba. Email: fernan@matcom.uh.cu.
[2]Scalian Consulting Spain SL. España. Email: jose.rodriguez@scalian.com.
*Autor para Correspondencia (*Corresponding Author*)

Editado por (*Edited by*): Damian Valdés Santiago, Facultad de Matemática y Computación, Universidad de La Habana, La Habana, Cuba.

Citar como: Rodriguez Flores, F.R.,& Rodriguez Salgado, J.J. (2024). Grafo para la evaluación automática de soluciones vecinas en problemas de enrutamiento de vehículos. *Ciencias Matemáticas*, 37(1), 15–29. DOI: https://doi.org/10.5281/zenodo.14333111. Recuperado a partir de https://revistas.uh.cu/rcm/article/view/9737.

Introducción

Bajo el nombre Problema de Enrutamiento de Vehículos (VRP) se engloba una familia de problemas que tienen en común el objetivo de satisfacer las demandas de clientes geográficamente dispersos al mismo tiempo que se optimiza el uso de algún recurso (tiempo, combustible, etc.), y se cumplen determinadas restricciones que dependen del problema especí-

fico que se quiera resolver. Estos problemas se presentan con frecuencia en la práctica [23] y son objeto de investigación actual [2, 7, 18, 25, 17, 4, 5, 20, 16, 1].

Los VRP son NP-Duros [23], por lo que los métodos exactos para solucionarlos son muy costosos y solo permiten resolver instancias pequeñas [13]. Para solucionar los problemas que se presentan en la práctica se suelen utilizar métodos aproximados, como heurísticas [23] o metaheurísticas [6]. De estos, los métodos de búsqueda local resultan atractivos por la calidad de los resultados que ofrecen en poco tiempo de ejecución [15, 21].

Para solucionar un VRP mediante un algoritmo de búsqueda local se parte de una solución inicial S_0 a la que se le realizan ciertas transformaciones (como cambiar clientes de posición o cambiar los vehículos que visitan a los clientes de una ruta) y a partir de estos cambios se obtiene un conjunto de nuevas soluciones que se denomina vecindad de S_0 . De esta vecindad, se selecciona una solución S_1 de acuerdo a algún criterio, que depende del algoritmo en cuestión [6], y a partir de esta última se repite el proceso hasta que se cumpla algún criterio de parada.

Las transformaciones que se le realizan a la solución inicial para obtener las vecinas reciben del nombre de operaciones elementales y una solución S_1 que sea vecina de S_0 se puede describir unívocamente a partir de ellas.

El objetivo de este trabajo es diseñar e implementar una estrategia que permita calcular, eficientemente y de manera automática, el costo de cualquier solución vecina de una solución dada en un VRP. Esta propuesta es eficiente por dos motivos. El primero está relacionado con el hecho de que para obtener el costo de la solución vecina se realiza la menor cantidad de operaciones posibles. El segundo motivo está relacionado con el esfuerzo humano necesario para obtener este costo, ya que para calcularlo solo es necesario programar cómo se evalúa la solución inicial, y no es necesario programar cómo se calcula el costo de los vecinos.

Para lograr esto se propone representar todas las operaciones necesarias para calcular el costo de la solución mediante un grafo de evaluación. Para obtener el costo de una solución vecina se modifica este grafo a partir de las operaciones elementales que deben realizarse para obtenerla. Una vez realizadas estas modificaciones se obtiene el grafo que representa la evaluación de la solución vecina y el costo de la misma.

El grafo de evaluación se construye cuando se evalúa la solución inicial. En el cálculo del costo de la solución vecina se realiza la menor cantidad de operaciones posible, que dependen únicamente de las operaciones elementales necesarias para obtenerla, y por otra parte, lo único que el usuario debe programar es cómo se evalúa la solución inicial, ya que el costo del vecino se realiza automáticamente a partir de las modificaciones al grafo.

Disponer de una estrategia como la propuesta permitiría ampliar el rango de problemas de enrutamiento de vehículos que se pudieran resolver usando estrategias ya existentes de búsqueda de vecindad local. Estas estrategias usualmente se implementan para una variante específica de VRP [16, 15, 10, 25] y no siempre resulta sencillo o evidente cómo adaptarlas a otras variantes del problema, entre otros factores, porque el cálculo del costo de las soluciones vecinas puede ser muy diferente. Con la propuesta que se presenta en este trabajo, si lo único que es diferente de un problema a otro es la evaluación del costo de las soluciones vecinas, se podrían resolver las nuevas variantes del VRP de manera inmediata a partir de los algoritmos ya existentes.

Hasta donde los autores conocen, en la literatura no se ha reportado una estrategia similar a la propuesta en este trabajo para calcular el costo de cualquier solución vecina de un problema de optimización combinatoria a partir, únicamente, de la evaluación de la solución inicial. Esta idea está inspirada en las técnicas de la diferenciación automática [9].

La estructura de este documento es la siguiente. En la sección 1 se presentan los elementos básicos del trabajo: el problema de enrutamiento de vehículos, los algoritmos de búsqueda local, cómo se puede representar una solución para este problema y algunos elementos sobre el cálculo del costo de una solución dada. En la sección 2 se presenta el grafo de evaluación que representa todas las operaciones realizadas durante el cálculo del costo de la solución y su relación con los distintos elementos de la misma. En la sección 3 se describe cómo se construye el grafo a partir de la evaluación de una solución. La sección 4 describe cómo obtener el costo de una solución vecina a partir de las operaciones elementales que la definen, y en la sección 5 se muestran los resultados de comparar los tiempos de ejecución de un algoritmo en el que se usa el grafo de evaluación y el mismo algoritmo optimizado para calcular el costo de los vecinos de un problema VRP específico. Finalmente, se presentan las limitaciones de la propuesta realizada y los escenarios en los que puede resultar útil, así como las conclusiones, recomendaciones y trabajo futuro.

Relevancia del estudio

Programar cómo calcular el costo de una solución vecina en un problema de enrutamiento de vehículos específico, es una de las tareas que más tiempo consume en la solución de un VRP. Por otro lado, programar cómo evaluar una única solución es una tarea sencilla que se puede hacer en muy poco tiempo. La propuesta realizada en este trabajo permite calcular el costo de cualquier vecino de múltiples VRP, programando únicamente cómo evaluar una solución. Esto permitiría reducir el tiempo requerido para resolver problemas reales de enrutamiento de vehículos.

1. Preliminares

En esta sección se presentan los elementos del VRP que permiten calcular automáticamente el costo de soluciones vecinas de una dada: las componentes del VRP, la forma de las soluciones, los algoritmos de búsqueda local y cómo evaluar la solución inicial.

1.1 Componentes del VRP

El objetivo de los Problemas de Enrutamiento de Vehículos (VRP) es satisfacer las demandas de clientes dispersos geográficamente. Para ello se dispone de una flota de vehículos, se desea optimizar algún recurso como el tiempo, o el costo total de la operación, y se deben satisfacer las restricciones de cada situación específica . En la literatura se pueden encontrar múltiples variantes de este problema, cómo modelarlas matemáticamente y como resolverlas [23].

Las componentes fundamentales de un VRP son los clientes, los vehículos, los depósitos, las rutas que debe recorrer cada vehículo, la función objetivo y las restricciones específicas de cada variante. Cada una de estas componentes tiene asociado un conjunto de propiedades que dependen de cada problema.

Algunos ejemplos de estas propiedades son la demanda de cada cliente y la capacidad máxima de carga de los vehículos. Entre las posibles funciones objetivo para los problemas se pueden considerar minimizar el costo total del transporte, la distancia recorrida por los vehículos, el número de vehículos, el tiempo de distribución, entre otros [8, 23]. Algunas de las restricciones usuales en este tipo de problema son que cada cliente se visite una sola vez, o que determinados vehículos solo pueden atender a clientes específicos [23].

El VRP más sencillo es el Problema de Enrutamiento de Vehículos con Restricciones de Capacidad (CVRP, por sus siglas en inglés) y fue estudiado por primera vez en 1959 [3]. En el CVRP se tiene un único depósito central, una flota infinita de vehículos idénticos y cada cliente se debe visitar exactamente una vez. En este caso, las propiedades de las componentes son: los clientes tienen una demanda que debe ser satisfecha y los vehículos tienen una capacidad de carga que no debe excederse en ningún momento de la ruta. No existe otro tipo de restricción y se desea minimizar la distancia total recorrida por todos los vehículos. En [23] se pueden encontrar varios modelos de programación matemática para este problema.

Los Problemas de Enrutamiento de Vehículos son NP-duros [23], por lo que los métodos exactos resultan muy costosos. En la práctica se suelen emplear heurísticas y metaheurísticas para resolverlos [13, 21, 15, 2]. En este tipo de algoritmos el concepto de solución juega un papel importante.

1.2 Solución de un VRP

Una solución de un VRP especifica cómo visitar los clientes usando la flota disponible. Por lo general, se suelen representar mediante un conjunto de rutas, donde en cada ruta se especifican los clientes que deben ser visitados y en qué orden, así como otros aspectos que puedan ser relevantes para la ruta y que dependa del problema específico como vehículos, o depósitos que se usarán [21, 15].

Por ejemplo, una solución que tenga dos rutas para un CVRP con tres clientes se puede representar mediante $S^* = [[1,2][3]]$. En esta solución, en la primera ruta se parte del depósito central, se visita el cliente 1, luego el cliente 2 y se regresa al depósito; mientras que en la segunda ruta se parte del depósito, se visita al cliente 3 y se regresa al depósito. En este caso no es necesario especificar qué vehículo recorre cada ruta ni de qué depósito parten los vehículos, porque en el CVRP se asume que todos los vehículos son iguales y que existe un único depósito. En un problema con flota heterogénea [22] sí sería necesario definir en cada ruta cuál es el vehículo que realiza el recorrido.

A partir de esta representación de las soluciones se puede definir el concepto de coordenadas o posiciones dentro de una solución. Una coordenada en una solución del VRP es un par ordenado (r, p) donde el primer número representa una ruta, y el segundo, una posición dentro de esa ruta. Tanto las rutas como las posiciones dentro de las rutas se numeran a partir de 1. En el caso de la solución S^*, en la coordenada $(1,1)$ está el cliente 1, y en la coordenada $(2,1)$, el cliente 3. Por convenio, se asume que en la posición $(r,0)$ está el depósito de la ruta r, y si en una ruta hay l clientes, entonces en la posición $(r,l+1)$ se encuentra el depósito de la ruta al que retornan los vehículos al terminar el recorrido.

A partir de la idea de coordenadas es posible hacer referencia al elemento que está en la posición anterior o siguiente a la de un cliente dado. El "elemento" anterior al de la posición (r,p) es el elemento que esté en la posición $(r,p-1)$, y el siguiente es el de la posición $(r,p+1)$. Por definición, el depósito que está al comienzo de la ruta no tiene elemento anterior, y al que regresan los vehículos al final de la ruta no tiene elemento siguiente.

Este tipo de representación se suele usar en algoritmos de Búsqueda Local.

1.3 Algoritmos de Búsqueda Local

Dentro de los métodos aproximados para resolver problemas de enrutamiento de vehículos, los Algoritmos de Búsqueda Local resultan atractivos por la calidad de las soluciones que obtienen en poco tiempo y porque suelen ser sencillos de implementar [21, 21, 19, 15].

Para solucionar un VRP mediante un algoritmo de búsqueda local se parte de una solución inicial S_0 a la que se le realizan ciertas transformaciones (como cambiar clientes de rutas o cambiar los vehículos de las rutas) y a partir de estos cambios se obtiene un conjunto de nuevas soluciones que se denomina vecindad de S_0 y se representa por N_{S_0}. De esta vecindad, se selecciona una nueva solución S_1 y a partir de ella se repite el proceso hasta que se cumpla algún criterio de parada. La selección de S_1 se realiza de acuerdo a algún criterio, que depende del algoritmo en cuestión [6].

A las reglas que se aplican para modificar una solución y obtener las soluciones vecinas se les conoce con el nombre de *estructura de entorno* o *criterio de vecindad* y al proceso que se realiza para seleccionar la solución S_1 entre las demás que componen la vecindad N_{S_0} se conoce como exploración de la vecindad.

La mayoría de los criterios de vecindad reportados en la literatura para resolver variantes del VRP realizan modifica-

ciones a la posición de los clientes en las rutas como *cambiar un cliente de posición* e *intercambiar dos secuencias de clientes manteniendo el orden entre los clientes de cada secuencia* [15, 21], aunque también se pueden definir operaciones que actúen sobre otras componentes de la solución como *cambiar el vehículo de una ruta*.

Los criterios de vecindad se pueden describir en términos de operaciones elementales como *seleccionar un cliente de una posición dada*, *insertar un cliente en una posición dada*, *quitar el vehículo de una ruta* y *poner un vehículo en una ruta*. Por ejemplo, el criterio *cambiar un cliente de posición* está compuesto por las operaciones *extraer un cliente de una posición dada* e *insertar el cliente seleccionado en otra posición*.

Para poder aplicarle estas operaciones a una solución específica es necesario definir cuáles componentes de la solución se deben modificar. Una forma de lograr esto es mediante el uso de coordenadas en la solución. Por ejemplo, eliminar el primer cliente de la primera ruta se describe mediante la operación *seleccionar el cliente que está en la coordenada (1,1)*. De esta forma, un criterio de vecindad está definido mediante las operaciones que se le deben realizar a la solución, y la vecindad o conjunto de soluciones vecinas, se obtiene al aplicarle a la solución inicial las operaciones del criterio de vecindad con todos los posibles valores de las coordenadas.

A partir de una solución S_0, cualquier vecino S_1 se puede definir unívocamente en términos de las operaciones elementales y las coordenadas en que se le deben aplicar a S_0 para obtener S_1. Este hecho permite obtener el costo de un vecino dado, a partir del costo de la solución S_0 y las operaciones y coordenadas necesarias para obtener S_1.

Para obtener el costo de un vecino de S_0, a partir de su costo y las operaciones necesarias para obtener S_1, basta con modificar el costo de S_0 a partir del costo de realizar cada una de las operaciones que permiten obtener S_1. La variación en el costo que se produce al aplicarle una operación elemental a una solución, depende de qué influencia tiene la componente que modifica esta operación en el costo total.

Por ejemplo, en el CVRP, un cliente tributa de dos formas al costo total. Por una parte, contribuye con la distancia recorrida desde el elemento anterior hasta él, y desde él hasta el elemento siguiente; y por la otra, su demanda influyen en la carga que debe transportar el vehículo, lo que pudiera provocar que éste exceda su capacidad. Al eliminar un cliente de la solución, solo es necesario recalcular la distancia que recorre el vehículo, y modificar la carga que este debe transportar. Al modificar la carga que debe transportar el vehículo, también pudiera ser necesario modificar las penalizaciones que se le realizan a la carga del vehículo en esa ruta.

La influencia que tiene cada componente en el costo de la solución depende de cómo se evalúa la solución original. En este trabajo se propone una forma de obtener el costo de los vecinos a partir, únicamente, de la forma en que se evalúa la solución inicial. En la siguiente sección se presentan los elementos fundamentales de la evaluación de una solución de un VRP.

1.4 Evaluación de una solución

Para diseñar un algoritmo que evalúe una solución de un VRP es necesario realizar varias operaciones como crear variables, modificar el valor de esas variables, acceder a las propiedades de las componentes del problema, o penalizar determinados valores. Estas operaciones se combinan apropiadamente de acuerdo al problema que se quiera resolver.

Por ejemplo, para calcular el costo de una solución para un CVRP se pueden crear dos variables: una para almacenar la distancia total recorrida D y otra para almacenar el total de penalizaciones realizadas P.

Seguidamente se puede iterar por cada una de las rutas calculando la distancia que recorre el vehículo, y determinando para cada ruta la carga que el vehículo debe transportar. Esta carga (o demanda) se puede acumular en una variable diferente para cada ruta (*dRuta*). Después de visitar a todos los clientes y haber calculado la carga total que el vehículo debe transportar en esa ruta, se debe penalizar si esta es mayor que la capacidad del vehículo. El resultado de esta penalización se almacena en una variable P, que al terminar de procesar todas las rutas se le suma a la variable D, que es el valor que se devuelve.

Esta idea se puede apreciar en el pseudocódigo que se presenta en la Figura 1, donde *incDist* es una función que incrementa el valor de una variable con la distancia entre los dos elementos que recibe como argumento, *incDemanda* lo aumenta con el valor de la demanda del cliente y *penaliza* recibe una variable y la aumenta con el resultado de penalizar su segundo argumento con respecto valor del tercer argumento que recibe.

```
evaluar(S):
1.  creaVariable D = 0
2.  creaVariable P = 0
3.  por cada ruta r de S:
4.      creaVariable dRuta = 0
5.      prev = depósito(S)
6.      por cada cliente c de r:
7.          incDist(D, prev, c)
8.          incDemanda(dRuta, c)
9.          clientePrevio = c
10.     incDist(D, prev, depósito(S))
11.   penaliza(P, dRuta, capacidad(r))
12. incVar(D, P)
13. devolver D
```

Figura 1. Pseudocódigo para evaluar la solución de un CVRP [*Pseudocode to evaluate the solution of a CVRP*].

Para evaluar el costo de una solución de otra variante del VRP, basta con utilizar esas operaciones de forma que se calculen los valores deseados para ese problema y se penalicen los incumplimientos de las restricciones.

En este trabajo se propone un método para calcular el costo de una solución vecina de cualquier solución de un VRP, usando la menor cantidad de operaciones posible y sin que el usuario tenga que programar el código para realizar este cálculo. Esto se logra a partir de un grafo de evaluación en el que se reflejan todas las operaciones realizadas durante la evaluación de la solución. En las siguientes secciones se presenta este grafo y cómo construirlo a partir del código que permite evaluar una solución.

2. Grafo de evaluación de una solución en un VRP

Cuando se calcula el costo de una solución de un VRP es posible construir un grafo dirigido que contiene todas las operaciones realizadas durante la evaluación, así como su relación con cada uno de sus componentes. Esto permite calcular, con la menor cantidad de operaciones posibles, el costo de soluciones vecinas de la solución a partir de la cual se obtuvo el grafo. En esta sección se presentan las características de este grafo.

Un grafo dirigido $G = (N, A)$ es un par ordenado donde N es un conjunto cuyo elementos reciben el nombre de nodos, y los elementos de $A \subseteq N \times N$ reciben el nombre de aristas. Si existe una arista entre dos nodos n_1 y n_2, o sea, si $(n_1, n_2) \in A$, se dice que hay una arista que sale de n_1 y que incide en n_2.

En el grafo de evaluación de una solución para un VRP existen tres tipos de nodos: nodos estructurales, nodos operacionales y nodos acumuladores. Los nodos estructurales representan componentes de la solución que intervienen en el costo de la misma, como clientes, depósitos o vehículos. Los nodos operacionales representan las operaciones que se realizan durante la evaluación, como aumentar el valor de una variable con la distancia entre dos elementos, o penalizar una variable de acuerdo a alguna propiedad de una componente del problema. Los nodos acumuladores representan las variables que se usan durante la evaluación de la solución.

El conjunto de nodos estructurales se denota por N_E, el de los nodos operacionales por N_O, y el de los acumuladores por N_A. Por convenio, cada vez que se haga referencia al "nodo estructural C", donde C es una componente de la solución, se refiere al nodo estructural que representa la componente C. Del mismo modo, si en la evaluación de la solución se usó una variable V, el "nodo acumulador V" se refiere al nodo acumulador que representa a la variable V.

En el grafo de evaluación existen 4 tipos de aristas: aristas entre nodos estructurales y nodos operacionales, entre nodos acumuladores y nodos operacionales, entre nodos operacionales y acumuladores, y entre dos nodos operacionales. Matemáticamente esto se puede escribir como: $A \subset (N_E \times N_O) \cup (N_A \times N_O) \cup (N_O \times N_A) \cup (N_O \times N_O)$.

Una arista $(e, o) \in N_E \times N_O$ indica que la componente de la solución representada por el nodo estructural e es argumento de la operación representada por el nodo o. Las aristas de la forma $(a, o) \in N_A \times N_O$ indican que la variable representada por el nodo acumulador a es argumento de la operación que representa el nodo o. Las aristas $(o, a) \in N_O \times N_A$ que salen que un nodo operacional e inciden en un nodo acumulador indican que el resultado de la operación o modifica la variable representada por a. Finalmente, las aristas entre dos nodos operacionales (o_1, o_2) indican el orden en que se realizaron las operaciones durante la evaluación de la solución.

Por ejemplo, en un nodo operacional o_1 que represente la operación *incrementar la variable D con la distancia entre los nodos c_1 y c_2* (y que se representa por *incDist(D,c_1,c_2)* en el pseudocódigo de la Figura 2) incidiría una arista con origen en el nodo estructural c_1, y otra que sale del nodo estructural c_2. Al mismo tiempo, de o_1 sale una arista que incide en el nodo acumulador que representa a la variable D.

De manera similar se puede considerar el nodo o_2, que representa la operación *incrementar el valor de la variable D con el valor de la variable P* (y que se representa por *incVar(D, P)* en el pseudocódigo de la Figura 2). Esta operación incrementa el valor de la variable D (distancia total) con el valor de la variable P (penalizaciones). En este caso, en o_2 incide una arista que sale del nodo acumulador P, y de él sale una arista que incide en el nodo acumulador D.

Además de las aristas mencionadas en los ejemplos, que dependen del tipo de operación que represente cada nodo, todo nodo operacional o_i forma parte de dos aristas. Una de ellas incide en o_i y sale del nodo o_{i-1} que representa la operación que se realizó justo antes de o_i. La otra, sale de o_i e incide en o_{i+1}, que representa la operación que se realizó inmediatamente después de o_i. Los únicos dos nodos operacionales en los que esto no se cumple, son el nodo que representa la primera operación de la evaluación, y en el nodo que representa la última operación que se realiza.

A partir de las aristas que inciden en un nodo operacional n_o, y las que salen de él, es posible calcular el resultado de la operación representada por ese nodo. A este proceso se le llama *evaluar* (o *reevaluar*) el nodo. Otro proceso relacionado con un nodo operacional es el de *desevaluar* la operación, que consiste en deshacer el cálculo representado por la operación. Estas operaciones de *reevaluar* y *desevaluar* modifican los valores almacenados en los nodos acumuladores en los que se almacenan los resultados de las operaciones, y permiten modificar el grafo para obtener el costo de las soluciones vecinas, como se describe en la sección 4.

En la Figura 3 se muestra el grafo de evaluación que se obtiene al evaluar la solución $S^* = [[1,2][3]]$ en un problema simplificado, en el que solo interesa determinar la distancia recorrida por todos los vehículos. Esto se puede obtener con un código similar al presentado en la Figura 2.

En este caso los nodos operacionales están representados por rectángulos y con el indicador d_{ij}, y representan el cálculo de la distancia entre los elementos i y j, donde tanto i como j pueden ser cualquiera de los tres clientes o el depósito. Los nodos estructurales están representados por círculos y tienen las etiquetas dep, $c1$, $c2$ y $c3$. Dep representa al depósito y cada nodo c_i, al cliente c_i. Finalmente, el único nodo acumulador es el denotado por D, dentro de un rombo, en el que se

```
evaluar(S):
    creaVariable D = 0
    por cada ruta r de S:
        prev = depósito(S)
        para cada cliente c de r:
            incDist(D, prev, c)
            clientePrevio = c
        incDist(D, prev, depósito(S))
    devolver D
```

Figura 2. Pseudocódigo para evaluar una solución de VRP simplificado [*Pseudocode to evaluate a solution for a simplified VRP*].

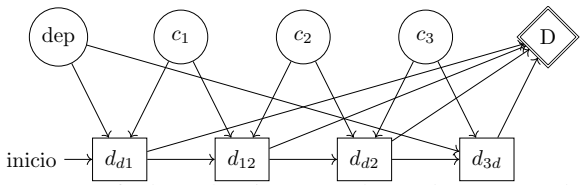

Figura 3. Grafo de evaluación para S^* [*Evaluation graph for* S^*].

acumula la distancia total y representa a la variable *D*. El nodo d_{d1} está señalado como la primera operación que se realiza, y el nodo *D*, encerrado entre dos rombos, como el resultado final que devuelve el algoritmo.

En el grafo aparecen señalados el nodo d_{d1} como la primera operación y el nodo *D* como el resultado final que devuelve el algoritmo. Que exista una arista entre el nodo estructural c_1 y el nodo operacional d_{12} significa que el nodo c_1 contribuye a la operación d_{12}. La arista entre el nodo d_{d1} y el nodo *D* significa que el resultado de la operación d_{d1} se almacena en la variable representada por el nodo *D*. La arista entre el nodo d_{d1} y el nodo d_{12} significa que después de realizar la operación d_{d1}, se realizó la denotada por d_{12}.

Como el algoritmo descrito en 2 devuelve la distancia total, que se calcula en la variable *D*, una vez construido el grafo en el nodo *D* se tiene el costo de la solución.

En la siguiente sección se describe cómo es posible construir el grafo de evaluación para una solución a partir de su evaluación.

3. Construcción del grafo

En esta sección se presentan las ideas fundamentales para construir el grafo de evaluación al mismo tiempo que se calcula el costo de la solución inicial. La propuesta está inspirada en la sobrecarga de operadores que se usa en la diferenciación automática para construir el grafo de evaluación [9] y es la siguiente: para implementar la evaluación de una solución el usuario debe usar funciones específicamente diseñadas para ello. Cada una de esas operaciones realiza dos acciones: realizar los cálculos correspondientes, y además, agregar al grafo los nodos y aristas necesarios para almacenar la operación.

Por ejemplo, en el código mostrado en la Figura 2 la instrucción `incDist(D, prev, c)`, además de incrementar el valor de la variable `D` con la distancia entre el cliente actual (`c`) y el anterior (`prev`), agrega al grafo un nodo que representa esa operación, y también agrega las aristas que permiten conocer cuáles son los argumentos de la operación, dónde se almacena el resultado y cuáles son los elementos de la solución involucradas en ese cálculo.

A continuación se describen los distintos tipos de operaciones que se pueden agregar al grafo y las características de cada una de ellas.

En un algoritmo que permita evaluar la solución de un VRP se pueden identificar dos tipos de operaciones: las que tributan directamente al cálculo del costo de la solución y el resto de las instrucciones. Las que tributan directamente al costo son aquellas que modifican variables a partir de los resultados de las operaciones y aquella que devuelve el valor calculado. El resto de las operaciones son las de control de ciclo y en las que se asignan a variables propiedades de los componentes de la solución. En el grafo de evaluación solo se almacenan aquellas operaciones que tributan al costo.

Las operaciones que tributan al costo de la solución se pueden clasificar en tres grandes tipos: operaciones que crean variables, operaciones que modifican el valor de una variable y la instrucción que indica cuál es el valor que se debe devolver.

Las operaciones que modifican el valor de una variable lo pueden hacer de tres formas: incrementando el valor de la variable con alguna propiedad del problema, incrementándolo a partir del valor de otra variable, o como resultado de penalizar un valor previamente calculado. Cada una de estas operaciones recibe una cantidad diferente de argumentos.

La cantidad de argumentos de las operaciones de tipo *incrementar variable con el valor de una propiedad* puede recibir dos o tres argumentos en dependencia de la propiedad. Por ejemplo, la operación *incrementar variable con la distancia entre dos elementos* recibe tres argumentos: la variable que se debe incrementar y los dos elementos; mientras que la operación *incrementar variable con demanda de un cliente* recibe dos argumentos: la variable que se debe incrementar y el cliente cuya demanda se le debe sumar a la variable.

Otra operación que recibe tres argumentos es *incrementar variable con el resultado de alguna penalización*, que recibe la variable que se debe incrementar, la variable que tiene el valor que se desea penalizar, y la propiedad del problema con respecto a la que se penalizará. La operación *incrementar variable con valor de otra variable* recibe dos argumentos: las dos variables. Finalmente, la operación que indica qué valor se debe devolver, recibe ese único argumento.

Tomando en cuenta la cantidad de argumentos de cada operación que interviene en la evaluación de la solución inicial, es posible construir el grafo, y esto se hace en dos partes. En la primera, se crean todos los nodos estructurales a partir de la solución y, en la segunda se agregan los nodos acumuladores y operacionales a partir de la evaluación de la solución. Para ello, se usan funciones especialmente diseñadas que, ade-

más de calcular el valor de la operación, agreguen al grafo los nodos y las aristas correspondientes. La forma en que se deben agregar estos nodos y aristas se muestra a continuación.

Crear variable: Al evaluar una operación en la que se crea una nueva variable se agrega un nuevo nodo acumulador al grafo.

Incrementar variable con propiedad del problema: En este caso, se crea el nodo operacional correspondiente, se conecta con los nodos estructurales que se usan como argumentos, y se crea una arista entre el nuevo nodo creado y el nodo acumulador que representa a la variable que se debe incrementar.

Incrementar variable con valor de otra variable: Al evaluar una operación de este tipo se crea el nodo operacional correspondiente y se agregan las aristas que lo conectan con los nodos acumuladores involucrados.

Incrementar variable con penalización: Esta operación recibe tres argumentos: la variable que se debe incrementar, la variable que se debe penalizar y la propiedad con respecto a la cual se desea penalizar. En este caso se crea un nodo operacional en el que inciden la variable que se desea penalizar y el nodo estructural que tiene la propiedad con respecto a la que se desea penalizar. Del nodo creado sale una arista que incide en la variable cuyo valor se incrementa.

Devolver el valor: Esta operación marca al nodo acumulador correspondiente como el nodo final del grafo.

Cada vez que se crea un nodo operacional (salvo cuando se crea el primero), además de todas las operaciones indicadas, también se crea una arista que conecta al último nodo operacional con el que se está creando. Esto permite almacenar el orden en el que se realizaron las operaciones.

De esta forma, al terminar la evaluación de la solución se tiene el grafo que representa esa evaluación.

A partir de una operación elemental (por ejemplo, *extraer el primer cliente de la ruta 1*), se puede modificar un grafo existente para obtener el correspondiente a la solución resultante de aplicar la operación, y al mismo tiempo obtener el nuevo costo. Este proceso se describe en la siguiente sección.

4. Evaluación de un vecino en un VRP

En esta sección se muestra cómo se puede obtener el costo de una solución vecina S_1 a partir del grafo de evaluación de S_0 y del conjunto de operaciones elementales que permiten obtener S_1 a partir de S_0. Para ello se indica qué modificaciones se le deben realizar al grafo a partir de las operaciones que se le aplican a la solución.

Existen tres tipos de operaciones que permiten modificar una solución:

- operaciones que eliminan clientes de la solución,
- operaciones que insertan clientes en la solución, y
- operaciones que modifican otros elementos de la ruta, como el vehículo, los depósitos, o alguna otra que pudiera depender del problema específico.

Cada vez que se realiza una operación en la que se elimina un cliente del grafo, es necesario eliminar las operaciones en las que ese cliente está involucrado. Al mismo tiempo que se eliminan esas operaciones se determinan qué nuevas operaciones deben insertarse en el lugar de las operaciones que se eliminan, y qué operaciones deben agregarse al grafo cuando el cliente que está siendo eliminado se inserte nuevamente.

Cuando se inserta un cliente en una posición dada hay que eliminar algunas operaciones del grafo e insertar otras. Estas últimas se calculan en el momento que se elimina el cliente que se desea insertar.

Al modificar otros elementos de la solución como vehículos o depósitos, se determinan cuáles son las operaciones relacionadas con el elemento y cuando se cambia el elemento, se calcula el nuevo valor.

A continuación se describen detalladamente los pasos que deben realizarse cada vez que se aplica una operación elemental de cada tipo a un grafo de operaciones, para obtener el grafo correspondiente a la solución vecina.

En la sección 4.1 se describen en detalle los pasos que se deben realizar para extraer un cliente de una posición dada. En la sección 4.2 se describen los pasos necesarios para insertar un cliente y, finalmente, en la sección 4.3 aparece el procedimiento necesarios al modificar otros elementos de la solución. En la sección 4.4 se muestra un ejemplo en el que se selecciona un cliente de una posición y se inserta en otra.

4.1 Operaciones que eliminan clientes de la solución

Cada vez que se desea eliminar un cliente de una posición, se realizan los siguientes pasos:

1. A partir de la coordenada de la que se desea eliminar el cliente, se determina el nodo estructural asociado con este. Sea n_e dicho nodo.

2. A partir de n_e, se determina el conjunto de nodos operacionales que dependen de él. Sea O_{n_e} el conjunto de estas operaciones.

3. Para cada nodo operacional n_o en O_{n_e} se realizan los siguientes pasos

    3.1. Desevaluar el nodo n_o y todos los que dependan de él, modificando correspondientemente los valores de los nodo acumuladores con los que están conectados.

    3.2. Determinar el conjunto de nodos operacionales I_{n_o} que deben agregarse al grafo cuando el cliente que está siendo eliminado sea insertado nuevamente.

Por ejemplo, si el nodo c_1 que será eliminado está conectado con un nodo operacional que representa la operación *incrementarVariableConDemanda*, cuando se inserte c_1 en otra posición de la solución es necesario agregar una operación *incrementarVariableConDemanda* en la posición adecuada en el grafo. En ese caso, la operación *incrementarVariableConDemanda* estaría en el conjunto I_{n_o}.

3.3. Agregar el conjunto I_{n_o} a la lista de nodos operacionales que deben ser insertados cuando se inserte el cliente representado por el nodo n_e. Sea I_{n_e} ese conjunto. Este conjunto será usado en el momento de insertar el cliente.

3.4. Determinar el conjunto de operaciones J_{n_o} que deben ser insertados en la misma posición de la que n_o está siendo eliminado.

Por ejemplo, si se desea eliminar el cliente que está en la coordenada (1,2), al eliminar el nodo *incrementarVariableConDistancia* que tiene como argumentos el cliente que está en la coordenada (1,1) y el que está en la (1,2), se debe insertar un nodo *incrementarVariableConDistancia* que recibe como argumentos los clientes de las coordenadas (1,1) y (1,3).

En ese mismo caso, al eliminar la operación que incrementa la distancia entre los clientes en las coordenadas (1,2) y (1,3) también es necesario agregar una operación de incrementar distancia entre los nodos (1,1) y (1,3). Sin embargo, esta operación solo se agregaría una sola vez porque I_{n_o} es un conjunto, .

En determinados casos, el conjunto J_{n_o} puede ser vacío. Por ejemplo, cuando el nodo n_o es de tipo *incrementarVariableConDemanda* no hay que agregar ninguna operación a J_{n_o}, ya que al eliminarla no es necesario agregar ninguna otra en esa misma posición.

3.5. Agregar el conjunto J_{n_o} al conjunto de nodos operacionales que deben ser insertados en la misma posición que en la que se elimina n_e. Sea J_{n_e} este conjunto.

3.6. Eliminar el nodo operacional n_o del grafo.

4. Una vez eliminados todos los nodos operacionales que dependan del nodo operacional n_e, se deben insertar todas las operaciones en el conjunto J_{n_e} en la misma posición del grafo en que se encontraban las operaciones relacionadas con n_e.

5. Eliminar el nodo n_e.

6. Reevaluar todas las operaciones de J_{n_e} que fueron insertadas, y todas las demás operaciones que dependen de ellas.

Una vez realizadas estas operaciones, la nueva estructura del grafo refleja la solución vecina y, producto de las reevaluaciones realizadas, en el nodo final del grafo se tiene la evaluación de la nueva solución.

En la sección 4.4 se ilustra la aplicación de estos pasos en un ejemplo concreto.

4.2 Operaciones que insertan clientes en la solución

Cuando se desea insertar un cliente (previamente seleccionado) en una posición dada, se realizan los siguientes pasos, en los que se asume que n_e es el nodo estructural que representa al cliente que será insertado.

1. A partir de la posición en la que se desea insertar el cliente, se determinan el nodo estructural que está en la coordenada anterior y el que estará en la siguiente, denotados por n_{e-1} y n_{e+1}, respectivamente.

2. Determinar el conjunto de nodos operacionales que dependen simultáneamente de n_{e-1} y n_{e+1}. Sea $O_{n_{e\pm1}}$ el conjunto de estas operaciones.

3. Para cada nodo operacional n_o que pertenezca a $O_{n_{e\pm1}}$ se realizan las siguientes operaciones:

   3.1. desevaluar n_o,

   3.2. eliminar n_o del grafo.

4. Insertar el nodo n_e entre los nodos n_{e-1} y n_{e+1}.

5. Insertar en el grafo las operaciones del conjunto I_{n_e}, que fue calculado en el momento de seleccionar el cliente.

6. Reevaluar las operaciones insertadas, y todas las que dependan de ellas.

Una vez realizadas estas operaciones, el nuevo grafo refleja la estructura de la solución vecina, y en el nodo final del grafo se tiene la evaluación de la misma. En la sección 4.4 se ilustra la aplicación de estos pasos en un ejemplo concreto.

4.3 Operaciones que modifican otros elementos de la solución

El tercer tipo de operación representa a aquellas que modifican algún elemento de la solución que no sean clientes, como los vehículos o los depósitos. Ejemplos de estas operaciones son: cambiar el vehículo o el depósito de una ruta.

Cuando se modifica uno de estos elementos, se realizan las siguientes operaciones:

1. Determinar el nodo estructural n_e que corresponde al elemento que se desea modificar. En este caso el nodo n_e representa a alguna componente de la solución como un vehículo o un depósito.

2. Determinar el conjunto de operaciones que dependen de n_e. Sea O_{n_e} este conjunto.

3. Para cada nodo operacional n_o del conjunto O_{n_e}:

   3.1. Desevaluar n_o, y todos los nodos que dependan de él.

   3.2. Marcar el nodo n_o y todos los que dependan de él como pendiente de reevaluación.

4. Cambiar el elemento del nodo n_e.

   Esto significa modificar la componente de la solución que estaba asociada con n_e. Por ejemplo, si n_e representaba un vehículo, en este paso se cambia ese vehículo por otro.

5. Reevaluar cada uno de los nodos operacionales que están en el conjunto O_{n_e}, y todos los que dependan de ellos.

Al realizar estos pasos se obtiene el grafo correspondiente a la solución vecina, y en el nodo final del mismo, el costo de la nueva solución. En la siguiente sección se muestra el resultado de aplicar estos pasos.

4.4 Ejemplo de modificación del grafo

A continuación se ilustra el método propuesto a partir de aplicar las operaciones *eliminar el cliente de la posición (1,1)* e *insertarlo en la posición (2,1)* al grafo computacional mostrado en la Figura 3, que representa la evaluación de la solución $S^* = [[1,2],[3]]$ usando el pseudocódigo 2.

El primer paso consiste en determinar el nodo estructural que corresponde al elemento que se encuentra en la posición $(1,1)$. En este caso, es el nodo c_1. El segundo paso es determinar el conjunto de operaciones que dependen de c_1. En este caso son los nodos d_{d1} y d_{12} y en la Figura 4 se muestran en rojo.

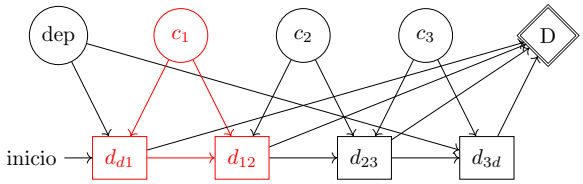

Figura 4. Operaciones que dependen de c_1 [*Operations depending on* c_1].

Cada uno de estos nodos se desevalúa, con lo que disminuye el valor almacenado en D en la distancia desde el depósito hasta el cliente 1, y desde el cliente 1 hasta el cliente 2.

Seguidamente, se agregan dos operaciones al conjunto I_{c_1}: un nodo para calcular la distancia desde el cliente anterior a la posición donde se inserte hasta c_1 y otro desde c_1 hasta el cliente que esté en la posición siguiente. También se agrega al conjunto J_{c_1} una operación de calcular distancia desde el depósito hasta el cliente c_2, porque el depósito es el cliente que está en la posición anterior a c_1 y c_2 es el que está en la posición siguiente.

Los siguientes pasos son eliminar los nodos en O_{c_1}, eliminar el nodo estructural c_1 y, por último, insertar y reevaluar las operaciones en el conjunto J_{c_1}. Una vez terminado el proceso se obtiene el grafo que se muestra en la Figura 5, que es el que se obtiene después de realizar la operación *extraer el cliente de la posición (1,1)*, que representa la evaluación de la solución $S' = [[2],[3]]$, y que en el valor del nodo D tiene el costo de la misma.

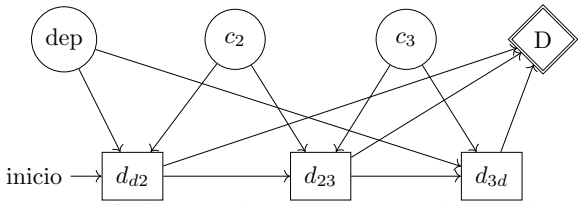

Figura 5. Grafo después de extraer c_1 [*Graph after the extraction of* c_1)].

Para realizar la operación *insertar el cliente seleccionado en la posición (2,1)* se realizan los siguientes pasos.

Determinar los nodos estructurales que corresponden a los clientes en las posiciones $(2,0)$ y $(2,1)$, que son los clientes que estarán antes y después del cliente insertado. En este caso son los nodos dep y c_3 que corresponden al depósito y al cliente 3, respectivamente.

Una vez identificados estos clientes, se determinan los nodos operacionales que dependan simultáneamente de los dos, se desevalúan y se eliminan. En este caso, es solo el nodo d_{d3}, que al desevaluarse disminuye el valor del nodo D en la distancia desde el depósito hasta el cliente 3.

Seguidamente, se inserta el cliente c_1 en la posición $(2,1)$ (entre el depósito y el cliente 3) y se agregan al grafo los nodos en el conjunto I_{c_1}, que en este caso son dos: uno para calcular la distancia desde el depósito hasta c_1 y otro para calcular la distancia desde c_1 hasta c_3, porque el depósito es el cliente anterior a la posición donde se insertará c_1, y c_3 está en la siguiente. Una vez insertados, estos nodos se reevalúan, así como todos los demás nodos operacionales que dependan de los valores que ellos modifican.

El grafo resultante de realizar estas operaciones se puede ver en la Figura 6, donde se resaltan en rojo los elementos insertados.

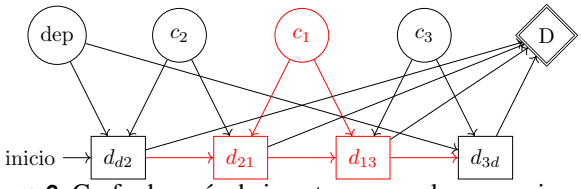

Figura 6. Grafo después de insertar c_1, con las operaciones en J_{c_1} resaltadas en rojo [*Graph after inserting* c_1 *with the operations in* J_{c_1} *highlighted in red*].

Finalmente, se reevalúan los nuevos nodos insertados y todas las demás operaciones que dependan de ellos. De esa forma se tiene el costo de la nueva solución en el nodo D.

Es importante resaltar que el cálculo del costo de la solución vecina depende únicamente de las operaciones realizadas

durante la evaluación de la solución inicial, que para hacerlo se realiza la menor cantidad de operaciones posibles, y que el único código computacional que debe escribir el usuario es cómo se evalúa la solución inicial.

En la siguiente sección se comparan los tiempos de ejecución de esta propuesta cuando se usa para resolver el CVRP, con los tiempos de ejecución de un algoritmo diseñado y optimizado para resolver únicamente ese problema.

5. Análisis de los tiempos de ejecución

En esta sección se compara el tiempo de ejecución de la propuesta realizada en este trabajo al resolver el CVRP con el tiempo de ejecución de un algoritmo diseñado y optimizado específicamente para ese problema.

El objetivo fundamental es medir dos factores: el tiempo de construcción del grafo inicial y cuánto más lento resulta usar esta estrategia en un algoritmo de Búsqueda Local para la solución del CVRP. El algoritmo seleccionado fue Búsqueda de Descenso por Vecindad [11].

Los experimentos se ejecutaron en una laptop con dos núcleos Intel(R) Core(TM) i7-4700MQ a 2.4GHz y 8GB de memoria RAM.

En los epígrafes siguientes se describen los experimentos realizados y los resultados obtenidos.

5.1 Tiempo de construcción del grafo

El objetivo del primer experimento es analizar el tiempo que toma calcular el costo de una solución y construir el grafo de evaluación, y compararlo con el tiempo que tomaría evaluar una solución sin construirlo. Para ello, se seleccionaron 7 problemas de [14], se generaron 5000 soluciones aleatorias para cada uno de ellos y se evaluaron dos veces, en una se construyó el grafo y en la otra no.

Los problemas escogidos fueron A-N33-K5, A-N33-K6, A-N32-K5, A-N37-K5, A-N65-K9, A-N80-K10 y F-N135-K7. En esa notación el número que aparece a la derecha de la N indica la cantidad de clientes en el problema, y el número que aparece a la derecha de la K indica la cantidad de rutas que debe tener la solución óptima.

La tabla 1 muestra los resultados de este experimento. En la primera columna se muestra el problema en el que fueron evaluadas las soluciones. La segunda columna muestra el tiempo que tomó evaluar las 5000 soluciones usando la estrategia propuesta en este trabajo. La tercera columna muestra el tiempo que tomó la evaluación "tradicional" en la que solo se calcula el valor de la solución sin construir el grafo. En la columna 4 aparece cuántas veces más lenta es la construcción del grafo que la evaluación tradicional. Este valor se obtiene dividiendo el tiempo que tomó construir el grafo entre el tiempo que toma evaluar la solución sin construirlo. En la quinta columna aparece el tiempo promedio de evaluar una solución mientras se construye el grafo, y en la sexta columna el tiempo promedio de la evaluación de una solución sin construir el grafo. Todos los tiempos son en segundos.

Tabla 1. Tiempo de construcción del grafo para 5000 soluciones [*Graph construction time for 5000 solutions*].

Problema	Grafo	No Grafo	Factor	Avg	Avg no grafo
A-N33-K5	0,71	0,014	50,71	0.0001	28×10^{-7}
A-N33-K6	0,45	0,013	34,85	0,0001	26×10^{-7}
A-N32-K5	0,43	0,013	32,85	0,0001	26×10^{-7}
A-N37-K5	0,54	0,016	33,81	0,0001	32×10^{-7}
A-N65-K9	1,25	0,027	46,11	0,0002	54×10^{-7}
A-N80-K10	1,77	0,032	55,25	0,0004	64×10^{-7}
F-N135-K7	5,09	0,058	87,72	0,001	$1{,}16\times10^{-5}$

A partir de estos resultados preliminares, parece ser que la construcción del grafo puede ser entre 30 y 90 veces más lenta que la evaluación tradicional. Sin embargo, esto no es un problema por dos motivos. El primero es que cuando se usa un algoritmo de búsqueda local como Búsqueda de Vecindad Variable, Búsqueda Local Iterada o Búsqueda en Vecindades Grandes [6, 21], la evaluación de una solución con la correspondiente construcción del grafo se realiza una única vez en todo el proceso. Y justamente la idea de calcular el costo de las soluciones vecinas usando el grafo de evaluación resulta útil en este tipo de algoritmos. El segundo motivo por el que eso no es un problema es que los tiempos promedio de construcción del grafo de evaluación, que se pueden apreciar en la columna 5 de la tabla 1, están en el orden de las milésimas de segundo.

En la siguiente sección se compara el tiempo de ejecución de un algoritmo de Búsqueda de Descenso por Vecindad para resolver el CVRP, usando dos variantes para calcular el costo de las soluciones vecinas. En la primera se usa el grafo de evaluación y en la segunda se usa una implementación especialmente optimizada para ese problema.

5.2 Tiempo de ejecución de un algoritmo

Para analizar el comportamiento de la propuesta realizada durante la corrida de un algoritmo completo, se implementó una Búsqueda de Descenso por Vecindades [11] con una estrategia de búsqueda exhaustiva y selección del mejor vecino.

La Búsqueda de Descenso por Vecindades recibe un criterio de vecindad, una solución actual y realiza los siguientes pasos: se analiza el costo de todos los vecinos de la solución (por eso es búsqueda exhaustiva) y si el mejor de ellos tiene un costo menor que la solución actual, ese se convierte en la solución actual. El algoritmo se detiene cuando el mejor vecino tiene un costo mayor o igual que la solución actual, y se considera que esa es la mejor solución posible. Como se usa búsqueda exhaustiva y la solución actual siempre se cambia por la mejor de la vecindad, este algoritmo es determinista. En caso de que haya varias soluciones vecinas con el mismo mejor costo, se devuelve la primera que encuentra el algoritmo.

Para estos experimentos su usaron los mismos problemas presentados en la sección anterior y 6 criterios de vecindad, que son:

RAB: seleccionar un cliente y ubicarlo en otra posición dentro de su misma ruta,

RARB: seleccionar un cliente y ubicarlo en otra posición de cualquier ruta,

RARAC: seleccionar dos clientes e intercambiarlos,

REF: seleccionar una subruta y ubicarla en otra posición dentro de su misma ruta,

RERF: seleccionar una subruta y ubicarla en otra posición en cualquier ruta, y

REREG: seleccionar dos subrutas e intercambiarlas.

Para cada par *(problema, criterio de vecindad)* se generaron 60 soluciones aleatorias, con las excepciones de los pares *(A-N80-K10, RERF)*, *(F-N135-K7, RERF)* y *(F-N135-K7, REREG)* para las que solo se generaron 30.

Para cada combinación de *problema, criterio de vecindad y solución inicial*, se corrió el algoritmo dos veces. En la primera, el cálculo del costo de cada solución vecina se realizó siguiendo el algoritmo propuesto en [24], que garantiza que se realicen la menor cantidad de operaciones posibles, tomando en cuenta la implementación realizada.

En la segunda corrida, el costo de las soluciones vecinas se calcula usando el grafo de evaluación. Con esos dos tiempos se calculó el cociente *tiempo de la versión que usa el grafo sobre tiempo de la versión optimizada*. Cuando ese cociente es 1 significa que los dos tiempos fueron iguales. Si el cociente es mayor que 1 significa que la propuesta fue más lenta, y cuando es menor que uno, significa que la propuesta fue más rápida. El objetivo del experimento es tener una idea inicial de cuánto más lento resulta utilizar el grafo de evaluación.

Hubo tres combinaciones de *problema, criterio y solución* en las que el tiempo de corrida para la variante optimizada fue inferior a 10^{-4}, y ese tiempo se reportó como 0.0 segundos. En esos casos no fue posible calcular el cociente, y por tanto fueron excluidos del análisis. Eso ocurrió en problemas con 32, 33 y 37 clientes, y en los tres casos, el tiempo reportado por el algoritmo completo con el cálculo automático del costo de los vecinos fue de 0.001 segundos. Al no poder calcular el cociente, estos tres casos fueron eliminados y el análisis se realizó con 2427 combinaciones de *problema, criterio de vecindad, solución inicial*.

Un análisis descriptivo de ese cociente arrojó la información que se muestra en la Tabla 2. De los 2427 casos analizados, el valor mínimo fue 0,5 y el máximo 12,0, con una media de 2,25. Sin embargo, en este caso se identificaron 18 *outliers*: 15 valores demasiado grandes y 3 demasiado pequeños.

Los valores que estuvieron por encima del máximo esperado se muestran en la Tabla 3 que tiene en la primera columna el problema y en la segunda el criterio de vecindad, además de los tiempos de ejecución sin grafo, con grafo, y el cociente. Ahí se puede apreciar que aunque el cociente de los tiempos de ejecución sea grande, en la mayoría de los casos (con la excepción de 2) los tiempos de ejecución están por debajo del segundo. En el caso del mayor cociente reportado, que fue 12.0, el tiempo que demoró la ejecución del algoritmo usando el grafo fue solo de 0.012 segundos. También es notable que con la excepción de dos casos (el 10 y el 14), las vecindades son "pequeñas" en el sentido de que tienen pocos elementos.

Tabla 2. Cuánto más lento es usar el grafo [*How much slower is it to use the graph*].

Propiedad	valor
Total	2427
Media	2,259434
Desviación estándar	0,718694
Mínimo	0,500000
25 %	1,730769
50 %	2,156309
75 %	2,713047
Máximo	12,000000

Tabla 3. Valor extremos superiores del cociente (*outliers*) [*Upper extreme values of the ratio (outliers)*].

	Problema	Criterio	Sin Grafo	Con Grafo	Cociente
1	A-N33-K5	RAB	0,012	0,051	4,250
2	A-N33-K5	RAB	0,012	0,051	4,250
3	A-N33-K6	RAB	0,002	0,014	7,000
4	A-N33-K6	RAB	0,001	0,005	5,000
5	A-N33-K6	RAB	0,011	0,047	4,273
6	A-N33-K6	RAB	0,024	0,170	7,083
7	A-N33-K6	RAB	0,009	0,053	5,889
8	A-N32-K5	RAB	0,001	0,012	12,00
9	A-N32-K5	RAB	0,001	0,006	6,000
10	A-N32-K5	RERF	0,843	3,550	4,211
11	A-N37-K5	RAB	0,016	0,080	5,000
12	A-N37-K5	RAB	0,019	0,095	5,000
13	A-N37-K5	RARB	0,172	0,752	4,372
14	A-N37-K5	RERF	1,695	7,654	4,516
15	A-N80-K10	RAB	0,093	0,416	4,473

Los tres *outliers* que estaban por debajo del mínimo esperado para los datos se muestran en la Tabla 4 y se puede apreciar que en estos casos los tiempos de ejecución usando el grafo de evaluación para calcular el costo de los vecinos es inferior a 0,1 segundos.

Tabla 4. Valor extremos inferiores del cociente (*outliers*) [*Lower extreme values of the ratio (outliers)*].

Problema	Criterio	Optimizado	Con grafo	Cociente
A-N33-K5	REF	0,002	0,001	0,500
A-N33-K6	RAB	0,061	0,034	0,557
A-N65-K9	RAB	0,011	0,008	0,727

Cuando se eliminan estos *outliers*, se obtiene la información que se muestra en la Tabla 5. En este caso se analizaron 2409 combinaciones con un cociente mínimo de 0,8, un cociente máximo de 4,17, y una media de 2,24. Además, se pudo apreciar, aunque no aparece en la tabla, que en 37 de las corridas realizadas, se obtuvo un cociente inferior o igual a 1, lo que significa que en esas corridas fue más rápido usar el grafo de evaluación para calcular el costo de los vecinos que usar el algoritmo optimizado para ese problema.

Con respecto al análisis de los datos preliminares obtenidos sobre los tiempos de corrida, al analizar los cocientes menores que uno (que son aquellos casos en los que el algorit-

Tabla 5. Resumen del análisis del cociente (sin *outliers*) [*Summary of the ratio analysis (without outliers*].

Propiedad	Valor
Total	2409
Media	2,240990
Desviación estándar	0,651907
Mínimo	0,800000
25 %	1,730748
50 %	2,152284
75 %	2,707593
Máximo	4,176471

mo usando el grafo fue más rápido que la versión optimizada manualmente para ese problema específico), se puede apreciar que en los problemas pequeños (con 32, 33 y 37 clientes) el menor cociente que se obtiene es 1. Esto significa que con esos problemas, el mejor desempeño del algoritmo con grafo es el mismo del algoritmo sin grafo. Sin embargo, en problemas más grandes en los que se usaron criterios de vecindad que requieren explorar más vecinos (problemas con 65, 80 y 135 clientes, y criterios como intercambiar dos subrutas), algunos cocientes son menores que 1. Esto pudiera sugerir que mientras más grande sea la vecindad que se desea explorar más conveniente sea usar esta propuesta. Sin embargo, para confirmar o refutar esa afirmación sería necesaria una mayor experimentación.

De este análisis preliminar pudiera esperarse que usar la propuesta presentada en este trabajo para resolver una variante específica del VRP sea 4 o 5 veces más lento que usar una variante en la que se haya programado (y optimizado) cómo calcular el costo de cada vecino en ese problema específico. Y aunque el tiempo de corrida pueda ser 4 o 5 veces mayor, se tendría la ventaja de que no habría que dedicarle tiempo a programar cómo calcular el costo de cada vecino, que es una actividad que toma mucho más tiempo que 4 o 5 veces el tiempo de ejecución de la versión optimizada. Esto pudiera sugerir que con respecto al tiempo total dedicado a la solución del problema, esta propuesta puede resultar ventajosa.

6. Limitaciones de la propuesta y escenarios donde se puede usar

En este trabajo se ha propuesto un mecanismo que permite calcular, en un problema de enrutamiento de vehículos cualquiera, el costo de una solución vecina de otra cuya evaluación se conoce, de forma que se este costo se obtenga con la menor cantidad de operaciones posibles y sin tener que escribir el código para calcularlo. Esto se logra construyendo un grafo que representa el cálculo del costo de la solución al que, para obtener el costo del vecino, se le realizan la menor cantidad de operaciones posibles para obtener el grafo que corresponde a la solución vecina. En esta sección se presentan las limitaciones de esta propuesta.

La alternativa que se presenta en este trabajo se puede aplicar en cualquier VRP en el que al evaluar una solución todas las penalizaciones relacionadas con una ruta se realicen después de visitar a todos los clientes de esa ruta.

Un ejemplo de problema que cumple con esta condición es el CVRP, en el que una vez que se visitan todos los clientes, se penaliza la carga del vehículo. Otras situaciones en las que se penalice algún valor "después de visitar todos los clientes" pueden ser que la distancia total recorrida por un vehículo deba ser menor que una cantidad prefijada, o que el tiempo que tome visitar a todos los clientes de la ruta sea menor que un valor conocido.

Ejemplos de problemas en las que esta condición no se cumple son el problema con ventanas de tiempo [23] o problemas con entrega y recogida simultánea [12] en el que, al visitar a cada cliente se penalice la carga del vehículo para garantizar que la solución obtenida sea factible. En estos casos cada vez que se visita un cliente se debe verificar el cumplimiento de determinadas propiedades como estar dentro de la ventana de tiempo o no sobrepasar la capacidad del vehículo.

El motivo por el que solo se puede calcular el costo de los vecinos en problemas en los que se penalice al final de la ruta es que cuando se construye el grafo, en los nodos acumuladores que representan las variables, solamente se guarda el último valor que estas tuvieron al finalizar la evaluación. Si no existen penalizaciones durante la evaluación de una ruta, el último valor de la variable depende directamente de las propiedades de los clientes (demanda o tiempo de servicio, por ejemplo) y de esa forma, cuando se modifican las rutas, es posible obtener el nuevo valor de la variable.

Sin embargo, si durante la evaluación de la ruta se realizan penalizaciones, estas reciben como argumento el valor que tenía la variable en el momento que se penalizan. Cuando se modifique la ruta extrayendo o insertando un cliente, para actualizar el costo de la penalización en la solución vecina haría falta tener acceso al valor de la variable en el momento que se realiza la penalización, y como en el grafo de evaluación solo se guarda el último valor de la variable, esto no es posible.

Sin embargo, en cualquier VRP en el que las penalizaciones se realicen después de visitar todos los clientes de la ruta es posible aplicar esta propuesta.

7. Conclusiones, recomendaciones y trabajo futuro

En este trabajo se ha propuesto una vía para evaluar, de manera automática y eficiente, soluciones vecinas de una dada en un problema de enrutamiento de vehículos. Que la evaluación sea eficiente significa que se realizan la menor cantidad de operaciones posibles para obtener el costo, y que sea automática significa que el usuario solo debe programar cómo se evalúa una solución y puede desentenderse del cálculo del costo de los vecinos.

Esto se logra mediante el uso de un grafo de evaluación que refleja todas las operaciones realizadas durante el cálculo del costo de una solución, y en el que existen tres tipos de nodos: operacionales, estructurales y acumuladores. Los nodos operacionales representan las operaciones realizadas durante

la evaluación; los estructurales, las componentes de la solución evaluada; y los acumuladores representan las variables que se usan en los cálculos.

El grafo se construye cuando se evalúa la solución usando funciones especialmente diseñadas para eso, de forma que cuando se ejecutan, además de calcular los valores correspondientes, agregan al grafo los nodos relacionados con esa operación. Las soluciones vecinas se definen mediante las operaciones elementales necesarias para obtenerlas a partir de la solución actual, y para el evaluar el costo de una de ellas, el grafo se modifica apropiadamente de forma que al terminar las modificaciones se obtiene el grafo que representa a la solución vecina y su costo.

En los experimentos preliminares realizados para analizar los tiempos de ejecución al usar esta propuesta, se obtuvo que la evaluación inicial de la solución puede ser entre 30 y 90 veces más lenta que una evaluación en la que no se construya el grafo de evaluación, pero esto no es un problema porque en los algoritmos de búsqueda local este grafo se construye una única vez al comienzo del mismo y además, el tiempo promedio de la construcción del grafo está en el orden de las milésimas de segundo.

Por otra parte, también se evaluó cuánto más lento resulta usar esta propuesta en una Búsqueda de Descenso por Vecindad y se compararon los dos tiempos de corrida: cuando se usa el grafo de evaluación y cuando no se usa. De estos experimentos se obtuvo que al usar el grafo se puede esperar que el tiempo de ejecución sea hasta 5 veces más lento que un algoritmo optimizado para el problema específico, aunque también hubo casos (muy pocos) en que el algoritmo que calcula el costo de los vecinos usando el grafo de evaluación fue más rápido (hasta 0.8 veces) que la versión optimizada. Estos resultados sugieren que puede ser conveniente usar esta propuesta para resolver diversos problemas de enrutamiento de vehículos, ya que no sería necesario programar cómo calcular eficientemente el costo de una solución vecina, y programar eso suele tomar mucho más de 5 veces el tiempo de corrida del algoritmo.

La propuesta realizada tiene una limitación importante, y es que solo puede usarse en aquellos problemas en los que las penalizaciones se realicen únicamente después de analizar todos los clientes de la ruta. Esto se debe a la forma en que se almacena la información y los valores de las variables en el grafo de evaluación.

Para trabajos futuros se recomienda modificar la construcción del grafo para poder considerar problemas en los que sea necesario penalizar determinados valores antes de terminar de procesar la ruta. Ejemplo de estos problemas son el problema con Ventanas de Tiempo y el problema con Entrega y Recogida Simultáneas. También se propone realizar una experimentación más exhaustiva para estudiar los tiempos de corrida de la propuesta.

Suplementos

El código de programación utilizado en este artículo estará disponible para los autores que lo requieran al autor para correspondencia.

Agradecimientos

Los autores agradecen a los revisores por sus sugerencias y recomendaciones.

Conflictos de interés

Se declara que no existen conflictos de interés.

Contribución de autoría

Conceptualización F.R.R.F., J.J.R.S.

Análisis formal F.R.R.F.

Investigación F.R.R.F., J.J.R.S.

Metodología F.R.R.F., J.J.R.S.

Administración de proyecto F.R.R.F.

Software J.J.R.S.

Supervisión F.R.R.F.

Validación F.R.R.F., J.J.R.S.

Visualización F.R.R.F.

Redacción: preparación del borrador original F.R.R.F.

Redacción: revisión y edición F.R.R.F., J.J.R.S.

Referencias

[1] Arnold, F. y K. Sorensen: *Knowledge-guided local search for the vehicle routing problem*. Computers & Operations Research, 105:32–46, 2019. https://www.sciencedirect.com/science/article/pii/S0305054819300024.

[2] Asghari, M., S. Mirzapour Al-e hashem y J. Mohammad: *Green vehicle routing problem: A state-of-the-art review*. International Journal of Production Economics, 231, jan 2021. https://www.sciencedirect.com/science/article/pii/S0925527320302607.

[3] Dantzig, G.B. y J.H. Ramser: *The Truck Dispatching Problem*. Management Science, 6, Octubre 1959. https://pubsonline.informs.org/doi/abs/10.1287/mnsc.6.1.80.

[4] Derbel, H., B. Jarboui y P. Siarry: *Green Transportation and New Advances in Vehicle Routing Problems*. Springer, 2020, ISBN 9783030453114. https://link.springer.com/book/10.1007/978-3-030-45312-1.

[5] Erdelic, T. y T. Caric: *A Survey on the Electric Vehicle Routing Problem: Variants and Solution Approaches*. Journal of Advanced Transportation, 2019, may 2019. https://onlinelibrary.wiley.com/doi/abs/10.1155/2019/5075671.

[6] Gendreau, M. y J. Y. Potvin: *Handbook of Metaheuristics*. International Series in Operations Research & Management Science 272. Springer International Publishing, 3rd edición, 2019. https://link.springer.com/book/10.1007/978-3-319-91086-4.

[7] Ghorbani, E., M. Alinaghian, G.B. Gharehpetian, S. Mohammadi y G. Perboli: *A Survey on Environmentally Friendly Vehicle Routing Problem and a Proposal of Its Classification*. Sustainability, 12, oct 2020. https://www.mdpi.com/2071-1050/12/21/9079.

[8] Goel, R. y R. Maini: *Vehicle routing problem and its solution methodologies: a survey*. International Journal of Logistics Systems and Management vol. 28 iss. 4, 28, 2017. https://www.inderscienceonline.com/doi/abs/10.1504/IJLSM.2017.087786.

[9] Griewank, A. y A. Walther: *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Society for Industrial and Applied Mathematic, second edición, 2008. https://epubs.siam.org/doi/pdf/10.1137/1.9780898717761.bm.

[10] Groer, C., B. Golden y E. Wasil: *A library of local search heuristics for the vehicle routing problem*. Mathematical Programming Computation, 2, apr 2010. https://link.springer.com/article/10.1007/s12532-010-0013-5.

[11] Hansen, P., N. Mladenovic y J.A. Moreno Perez: *Variable neighbourhood search: methods and applications*. 4OR: A Quarterly Journal of Operations Research, 6, nov 2008. https://link.springer.com/article/10.1007/s10288-008-0089-1.

[12] Hof, J. y M. Schneider: *An adaptive large neighborhood search with path relinking for a class of vehicle routing problems with simultaneous pickup and delivery*. Networks, 74, apr 2019. https://onlinelibrary.wiley.com/doi/abs/10.1002/net.21879.

[13] Laporte, G.: *Fifty Years of Vehicle Routing*. Transportation Science, 43, nov 2009. https://pubsonline.informs.org/doi/abs/10.1287/trsc.1090.0301.

[14] Lima, I. y colaboradores: *CVRPLIB: Capacitated Vehicle Routing Problem Library*. http://vrp.atd-lab.inf.puc-rio.br.

[15] Marwa, A., T. Said, J. Bassem y E. Mansour: *A variable neighborhood search algorithm for the capacitated vehicle routing problem*. Electronic Notes in Discrete Mathematics, 58, Abril 2017. https://www.sciencedirect.com/science/article/pii/S1571065317300665.

[16] Mohamed, N.H., S. Salhi, G. Nagy y N.A. Mohamed: *A matheuristic approach for the split delivery vehicle routing problem: an efficient set covering-based model with guided route generation schemes*. International Journal of Mathematics in Operational Research, 2019. https://www.inderscienceonline.com/doi/abs/10.1504/IJMOR.2019.101613.

[17] Montagné, R., D. Torres Sanchez y H.O. Storbugt: *VRPy: A Python package for solving a range of vehicle routing problems with a column generation approach*. Journal of Open Source Software, 5(55):2408, 2020. https://joss.theoj.org/papers/10.21105/joss.02408.pdf.

[18] Mor, A. y M.G. Speranza: *Vehicle routing problems over time: a survey*. 4OR: A Quarterly Journal of Operations Research, mar 2020. https://link.springer.com/article/10.1007/s10479-021-04488-0.

[19] Pop, P.C. y A. Horvat-Marc: *Local search heuristics for the generalized vehicle routing problem*. En *2012 International Conference on System Modeling and Optimization (ICSMO 2012), IPCSIT*, volumen 23, páginas 84–87, 2012. https://www.academia.edu/download/41333655/Local_search_heuristics_for_the_generali20160119-18577-18g8ng.pdf.

[20] Praveen, V., P. Keerthika, A. Sarankumar y G. Sivapriya: *A Survey on Various Optimization Algorithms to Solve Vehicle Routing Problem*. En *IEEE 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*. IEEE, mar 2019. https://ieeexplore.ieee.org/abstract/document/8728417/.

[21] Prodhon, C. y C. Prins: *Metaheuristics for Vehicle Routing Problems*, páginas 407–437. Springer International Publishing, 2016. https://link.springer.com/chapter/10.1007/978-3-319-45403-0_15.

[22] Tolga, K.C., J. Ola y L. Gilbert: *Thirty years of heterogeneous vehicle routing*. European Journal of Operational Research 2016-feb vol. 249 iss. 1, 249, feb 2016. https://www.sciencedirect.com/science/article/pii/S0377221715006530.

[23] Toth, P. y D. Vigo: *The Vehicle Routing Problem Problems, Methods, and Applications*. Monographs on Discrete Mathematics and Applications. SIAM, 2ª edición, 2014, ISBN 1611973589. https://epubs.siam.org/doi/abs/10.1137/1.9781611973594.fm.

[24] Villavicencio Sánchez, E.D. Nuñez de: *Propuesta de exploración ágil de cualquier vecindad en un VRP*. Tesis de Diploma, Facultad de Matemática y Computación, Universidad de La Habana, 2018.

[25] Wang, X., Tsan Ming Choi, Zhiying Li y Shuai Shao: *An Effective Local Search Algorithm for the Multidepot Cumulative Capacitated Vehicle Routing Problem*. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 50(12):4948–4958, 2020. https://ieeexplore.ieee.org/abstract/document/8844993/.