UNIVERSIDAD CATÓLICA
“Nuestra Señora de la Asunción”

Facultad de Ciencias y Tecnología
Departamento de Electrónica e Informática

Ingeniería de Tráfico en Redes Dinámicas Periódicas Utilizando el Algoritmo SPEA

Trabajo de Tesis
como requisito para obtener el título de
Ingeniero en Informática

Tutor: Prof. D.Sc. Benjamín Barán

Autor: Uri Yael Rozenworcel

Junio 2009
Asunción - Paraguay

Índice General

Índice General *2*

Índice de Figuras *4*

Índice de Tablas *6*

Índice de Algoritmos *7*

Siglas y Acrónimos *8*

Notación Utilizada *10*

1. Introducción *13*

2. Ruteo en Redes Dinámicas Periódicas *17*

2.1. Algoritmos No Orientados a Grafos Evolutivos **18**

2.1.1. Destination-Sequenced Distance-Vector 19

2.1.2. Dynamic Source Routing 22

2.1.3. Ad-hoc On-Demand Distance Vector 27

2.2. Algoritmos Mono-Objetivo Orientados a EG **28**

2.2.1. Evolving Graph Foremost 31

2.2.2. Evolving Graph Shortest 34

3. Problema de Optimización Multi-objetivo *37*

4. Algoritmo Evolutivo Multi-Objetivo *41*

4.1. Strength Pareto Evolutionary Algorithm **47**

4.2. Reformatorio **57**

5. Métricas de Eficiencia *61*

6. Simulaciones *63*

6.1. Topología de la Red **63**

6.2. Carga de Tráfico **66**

6.3. Parámetros del Algoritmo SPEA **67**

7. Resultados *70*

7.1. Simulaciones en Redes Pequeñas **70**

7.1.1. Saltos Hasta Llegar a Destino 72

7.1.2. Eras Vividas Hasta Llegar a Destino 74
7.1.3. Eras Viajadas Hasta Llegar a Destino 76
7.2. Simulaciones en Redes Medianas 77
7.3. Simulaciones en Redes Grandes 78
7.4. Variación de los Parámetros del Algoritmo SPEA 80
7.4.1. Tamaño de la Población 81
7.4.2. Cantidad de Iteraciones 82
7.4.3. Probabilidades en el Reformatorio 82
8. Conclusiones y Trabajos Futuros 83
9. Referencias 85
Anexo I. Dijkstra Shortest Path Algorithm 88
Anexo II. Tráfico pesado o congestionado para la red mediana de la Figura 18 89
Anexo III. Tráfico pesado o congestionado para la red grande de la Figura 19 91

Índice de Figuras

Figura 1. Topología de una red dinámica periódica 15

Figura 2. Matriz de adyacencia 18

Figura 3. Una topología de red y la tabla calculada con el algoritmo DSDV en el nodo A. . 21

Figura 4. Mecanismo Route Discovery 16

Figura 5. Ejemplo de representación de un grafo evolutivo 30

Figura 6**.** Caminos en una red dinámica periódica 31

Figura 7. Ejemplo de tres paquetes 32

Figura 8. Cromosoma con genes 42

Figura 9. La población *P* luego de evolucionar I_k veces 43

Figura 10. Ejemplo de efectuar la operación genética de cruzamiento 44

Figura 11. Ejemplo de efectuar la operación genética de mutación 45

Figura 12. Diagrama de flujo de un Algoritmo Evolutivo simple 46

Figura 13. Posible solución al problema de encaminar paquetes 50

Figura 14. Diagrama de flujo del algoritmo SPEA 51

Figura 15. Una red fragmentada 56

Figura 16. Una red con islas 61

Figura 17. Una red pequeña 63

Figura 18. Una red mediana 64

Figura 19. Una red grande 65

Figura 20. Soluciones para una red pequeña 71

Figura 21. Saltos en el camino 73

Figura 22. Saltos en el camino con *SPEA* 74

Figura 23. Eras vividas 75

Figura 24. Eras vividas con *SPEA* 75

Figura 25. Eras viajadas 76

Figura 26. Soluciones para una red mediana 78

Figura 27. Soluciones para una red grande 79

Índice de Tablas

Tabla 1. Eras en las que los enlaces de la Figura 18 se encuentran activos 66

Tabla 2. Tráfico pesado o congestionado para la red pequeña de la Figura 17 67

Tabla 3. Valores asignados a los parámetros utilizados en el algoritmo SPEA para la red dinámica periódica pequeña 68

Tabla 4. Valor promedio de las funciones objetivo calculadas para las soluciones obtenidas en la simulación utilizando la red pequeña mostrada en la Figura 17 72

Tabla 5. Ejemplo de valores asignados a los parámetros utilizados en el algoritmo SPEA para la red dinámica periódica mediana 77

Tabla 6. Valor promedio de las funciones objetivo calculadas para las soluciones obtenidas en la simulación utilizando la red mediana mostrada en la Figura 18 77

Tabla 7. Ejemplo de valores asignados a los parámetros utilizados en el algoritmo SPEA para la red dinámica periódica grande 78

Tabla 8. Valor promedio de las funciones objetivo calculadas para las soluciones obtenidas en la simulación utilizando la red grande mostrada en la Figura 19 79

Tabla 9. Simulaciones en una red pequeña 81

Índice de Algoritmos

Algoritmo 1. Pseudocódigo de secciones del algoritmo DSDV 20

Algoritmo 2. Pseudocódigo de secciones del algoritmo DSR 23

Algoritmo 3. Pseudocódigo del algoritmo EG Foremost 33

Algoritmo 4. Pseudocódigo del algoritmo EG Shortest 35

Algoritmo 5. Pseudocódigo del algoritmo SPEA implementado en este trabajo 55

Algoritmo 6. Pseudocódigo de la función reformatorio 59

Siglas y Acrónimos

AODV	Ad-hoc On-Demand Distance Vector
DSDV	Destination-Sequenced Distance-Vector
DSR	Dynamic Source Routing
EA	Evolutionary Algorithm - Algoritmo Evolutivo
EG	Evolving Graph
EG Foremost	Evolving Graph Foremost
EG Shortest	Evolving Graph Shortest
FSDN	Fixed Schedule Dynamic Networks
GA	Genetic Algorithms - Algoritmos Genéticos
HELLO	Paquete de control utilizado en el descubrimiento de nodos en una red dinámica
LEO	Low Earth Orbit - Satélites de Órbita Baja
MANET	red móvil de tipo ad-hoc
MOEA	Multi-Objective Evolutionary Algorithm - Algoritmo Evolutivo Multi-Objetivo
MOP	Multiobjective Optimization Problem - Problema de Optimización Multi-objetivo
NSGA2	Non-dominated Sorting Genetic Algorithm II
OLPC	One Laptop Per Child
QoS	Quality of Service – Calidad de Servicio

RREP	Paquete del tipo Route Replay
RREQ	Paquete del tipo Route Request
RERR	Paquete del tipo Route Error
SPEA	Strength Pareto Evolutionary Algorithm
SPEA2	Strength Pareto Evolutionary Algorithm II

Notación Utilizada

1a, …, 1z, 2a, ..., 2z,…	Identificadores de nodos que forman conceptualmente un sub grupo denominado según el número.
A, B, C, ..., Z	Identificadores de nodos
[A B C]	Camino formado por los nodos A, B y C
E	Conjunto de eras definidas en un Grafo Evolutivo
e	Una era perteneciente al conjunto de eras *E*
e_{now}	Era en la que se encuentra el sistema
EG	Evolving Graph – Grafo Evolutivo
f	Una función objetivo
f_i	Vector correspondiente a valorar una solución P_i en $F(P_i)$
$fitness_i$	Valor de adaptabilidad de un individuo *i* pertenecienta a la población *P*
$fitness_j$	Valor de adaptabilidad de un individuo *j* pertenecienta a la población externa *P'*
F(x)	Función vectorial compuesta por funciones objetivo
G(x)	Restricciones para las funciones objetivo de F(x)
I_0	Iteración inicial del algoritmo en la evolución de la población
I_k	Iteración número *k* del algoritmo en la evolución de la población

I_{max}	Cantidad máxima de iteraciones que el algoritmo evolutivo ejecuta
$\Re$	Dominio de los números reales
T	Período
t	Instante
N	Conjunto de nodos existentes en un grafo evolutivo
n	Nodo perteneciente al conjunto N
n_r	Nodo remitente
n_d	Nodo destinatario
$<n, e>$	Par que define un salto, compuesto por un nodo y una era
$<n_r, n_d, e>$	Paquete con n_r y n_d como nodos remitente y destinatario, disponible desde la era e
V_G	Conjunto de enlaces existentes en un grafo evolutivo
P	Población genética compuesta por individuos
P_i	Individuos de la población P
P_{max}	Cantidad máxima de individuos que puede contener la población P
p_{nuevo}	Individuo auxiliar
P_{nueva}	Población genética auxiliar
$\|\|P\|\|$	Cantidad de individuos que integran la poblacón P
P'	Población genética externa compuesta por soluciones no dominadas

P'_{max}	Cantidad máxima de individuos que puede contener la población externa *P'*
$strength_i$	Valor de fuerza de un individuo *i* pertenecienta a la población *P*
$strength_j$	Valor de fuerza de un individuo *j* pertenecienta a la población externa *P'*
x	Conjunto de genes que forman una solución o individuo
y	Vector correspondiente a valorar una solución x en F(x)

1. Introducción

Una red dinámica está compuesta por nodos o estaciones que se conectan y desconectan de la red, ya sea porque se desactivan o porque dejan de estar en el rango de alcance [1, 4]. La autonomía energética de los dispositivos es de central importancia en los sistemas. Los dispositivos pueden ser desactivados a fin de ahorrar energía. Dicha estrategia implica retrasar el envío de información. La decisión de reactivar los dispositivos puede darse por diferentes motivos, por ejemplo, un cronograma o la existencia de una cantidad mínima disponible de datos a ser transmitidos. Los dispositivos hacen uso eficiente del tiempo de actividad (sin permanecer ociosos) si durante el mismo, la disponibilidad de información a ser enviada es alta. Dos dispositivos que forman parte de la red y son adyacentes (i.e. existe un enlace entre ellos), pueden perder repentinamente la conexión por motivos de distancia geográfica, por baja calidad en la señal como efecto de diferentes tipos de interferencia o por obstáculos que se interpongan.

Ejemplos de redes dinámicas son las formadas por estaciones portátiles, donde puede no existir influencia por parte de los dispositivos acerca de su futura posición o traslación. Estas redes son llamadas MANET (red móvil de tipo ad-hoc) [11, 12, 13, 28]. Las redes MANET pueden ser utilizadas por grupos de exploradores movilizándose cada uno a través de distintos senderos en un terreno desconocido; vehículos que poseen dispositivos de red y transitan dentro de una ciudad; un escuadrón de rescate que se interna en una zona de desastre y donde cada integrante mantiene contacto comunicándose a través de una estación incorporada; además, una red dinámica puede ser utilizados en ambientes educativos para formar una malla, como en el proyecto OLPC [6]. En dichos ejemplos, la distancia y los obstáculos entre estaciones pueden impedir una conexión constante.

En este trabajo se consideran las redes dinámicas periódicas. Estas redes, aunque son dinámicas, tienen el mismo estado cada un período T (i.e. el conjunto de nodos que forma la red y sus respectivos enlaces, son los mismos luego de haber transcurrido dicho período). Es decir que la red tiene la misma topología en los instantes t y $t+T$.

El lapso de tiempo durante el cual un nodo deja de estar conectado a la red, puede ser planificado, conocido o desconocido. En el caso de las redes dinámicas periódicas enfocadas en el presente trabajo, se cuenta con una descripción del comportamiento de los nodos que la componen. En este sentido, existe una descripción de todos los nodos miembros de la red, de sus periodos de conectividad y de la activación y desactivación de los enlaces entre ellos.

La red puede ser *ad-hoc* o tener un ente centralizador que coordine la incorporación de las estaciones [1, 4, 12, 24, 28]. En el caso de ser *ad-hoc*, los nodos tienen la responsabilidad de modificar su conocimiento acerca de la topología de la red, descubriendo a los nuevos nodos e identificando a los que dejan de formar parte de la misma. Por el otro lado, si existe un ente centralizador, un nodo que ingresa a la red debe anunciarse a dicho ente brindando la información de su estado; luego, a través de un esquema de actualización, el ente centralizador y los nodos de la red dinámica entran en contacto para transferir información y actualizar su conocimiento acerca de la topología.

Ejemplos de redes dinámicas periódicas son las formadas por:

- Satélites LEO (*Low Earth Orbit* o *Low Orbit Satellites*) que entran y salen de rango y donde la red tiene la misma topología cada un período *T* de tiempo [8].

- Redes de sensores que se activan y desactivan en un horario determinado y debidamente planificado. De igual forma al ejemplo anterior, existe un período *T* luego del cual la topología de la red es la misma [23, 24].

- Redes de comportamiento previsible, en inglés *Fixed Schedule Dynamic Networks* (FSDN). En dichas redes se pueden predecir los lapsos de actividad de cada nodo y de sus enlaces [18, 19].

Para transmitir un paquete desde un nodo fuente (*remitente*) hasta uno destino (*destinatario*), debe encontrarse un camino entre ellos. Los nodos intermedios forman parte del camino y reenvían cada paquete hasta su destino. Puede ocurrir, en un instante dado, la inexistencia de un camino completo entre los nodos remitente y destinatario porque estos, o los nodos intermedios, no se encuentran en la red ya sea por una imposibilidad de establecer y mantener una conexión, o por encontrarse inactivos. Sin embargo, los nodos pueden guardar el paquete en memoria, esperando un momento posterior donde la topología haya variado y se pueda avanzar en el camino hacia el nodo destinatario.

El *camino* incorpora lo que se puede inferir como una dimensión adicional: *la era*, delimitada como un lapso en el tiempo. Para representar estas características cambiantes se utilizan grafos evolutivos (*Evolving Graphs* - EG). La Figura 1 muestra la topología de una red

dinámica periódica que define cuatro eras distintas. La topología inicial se representa en (1), mientras en (2) se aprecia la traslación efectuada por los nodos denominados “a” y “b”, dejando de existir un enlace entre los nodos “b” y “e”. Además, el nodo “f” deja de compartir un enlace con “e” y establece uno con el nodo “c” pese a, aparentemente, no haber modificado su posición geográfica. La red presentada en la Figura 1 es periódica y su topología, luego de la mostrada en (4), vuelve a ser la descripta en (1).

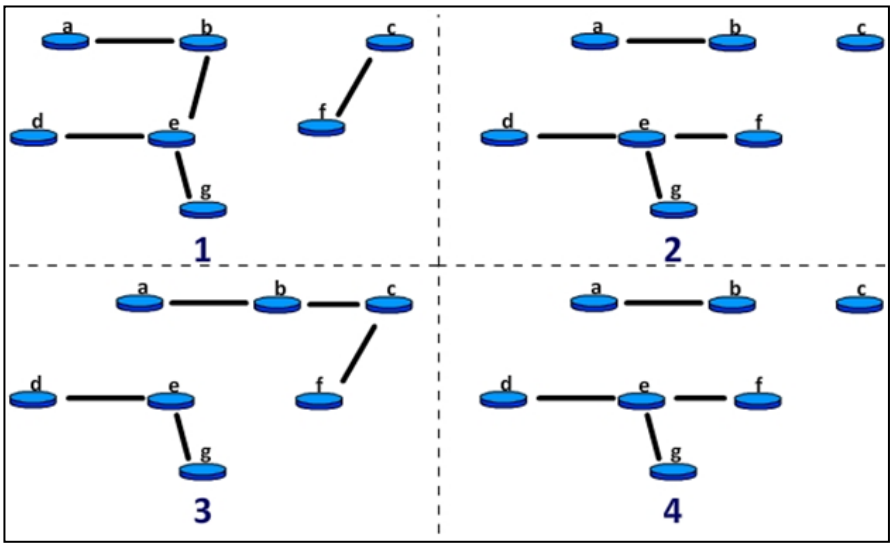


Figura 1. Topología de una red dinámica periódica. 1) Topología inicial; **2)** Los nodos *a* y *b* se alejan del nodo *e* y se acercan al nodo *c*, *f* se acerca a *e*; **3)** *b* se conecta a *c*, y el nodo *f* vuelve a acercarse a *c*; **4)** *a* y *b* se alejan de *c*, mientras que *f* vuelve a acercarse a *e*.

Los algoritmos de encaminamiento publicados hasta la fecha son del tipo mono-objetivo, entre ellos los comúnmente utilizados para realizar el encaminamiento en redes dinámicas periódicas son: *Destination-Sequenced Distance-Vector* (DSDV) [9], *Dynamic Source Routing* (DSR) [12] y *Ad-hoc On-Demand Distance Vector* (AODV) [26]. Trabajos recientes estudian en forma extensa a dichos algoritmos [18, 24, 25, 28, 29]; los mismos, por lo general se centran en minimizar la cantidad de saltos realizados por un paquete hasta llegar a destino (*shortest journey*), en lograr que el horario y fecha de llegada a destino sea el más temprano posible (*foremost journey*) o que el paquete utilice llegue a destino lo más rápido posible, minimizando así el tiempo de viaje (*fastest journey*) [4, 18].

Luego de la introducción de los Grafos Evolutivos como formalismo para representar la topología cambiante de las redes [3, 11, 14, 19], dos algoritmos mono-objetivo denominados *Evolving Graph Foremost* y *Evolving Graph Shortest* fueron publicados por Monteiro en el año

2007. Dichos algoritmos basados en Grafos Evolutivos son óptimos, cada uno en su objetivo específico, y fue demostrado en [18] que encuentran soluciones igualmente buenas o inclusive mejores, si existen, a las encontradas por los algoritmos *DSDV*, *DSR* y *AODV*.

El presente trabajo propone, por vez primera, la utilización de un *Algoritmo Evolutivo Múlti-Objetivo* capaz de hallar un conjunto de soluciones Pareto. El mismo no sólo contiene individuos competitivos con las soluciones calculadas por otros algoritmos que forman parte del estado del arte, sino además, se logra calcular individuos que dominan a las soluciones consideradas como óptimas cuando una sola función objetivo es considerada. Esto se debe a la capacidad de mejorar, en un contexto multi-objetivo, a las soluciones que pueden ser consideradas como óptimas en un entorno mono-objetivo.

En lo que sigue, la sección 2 presenta a los algoritmos utilizados actualmente para realizar el ruteo en redes dinámicas periódicas. Dichos algoritmos se dividen en los no orientados a grafos evolutivos (sección 2.1.) y los algoritmos mono-objetivo que sí hacen uso de los grafos evolutivos como formalismo para representar redes dinámicas (sección 2.2.). En la sección 3 se modela el problema del ruteo en redes dinámicas periódicas como un Problema de Optimización Multi-objetivo, o por sus siglas en inglés (MOP). Más adelante, la sección 4 propone un Algoritmo Evolutivo Multi-Objetivo, en inglés *Multi-Objective Evolutionary Algorithm* (MOEA) para resolver el problema. Específicamente, en dicha sección se propone la implementación del algoritmo *Strength Pareto Evolutionary Algorithm* (SPEA). Luego, la sección 5 define las métricas utilizadas para evaluar a las soluciones, la sección 6 presenta las simulaciones llevadas a cabo en diferentes topologías de redes y especificando además el tráfico que será introducido a la red dinámica periódica. Basándose en la sección anterior, la sección 7 presenta los resultados de las simulaciones llevadas a cabo mostrando el resultado de utilizar en las simulaciones, los diferentes algoritmos y sus variaciones de parámetros. Por último, en la sección 8 se presentan las conclusiones y propuestas de trabajos futuros.

2. Ruteo en Redes Dinámicas Periódicas

Los algoritmos de ruteo pueden ser catalogados como reactivos o proactivos. En el caso de los reactivos, el cálculo del camino se retrasa hasta la existencia de necesidad por utilizar dicha información (e.g. los datos son requeridos porque existe un paquete a ser enviado a un nodo destinatario, y es necesario descubrir la ruta actual que debe tomar para ser encaminado). Por el otro lado, los algoritmos proactivos realizan el cálculo de los caminos sin ser aun requerida la información, por ejemplo realizan el cálculo durante momentos ociosos, cuando detectan poca cantidad de tráfico recorriendo la red y la misma no se encuentra saturada. De esta forma, se cuenta con una estrategia para la distribución de carga de cómputo en el nodo, realizando intentos por no saturar a la red.

Algunos algoritmos utilizan el encaminamiento en el origen. Esto es, el nodo que crea el paquete de datos define el camino completo y lo almacena en la cabecera, detallando cada uno de los tramos que recorrerá dicho paquete. Otros algoritmos dejan a criterio de cada nodo intermedio, la decisión acerca de cuál será el siguiente nodo intermedio en el camino hasta el nodo destinatario.

Los conocidos algoritmos *Destination-Sequenced Distance-Vector* (DSDV) [9], *Dynamic Source Routing* (DSR) [12] y *Ad-hoc On-Demand Distance Vector* (AODV) [26], no contemplan la existencia de *eras*, y no utilizan grafos evolutivos para describir la topología de una red. En contrapartida, los algoritmos *Evolving Graph Foremost* (EG Foremost) y *Evolving Graph Shortest* (EG Shortest) propuestos en [18] así como el *Strength Pareto Evolutionary Algorithm* (SPEA) implementado en este trabajo, incorporan la utilización de los grafos evolutivos para describir la topología cambiante de la red dinámica, incluyendo así el concepto de *eras*.

En la Sección 1 se introdujo la equivalencia de topologías en las redes periódicas para los instantes t y $t+T$. Se dice que dos eras pertenecen a períodos distintos, si entre ambas transcurrieron al menos T eras. En las redes dinámicas periódicas pueden adoptarse dos formas de referirse a eras de diferentes períodos. Considérese una red periódica que denota con el número uno a la primera era. Existen dos formas distintas de referir una misma era: reiniciar la cuenta a la menor de las eras, o continuar con la numeración. A modo de ejemplo, en una red periódica compuesta por quince eras, es equivalente referir como uno o dieciséis, a la era sucesora de la número quince.

En el presente trabajo, en lo que respecta a las representaciones gráficas, se optará por reiniciar la cuenta; por otro lado, los cálculos matemáticos son más sencillos si se sigue la numeración sin reiniciarla, y se adopta esta opción en lo referente a cálculos matemáticos.

2.1. Algoritmos No Orientados a Grafos Evolutivos

Originalmente, los algoritmos no orientados a grafos evolutivos no consideran las *eras* en las que un enlace existe. Las decisiones se basan en información provista por mediciones como la cantidad de saltos o la cantidad de eras que toma la llegada de una respuesta desde un nodo distante. Esto se logra enviando paquetes de control y realizando los cálculos cuando se recibe una respuesta a dicho paquete de control. Otra estrategia, es la de realizar mediciones acerca de diferentes tasas, como la de error por falta de ruta, error por memoria insuficiente, entre otras. En [12, 18, 20, 26] para la toma de decisiones acerca del encaminamiento se opta, si es necesario, por el método de mediciones basadas en paquetes de control y no en las otras métricas arriba citadas.

La descripción de la topología de la red puede ser representada por matrices de adyacencia como la ilustrada en la Figura 2. Una matriz de adyacencia debe especificar la conectividad de cada par de nodos vecinos, y las eras en que los enlaces entre ellos se encuentran activos. Si bien dichas matrices incluyen la información completa, la organización de los datos no facilita la búsqueda de diversas opciones ni de sus consecuencias a la hora de elegir un instante en particular en el cual se hará el envío de un paquete, utilizando un enlace dado. En la Figura 2, entre los nodos “A” y “B” existe un enlace activo en las eras 3 y 10, entre “A” y “C” en las eras 1, 7 y 13, y así sucesivamente.

	A	**B**	**C**	**D**	**E**
A		3, 10	1, 7, 13		
B	3, 10		7, 10	1, 5, 10	1, 12
C	1, 7, 13	7, 10			3, 7, 11
D		1, 5, 10			6, 10, 15
E		1, 12	3, 7, 11	6, 10, 15	

Figura 2. Matriz de adyacencia. Representación matricial de una red dinámica periódica que define 15 eras.

En el caso de los algoritmos proactivos, la información de cambios en la red toma un tiempo considerable en expandirse a través de la red. Dado este hecho, es usual que los nodos

no tengan una representación actualizada de la red. En cambio, los algoritmos reactivos generalmente realizan los cálculos en un tiempo posterior al de los proactivos, por lo que contemplan los cambios que se produjeron en la red. Los algoritmos reactivos no garantizan que un nodo contenga información correcta, ya que puede suceder que ocurran cambios en la red durante el lapso de tiempo transcurrido mientras el camino es calculado.

A continuación, se presentan los algoritmos comúnmente utilizados para realizar el encaminamiento en redes dinámicas periódicas [18, 24, 25, 28, 29]. Primeramente se desarrolla el algoritmo *Destination-Sequenced Distance-Vector* (DSDV) [9]; luego, *Dynamic Source Routing* (DSR) [12] y finalmente *Ad-hoc On-Demand Distance Vector* (AODV) [26].

2.1.1. **Destination-Sequenced Distance-Vector**

El algoritmo *Destination-Sequenced Distance-Vector* (DSDV) fue publicado por Ford y Fulkerson en el año 1958 [9], está basado en el algoritmo *Distributed Bellman-Ford* [20, 25]. Es un algoritmo basado en tablas y catalogado como proactivo, dado que cada nodo periódicamente intercambia información con sus vecinos –nodos adyacentes -. La decisión del próximo salto (i.e. siguiente nodo al cual se enviará el paquete) es tomada en cada nodo intermedio. Ello permite, de ser posible, efectuar los cambios necesarios en las decisiones de encaminamiento ante modificaciones súbitas en la topología de la red, en vez de fracasar en su intento de alcanzar al nodo destinatario.

El Algoritmo 1 muestra algunas de las funciones utilizadas por el *DSDV* para mantener actualizadas las tablas de ruteo**.** La función ***actualizar_en_tabla*** recibe como parámetros a un nodo destinatario y un arreglo que contiene la información compartida por un nodo vecino acerca del camino a seguir para llegar al nodo destinatario. La función ***intercambiar_info*** es la encargada de desglosar la tabla recibida de un nodo vecino, e invocar a ***actualizar_en_tabla*** de forma tal a actualizar la información, si corresponde. Por último, la función ***publicar_tabla*** toma de a uno a los nodos que conforman la red, para intercambiar su tabla de ruteo con la de los vecinos, invocando a la función ***intercambiar_info***.

Cada nodo almacena en una tabla cuál es el siguiente nodo a través del que se debe encaminar un paquete para alcanzar a un nodo destinatario dado. La entrada de la tabla para cada nodo destino, contiene los campos: próximo nodo, costo del camino y un número de secuencia que es utilizado para comparar cuál de entre dos informaciones es la más nueva (ver Figura 3). Cuando un nodo recibe la tabla de actualización de un vecino, controla qué destinos

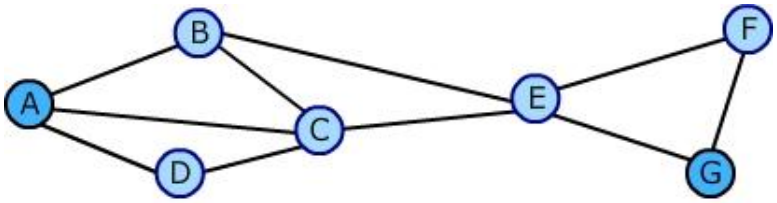
conoce dicho nodo adyacente, y luego compara el número de secuencia de la información para cada destino con el de los destinos que ya conocía. Si la secuencia es la misma y el costo del camino propuesto por el vecino es menor, entonces actualiza su tabla, si la secuencia es mayor, interpreta que el nodo destino actualizó su estado y debe necesariamente copiar el valor (ver función ***actualizar_en_tabla*** del Algoritmo 1).

```
/*

Variables:

tabla: arreglo perteneciente a cada nodo, indexado según el nodo
destino y contenedor de la información de encaminamiento.

info: arreglo que contiene sec el número de secuencia de la
información, el costo del camino y el siguiente nodo en el mismo.

*/

Función actualizar_en_tabla(nodo, info)
   1. sec_info ← secuencia de la información almacenada en tabla[nodo]
   2. Si sec > sec_info o si seq = sec_info y costo < tabla[nodo]
      • Actualizar la información de tabla[nodo] con el parámetro info
   3. Sino
      • Completar tabla[nodo] con el parámetro info
   4. Fin
Fin

Función intercambiar_info(nodo, tabla)
   1. Para cada índice vecino de tabla hacer
      • Llamar a actualizar_en_tabla(vecino, tabla[vecino])
   2. Fin
Fin

Función publicar_tabla
   1. Para cada nodo en la red hacer
      • Llamar a intercambiar_info(nodo, tabla)
   2. Fin
Fin
```

Algoritmo 1. Pseudocódigo de secciones del algoritmo *DSDV*. Funciones utilizadas para intercambiar información de control entre nodos.

Cada nodo es el encargado de aumentar el número de secuencia de la información acerca de cómo llegar a él mismo, comportándose como destinatario. Es decir, un nodo no puede modificar el número de secuencia de la información para otro nodo destino. Dicha información acerca de los cambios en la topología debe atravesar la red diametralmente, actualizando de esta forma a todos los nodos que la conforman.

Considerando a la red como del tipo *ad-hoc* y dado el esquema de actualización de la información acerca del estado de la red antes descripto, puede ocurrir que un nodo no posea información acerca de cómo llegar hasta otro nodo en particular, o que desconozca por completo la existencia misma de dicho nodo. En caso de recibir un paquete y si el mismo debe llegar a un destino acerca del cual no tiene información, el nodo decide descartar a dicho paquete por falta de ruta, retornando al origen un paquete de control que informa del error. Esto sucede especialmente cuando se desea enviar un paquete a un nodo destinatario recientemente incorporado a la red, si la información de su ingreso aun no se ha extendido a todos los otros nodos de la topología.



Dest.	A través	Dist.	Sec.
B	B	1	18
C	C	1	24
D	D	1	60
E	C	2	92
F	C	3	124
G	C	3	196

Figura 3. Una topología de red y la tabla calculada con el algoritmo DSDV en el nodo A. El nodo origen A necesita enviar un paquete a G, para ello consulta la tabla de encaminamiento, donde se indica que debe enviar al nodo C.

En el algoritmo *Destination-Sequenced Distance-Vector,* las actualizaciones se transmiten cada vez que los nodos intercambian sus tablas de ruteo. En la Figura 3 el nodo G se encuentra, en el mejor de los casos, a 3 saltos de distancia del nodo A. Suponiendo que el intercambio lo realizan todos los nodos al mismo tiempo y cada un período de tiempo *T*, deben realizarse como mínimo tres intercambios hasta que la información del nodo G llegue al nodo A y los cambios sean realizados en la tabla de ruteo del nodo A. En el peor de los casos, la actualización tomará una cantidad de intercambios igual al diámetro de la red (i.e. la longitud o cantidad de saltos del camino más largo entre los caminos más cortos). El algoritmo *Distributed Bellman-Ford* sufre del problema de *cuenta hasta el infinito* (Tanenbaum, A.S. Computer Networks, cuarta edición,

sección 5.2., página 359), el cual ocurre cuando se desconecta un nodo, dejando de formar parte de un camino largo.

Como ejemplo de la *cuenta hasta el infinito*, considérese una red en la que los nodos R y T son vecinos; además, existe un camino [R S T U V] que llega desde el nodo R hasta V. Si el nodo U deja de formar parte de la red, entonces el nodo T detecta la falta de camino hasta V. No obstante, y siendo que los nodos R y T son vecinos, durante el proceso de intercambio de tablas el nodo T decide encaminar los paquetes hasta V a través de R. Esto produce la siguiente situación: el nodo R encamina paquetes hacia V a través de T, y al mismo tiempo el nodo T lo hace a través de R, produciendo un bucle. En la situación descripta, durante cada intercambio de tablas entre los nodos R y T, el costo del camino hasta V será incrementado. Los nodos deducen la inexistencia de un camino hasta un nodo destino, cuando el costo supera una cota máxima. Dicha cota máxima generalmente es establecida en la cantidad de nodos que conforman la red más uno [25]. Sin la implementación de dicha cota máxima, los costos podrían tender a infinito y aun no detectarse la falta de ruta.

Si bien el algoritmo *DSDV* logra adaptarse a los cambios en la topología de una red, no lo hace en forma suficientemente rápida resultando ineficiente para redes altamente dinámicas, donde a menudo los nodos se conectan y desconectan, o los enlaces se activan y desactivan.

2.1.2. **Dynamic Source Routing**

El algoritmo *Dynamic Source Routing* (DSR) estudiado por Johnson y Maltz en [12], es catalogado como reactivo. El camino de un paquete de datos para ser entregado a un nodo destino es calculado cuando se requiere dicha información y no en forma anticipada. Cuando el nodo origen (i.e. el nodo que crea el paquete de datos en un primer momento) encuentra el camino completo, almacena la información del camino en la cabecera del paquete. De esta forma, cada paquete de datos contiene en su cabecera una descripción completa de la ruta a través de la cual se llega hasta el nodo destinatario, siendo indicados en orden los nodos intermedios a ser utilizados por el paquete como agentes de reenvío. Cada nodo intermedio lee la cabecera y extrae la información necesaria para enviar el paquete al siguiente nodo del camino. En el esquema descripto, los nodos son meros agentes de reenvío, y no realizan ninguna toma de decisión [12].

El Algoritmo 2 presenta el pseudocódigo de las dos funciones que implementan un mecanismo denominado *Route Discovery*, el cual se compone de dos fases. En la primera, se

utilizan paquetes del tipo *Route Request* (RREQ) para hallar un camino desde el nodo remitente hasta el destinatario; una vez alcanzado el mismo, finaliza esta primera etapa. La segunda fase hace uso de paquetes *Route Replay* (RREP) para informar al nodo remitente acerca del camino hallado en la primera fase.

```
/*
Variables:

registro: lista que contiene los números de secuencia y nombre de
remitentes para los que ya se retransmitió el RREQ

cabecera: arreglo que contiene el identificador del remitente, número de
secuencia, camino, tiempo de vida (ttl) y destinatario.
*/

Función RREQ(cabecera)
   1. Disminuir en 1 el campo ttl de cabecera
   2. Si ttl = 0, entonces salir de la función
   3. Si registro ya contiene la información de remitente y secuencia de
      cabecera, entonces salir de la función
   4. Agregar la información de remitente y secuencia a registro
   5. Agregar al campo camino de cabecera el nombre del nodo actual
   6. Si el nodo es el indicado en el campo destinatario de cabecera
         • llamar a la función RREP(cabecera)
   7. Sino, entonces retransmitir el paquete a los vecinos
Fin

Función RREP(cabecera)
   1. Si el nodo actual es el indicado en el campo destinatario de la
      cabecera y aun no se respondió al número de secuencia indicado en
      el RREQ, entonces extraer y almacenar la información de camino
   2. Sino, reenviar el paquete al siguiente nodo del camino
Fin
```

Algoritmo 2. Pseudocódigo de secciones del algoritmo *DSR*. Funciones utilizadas en el mecanismo *Route Discovery*.

El algoritmo *DSR* utiliza un mecanismo llamado *Route Discovery* para definir la ruta a ser utilizada por un paquete en su viaje desde el nodo remitente hasta el nodo destinatario. En dicho mecanismo, el remitente genera paquetes *RREQ (Route Request)* los cuales contienen un número de secuencia, tiempo de vida y el identificador o nombre del nodo destinatario al que se desea transmitir un paquete de datos. Cuando un nodo recibe un paquete *RREQ*, éste lo

retransmite a todos sus vecinos (i.e. nodos adyacentes) salvo al nodo del cual recibió el paquete. Cada nodo utiliza una lista para almacenar los números de secuencia de paquetes *RREQ* ya enviados. Al momento de retransmitir un paquete *RREQ,* se corrobora la existencia de su número de secuencia en la lista mantenida. Mientras dicho número de secuencia no exista previamente en la lista, el paquete será reenviado y el número incorporado a la lista; caso contrario, el paquete será descartado sin reenviarse. Cada nodo se agrega al camino almacenado en la cabecera antes de transmitir un paquete del tipo *Route Request*. Los paquetes *RREQ* tienen un tiempo de vida, para no permanecer viajando en busca del destino por tiempo indeterminado.

Una vez alcanzado el nodo destinatario por el primer paquete *Route Request*, dicho nodo extrae de la cabecera el camino recorrido por el paquete y crea un nuevo paquete, esta vez del tipo *RREP (Route Replay)*. Dicho paquete utilizará el camino inverso al del *RREQ* para alcanzar al nodo que inició el mecanismo *Route Discovery* (ver función *RREQ* en el Algoritmo 2). El paquete *Route Reply* viaja desde el nodo anteriormente considerado como destinatario, hasta el nodo remitente que creó el paquete *RREQ* en un primer momento. En el esquema *Route Discovery,* los paquetes del tipo *RREP* son los encargados de informar a los nodos remitente, qué ruta será la descripta en el encabezado de cada paquete de datos a ser enviado.

El nodo destino crea un único paquete *RREP* por cada secuencia diferente de *RREQ* recibida. Es decir, únicamente se responde al primer paquete *Route Request* una vez alcanzado su destino, encontrándose así el supuesto mejor camino para ese instante en particular. Los nodos intermedios por donde pasaron los paquetes *RREQ y RREP* almacenan la información en forma temporal, y si llegasen a recibir un *RREQ* de otro nodo remitente intentando llegar al mismo destinatario, aguarda al resultado del *Route Discovery* en proceso en vez de reenviar el pedido. Como la red dinámica periódica puede cambiar de topología durante el mecanismo *Route Discovery*, el camino calculado no puede asegurarse como óptimo.

Si un nodo descubre la existencia de un problema en una ruta (e.g. un enlace roto u otro problema), envía un paquete del tipo *Route Error* (RERR) informando de dicho problema al nodo remitente. El paquete *RERR* viaja a través de los nodos intermedios hasta el nodo considerado anteriormente como remitente; de esta forma, dichos nodos son informados acerca del problema. Este mecanismo logra que un problema ya detectado, sea informado lo antes posible al nodo remitente y a los nodos que actúan como agentes de reenvío. El camino recorrido por cada paquete de datos es definido en su cabecera y los nodos intermedios no

pueden alterar dicho campo. Por ello, al surgir un problema en una ruta, los paquetes que la utilicen están destinados a no poder culminar exitosamente su camino. Dicho inconveniente es solucionado recién al ejecutarse nuevamente el proceso *Route Discovery*, cuya activación puede acelerarse con el uso de los paquetes *RERR*. Algunas implementaciones del algoritmo *Dynamic Source Routing* no contemplan el uso de paquetes *Route Error*. Como consecuencia, ante el surgimiento de un problema en un camino, posiblemente mayor cantidad de paquetes de datos no llegarán a destino, respecto a aquellas implementaciones que sí contemplan informar acerca del error.

La Figura 4 describe una situación donde el nodo denominado A actúa como remitente, y necesita transmitir un paquete de datos al nodo G, el cual actúa como nodo destinatario. En la sección (a) de la figura, los paquetes *Route Request* ya transmitidos se encuentran indicados con flechas compuestas por guiones azules. Entre corchetes, encima de cada flecha, se nombran los nodos a través de los cuales ya ha pasado el paquete *RREQ*. En un primer momento, el nodo A transmite un paquete *RREQ* a sus nodos vecinos: nodos B, C y D. Dichos nodos, reciben el paquete no necesariamente en el mismo instante y, por lo tanto, tampoco todos los nodos retransmiten el paquete coincidentemente en el mismo instante. Cuando B retransmite el paquete a sus vecinos (los nodos C y E), se agrega a sí mismo a la cabecera indicando que forma parte del camino recorrido por el paquete. El paquete que transmitió B hasta C, llega en una era posterior a la del paquete enviado por A hasta C. Como C retransmite únicamente el primer paquete *RREQ* que llega con un número de secuencia dado, el paquete proveniente de B es ignorado y no se lo retransmite.

De esta forma, la Figura 4 (a) muestra que al nodo G llegaron dos paquetes *RREQ*, ambos iniciados por el nodo A. Uno de los paquetes recorrió el camino [A C E F], el otro [A C E]. Una vez que el nodo G recibió un paquete *RREQ*, controla haber realizado previamente el envío del paquete *RREP* correspondiente. Si este no fuera el caso para el nodo remitente y el número de secuencia indicados, el nodo destinatario G se agrega al camino indicado en la cabecera y da por concluida la fase de *Route Request*. La culminación de la fase *Route Request* da inicio a la siguiente: *Route Replay*.

En la fase *Route Replay*, el nodo al cual se quiere transmitir datos crea un paquete de tipo *RREP*, conteniendo la información calculada según se explica en el párrafo anterior. En la cabecera del paquete se indica la ruta a ser utilizada para llegar hasta el nodo que inició el

proceso *Route Discovery*. Dicha ruta de regreso se obtiene invirtiendo el camino utilizado por el paquete *RREQ*.

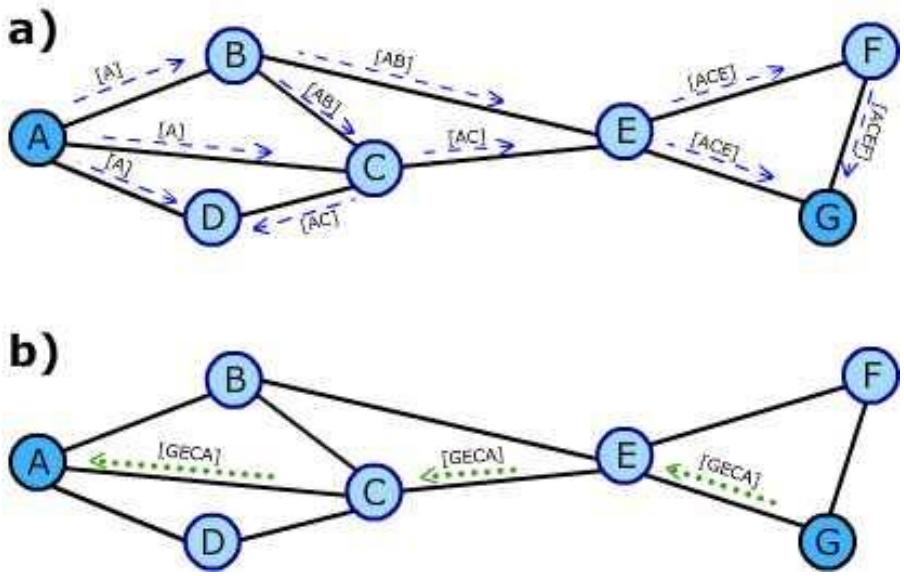


Figura 4. Mecanismo Route Discovery. a) RREQ. Los paquetes *RREQ* son enviados en busca de una ruta entre los nodos A y G. **b) RREP.** El nodo G crea un paquete *RREP* el cual es encaminado hasta el nodo A utilizando la ruta inversa a la calculada en (a).

La Figura 4 (b) indica con flechas compuestas por puntos verdes, a cada paquete *Route Replay*. El nodo G envía dicho paquete utilizando el camino inverso al recorrido por el primer paquete *RREQ* recibido, es decir [G E C A]. De esta forma concluye el proceso de *Route Discovery*, y el nodo A posee información suficiente para definir el camino a ser utilizado por los paquetes con destino al nodo G. El lapso de tiempo durante el cual se reutilizará la información calculada depende de la implementación. El proceso debe ser reiniciado posteriormente para incorporar los cambios en la topología de la red.

El algoritmo *Dynamic Source Routing*, a diferencia del *Destination-Sequenced Distance-Vector*, no envía paquetes de control en forma periódica para actualizar las rutas. El algoritmo *DSDV* utiliza recursos (e.g. ancho de banda y tiempo de cómputo) en el proceso de intercambiar las tablas entre nodos vecinos, aun cuando durante un período de tiempo no exista tráfico de datos con necesidad de ser encaminado a través de la red dinámica periódica. El algoritmo *DSR* no incurre en dicho malgasto de recursos, mientras no existan paquetes a ser transmitidos. Esto sucede por su condición de algoritmo reactivo.

En la siguiente situación se hace notoria una desventaja del algoritmo *DSR*. Estando definido un camino entre los nodos remitente y destinatario, un paquete de datos comienza su camino a través de la red. Supóngase que mientras el paquete se encuentra viajando, un nodo intermedio –integrante del camino- se desconecta de la red en forma súbita. Si el paquete de datos todavía no hubo visitado al nodo ahora desconectado, dicho paquete está destinado a no arribar al nodo destinatario. Lo anterior sucede inclusive cuando los nodos intermedios tienen conocimiento acerca del inconveniente. En contrapartida, en algoritmos como *DSDV* los nodos intermedios pueden corregir la situación al tomar decisiones en cada salto acerca del camino que siguen los paquetes.

2.1.3. **Ad-hoc On-Demand Distance Vector**

El algoritmo *Ad-hoc On-Demand Distance Vector* (AODV) [26], se basa en el algoritmo *DSDV* descrito anteriormente (ver sección 2.1.1.), aunque se diferencia del mismo en que *AODV* es reactivo y no utiliza un esquema de intercambio de tablas entre vecinos. Con *AODV*, al momento de definir la ruta que utilizará un paquete para llegar a destino, el nodo remitente utiliza el esquema *Route Discovery,* de igual forma a la propuesta en el algoritmo *Dynamic Source Routing* descrito en la sección anterior. A diferencia del algoritmo *DSR*, el algoritmo *AODV* –en un intento por no inundar la red con paquetes de control- inicia el proceso de descubrimiento de rutas dando a los paquetes de tipo *Route Request* un tiempo de vida corto. Dicho tiempo de vida es incrementado mientras no se reciba una respuesta satisfactoria al descubrimiento de rutas [15, 26]. El tiempo de vida representa generalmente la cantidad de saltos que un paquete puede dar antes de ser eliminado. Un nodo, previamente al envío de un paquete a un nodo vecino, modifica el campo tiempo de vida disminuyendo su valor en uno. Estrictamente, el valor máximo de tiempo de vida para un paquete está dado por la memoria disponible para almacenar al mismo en la cabecera; sin embargo, asignar valores mayores a la cantidad de nodos existentes en la red en vez de exactamente dicha cantidad, no redunda en beneficios y en cambio, implica un mayor costo de recursos [18].

Los nodos que se encuentran conectados a la red, mantienen una tabla de encaminamiento parecida a la implementada en el algoritmo *Destination-Sequenced Distance-Vector*. Dicha tabla es actualizada haciendo uso de la información calculada con el mecanismo *Route Discovery*. En el algoritmo *AODV*, los nodos envían a sus vecinos, en forma periódica, un paquete de control denominado *HELLO*. Si un nodo no recibe ningún paquete del tipo *HELLO* proveniente de un nodo vecino durante un período de tiempo dado, asume su ausencia de la red

y procede a eliminar las rutas en las que dicho nodo participa, de la tabla de encaminamiento. De esta forma, los cambios en la topología de la red pueden ser detectados antes de generarse un error durante el recorrido de un paquete, causado por la ausencia de un nodo.

2.2. Algoritmos Mono-Objetivo Orientados a EG

En un algoritmo mono-objetivo se define una función objetivo la cual se intenta optimizar. Esto es, minimizarla o maximizarla. Por el otro lado, en un algoritmo multi-objetivo se definen más de una función objetivo, las cuales se intenta optimizar en forma simultánea, bajo ciertas restricciones. Dichas funciones objetivo pueden estar en conflicto; es decir, la mejoría de un factor va en detrimento de otro. Un algoritmo multi-objetivo calcula un conjunto de soluciones no dominadas.

Para el caso de los algoritmos mono-objetivo, donde existe una única función a ser optimizada, el espacio de búsqueda se encuentra ordenado según el valor que resulte de aplicar la función a cada posible entrada. Dicho orden permite encontrar un *óptimo global*, ya sea éste un mínimo o un máximo global [18, 27]. Esto es, para una función objetivo $f : \Re^n \rightarrow \Re$, y para todos los valores de entrada $x \in \Re^n$, se considera que $x^* \in \Re^n$ es un óptimo global sí y solo sí:

$$\forall x \in \Re^n : f(x^*) \text{ es mejor o a lo sumo igual que } f(x) \qquad (2.1)$$

Existen casos especiales donde, si bien se está interesado en optimizar diferentes objetivos en vez de uno solo, dichos objetivos son mesurables en una misma unidad y pueden valorarse unos en forma relativa a los otros. Bajo las circunstancias citadas, se puede reformular un problema ponderando la importancia de cada objetivo con un valor real. De esta forma, el problema multi-objetivo se vuelve mono-objetivo, ya que se logró converger a un objetivo único [27].

En el presente trabajo se tratan las redes dinámicas periódicas. Un grafo puede representar la topología de una red en un instante dado considerando a cada nodo de la red como un nodo del grafo, y a cada uno de los enlaces entre nodos de la red, como arcos entre los nodos del grafo. Dividiendo el tiempo en instantes, puede construirse un grafo por cada instante, de forma a representar la topología de la red en cada instante de tiempo. Si en un conjunto de instantes consecutivos, la topología de la red no sufre variaciones, los grafos que representan a dichos instantes son iguales, y pueden ser agrupados en un conjunto. Un Grafo Evolutivo, en ingles

Evolving Graph (EG), está compuesto por el conjunto de grafos que representan, cada uno, a un instante de tiempo o a un conjunto de instantes consecutivos en los cuales la topología de la red permanece constante [4, 18, 19].

Una era es un conjunto de instantes o un lapso de tiempo durante el cual, la topología de la red conformada por los nodos y enlaces activos, permanece constante. Una red dinámica periódica puede dividirse en eras, donde cada una de ellas no necesariamente dura la misma cantidad de tiempo, pero en todas, la topología se mantiene constante. En un grafo evolutivo puede suceder que existan dos lapsos de tiempo distintos sin solaparse, los cuales duran la misma cantidad de tiempo, y cuyas topologías de red es exactamente la misma. No obstante las características de los lapsos de tiempo antes citados, no es posible considerar a ambos lapsos como una misma era si es que entre ellos existe al menos un instante donde la topología de red fue distinta [3, 14, 19].

Utilizando la definición anterior de era, se puede definir un grafo evolutivo como un formalismo que permite simplificar la representación de redes dinámicas periódicas, combinando en un solo grafo lo que se puede entender como el conjunto de todos los grafos que corresponden a cada una de las eras o instantes diferentes que tiene la red [4, 18, 19].

Los grafos evolutivos incorporan la noción de tiempo ya sea discreto o continuo a través de las eras. Para los algoritmos publicados hasta el presente trabajo, la complejidad de los cálculos en el caso de utilizar el tiempo como una dimensión continua es tan elevada, que hace inviable la ejecución exitosa de dichos algoritmos [9]. Basado en la experiencia adquirida, el presente trabajo utiliza el concepto de grafos evolutivos incorporando la noción de eras, donde cada era representa un periodo de tiempo, de forma que el tiempo sea discreto. Por claridad, en lo que resta del texto se asume que las eras en las que se divide una red dinámica periódica, duran igual cantidad de tiempo (i.e. son homogéneas en tiempo). No obstante, los cálculos y observaciones son igualmente válidos para el caso de eras con lapsos de tiempo heterogéneo.

Diversos algoritmos de ruteo se basan en el algoritmo de Dijkstra para Caminos Mínimos (ver Anexo I). Si se representa un camino mínimo por la sucesión ordenada de nodos que lo conforman, se define que un prefijo del camino mínimo es cualquier sección de dicho camino mínimo. En el algoritmo de Dijkstra para Caminos Mínimos, un prefijo de camino mínimo es también mínimo. El siguiente es un ejemplo de lo mencionado anteriormente: sea un camino mínimo el conformado por nodos [A B C D E] para llegar desde el nodo A hasta el nodo E. Por

lo tanto, el camino compuesto por el sufijo [B C D E] es necesariamente mínimo para viajar desde el nodo B hasta el nodo E.

Trasladando el ejemplo anterior al contexto de las redes dinámicas periódicas, supóngase la existencia de un enlace activo entre los nodos A y B en la era 2, entre los nodos B y C en la era 3 y entre B y E en la era 1. Al calcularse el camino mínimo entre A y E, el primer salto debió esperar hasta la era 2 en la cual existe un enlace al nodo B, y luego esperar hasta la 3 para utilizar el enlace al nodo C siguiendo así el camino. Si el nodo inicial fuera B en vez de A, entonces se hubiera podido hacer uso del enlace entre B y E en la era 1, habiéndose encontrado el camino mínimo [B E]. Esto demuestra que, en el ejemplo, [A B C D E] es un camino mínimo a diferencia del sufijo [B C D E], que no lo es.

En las redes dinámicas periódicas, la persistencia de los caminos mínimos no puede afirmarse unívocamente; esto es, un camino mínimo que inicia en una era *e*, no necesariamente es mínimo iniciando en una era *e'*. En general, dado un camino mínimo, no necesariamente un prefijo de dicho camino es también mínimo. Un viaje o camino está compuesto por pares ordenados <*n*, *e*>, donde *n* es un nodo y *e* una era en la que el nodo se encuentra conectado a al red. Dado un camino, si el paquete se encuentra almacenado en el nodo *n',* y el siguiente par es <*n*, *e*>, debe existir un enlace activo que conecte a los nodos *n* y *n'* en la era *e*.

La Figura 5 presenta a modo de ejemplo, un grafo evolutivo que define la topología de una red dinámica periódica. En dicha figura, los enlaces se encuentran activos en las eras listadas entre corchetes, encima de la línea que expresa al enlace. Los identificadores de los cuatro nodos integrantes de la topología son A, B, C y D. Entre los nodos A y B existe un enlace en las eras 1, 2 y 4; asimismo, entre los nodos C y D existe un enlace en las eras 2 y 3 y así sucesivamente.

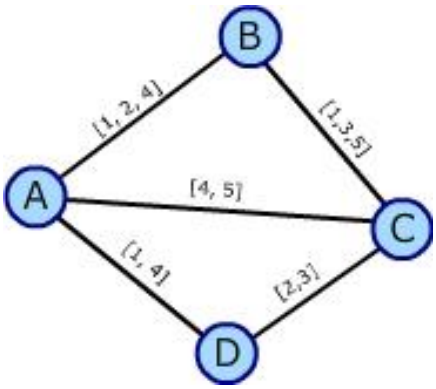


Figura 5. Ejemplo de representación de un grafo evolutivo.

En lo que sigue de la sección, se presentan dos algoritmos propuestos por Monteiro [18]. Dichos algoritmos hacen uso de grafos evolutivos para representar a los nodos, los enlaces y los cambios que se dan en la topología de las redes dinámicas periódicas. Estos algoritmos son denominados *Evolving Graph Foremost* (EG Foremost) y *Evolving Graph Shortest* (EG Shortest), los mismos son mono-objetivo y difieren en la función objetivo que cada uno intenta optimizar, siendo que el primero se aboca a la era de llegada de los paquetes, y el segundo a la cantidad de saltos realizados hasta llegar al destinatario.

2.2.1. **Evolving Graph Foremost**

El algoritmo *Evolving Graph Foremost* (EG Foremost) tiene un funcionamiento proactivo. Su función objetivo es la minimización de la era de llegada de los paquetes. La información es extraída del grafo evolutivo que representa a la red dinámica periódica, y con ella, calcula los caminos óptimos para cada par de nodos *{remitente; destinatario}* en cada una de las eras definidas en el EG. El algoritmo *EG Foremost* puede indicar como mejor camino uno que, a primera vista, no parece eficiente. Como ejemplo, supóngase el grafo evolutivo que se muestra en la Figura 6. Existen diferentes caminos para enviar un paquete desde el nodo A hasta F. A primera vista podría parecer conveniente utilizar el camino [A E F] a través del cual el paquete llegaría luego de 2 saltos en la era 4. No obstante, el algoritmo *EG Foremost* logra encontrar el camino [A B C D E F], el cual permite al paquete llegar hasta destino en 5 saltos pero en la era 1, siendo esa la era más temprana posible de llegada.

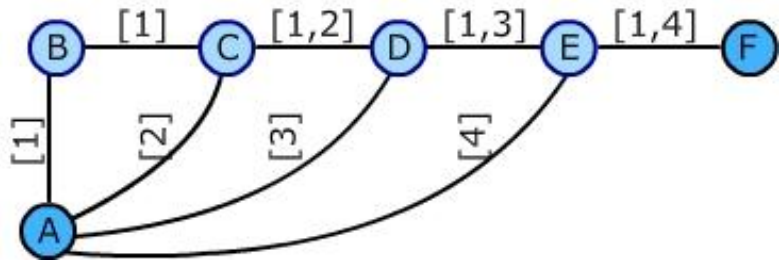


Figura 6. **Caminos en una red dinámica periódica**. Con el camino [A B C D E F] un paquete puede llegar en la era 1 con 5 saltos. Con [A E F] el mismo paquete puede llegar en la era 4 con 2 saltos.

Evolving Graph Foremost, es una adaptación hecha por Monteiro [18] del algoritmo de Dijkstra para Caminos de Costo Mínimo (ver Anexo I). En el algoritmo de Dijkstra, cualquier prefijo de un camino de costo mínimo, es también de costo mínimo. Como ya fuera mencionado, esta propiedad no se cumple en un grafo evolutivo ya que se depende de que las aristas se encuentren activas en una u otra era, como ya se explicó en párrafos anteriores.

La Figura 7 plantea, a modo de ejemplo, lo que ocurre en el viaje realizado por tres paquetes denominados *paquete 1*, *paquete 2* y *paquete 3* cuando los mismos utilizan caminos diferentes unos de otros para viajar del mismo nodo remitente al mismo nodo destinatario. Las líneas verticales indica el comienzo de cada era, siendo su número el indicado en el recuadro superior de cada línea. Los paquetes son identificados por una figura alargada, siendo su extensión correspondiente a la cantidad de eras transcurridas durante el viaje. Dichas figuras comienzan en la región de la era en que partieron del nodo remitente, y se extienden hasta la región de la era en la cual llegaron al nodo destino, culminando así su viaje. Debajo de la figura que representa al paquete, se encuentra en otra más pequeña la cantidad de saltos que realizó el paquete hasta llegar al destino (i.e. la cantidad de nodos que visitó en el camino). Los tres paquetes se encuentran disponibles para ser enviados desde la primera era y el inicio del viaje depende exclusivamente de la definición del recorrido para cada uno de ellos.

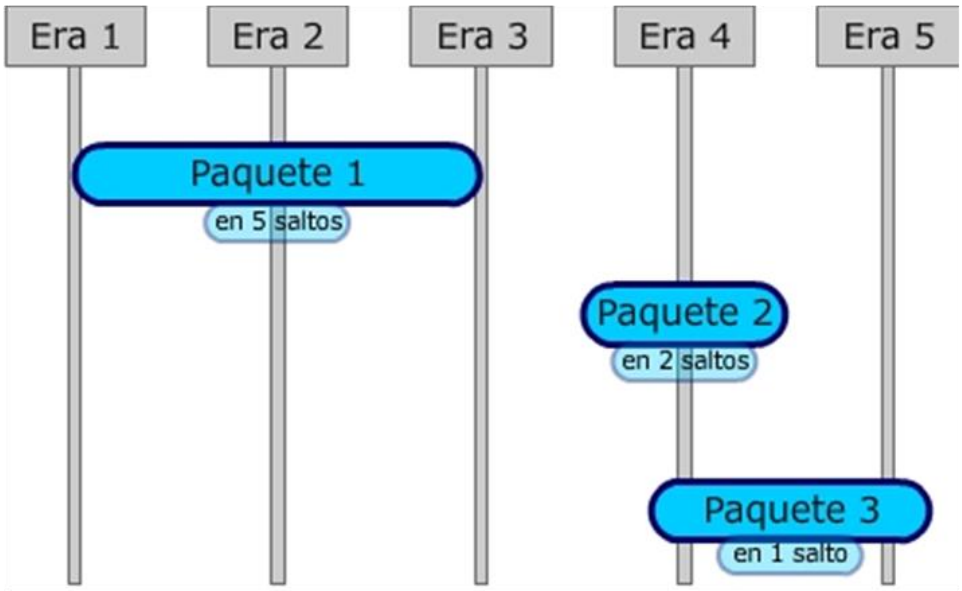


Figura 7. Ejemplo de tres paquetes que viajan del mismo nodo remitente al mismo nodo destinatario, pero utilizando tres caminos distintos.

Para el caso del *paquete 1*, el mismo comienza su viaje en la era 1, atraviesa el camino en 2 eras y llega a destino en la era 3. El *paquete 2* empieza el viaje en la era 4 y llega en la misma era 4 a destino habiendo necesitado solo 2 saltos. Como inició y culminó su recorrido en la misma era, sin que haya habido una transición a una era posterior, la cantidad de eras viajadas es 0. Por último, el *paquete 3* comienza su camino partiendo del nodo remitente en la era 4, y realiza 1 salto para llegar al nodo destino en la era 5; el *paquete 3* tuvo un viaje que duró 1 era.

El *paquete 1* culminó su viaje en una era anterior a las eras en que el *paquete 2* y el *paquete 3* terminaron de recorrer el suyo. El algoritmo *EG Shortest* opta por asignar a cada paquete el camino que logre minimizar la era de llegada del mismo, es decir, que llegue lo antes posible. Como se puede apreciar en la Figura 7, y suponiendo que los únicos tres caminos posibles son los tomados por cada uno de los paquetes, el algoritmo *EG Foremost* encamina a los paquetes utilizando el camino recorrido por el *paquete 1*. Esto es, porque la era 3 es la más temprana posible de llegada hasta el destino.

A continuación se presenta el pseudocódigo del algoritmo *Evolving Graph Foremost*, implementado por Monteiro en [18] (ver Algoritmo 3).

```
/*

Variables:

N: conjunto de nodos existentes en el grafo evolutivo
n_r: nodo remitente
e: era en que está el sistema
Q: lista que contiene a los nodos no cerrados
d: una tabla indexada por los nodos, donde d(n) almacena el costo de
llegar desde el nodo n_r hasta n

Se considera que un nodo está cerrado si ya fue calculado el costo
mínimo de llegar hasta él desde n_r

*/

   1. Inicializar d(n_r) ← e, y ∀n∈N n≠n_r, d(n) ← ∞
   2. Inicializar Q conteniendo únicamente al nodo n_r en el tope
   3. Mientras Q≠∅ hacer:
      a. n ← tope de Q
      b. Quitar n de Q
      c. Para cada vecino abierto n' de n hacer:
         • Calcular la primera arista con era mayor o igual a e
         • Si n' no está en Q, se lo inserta en la cola
         • Actualizar d(n') si es necesario
      d. Fin
      e. Para cada n en Q, ordenar Q de menor a mayor según d(n)
      f. Cerrar n
   4. Fin
```

Algoritmo 3. Pseudocódigo del algoritmo *EG Foremost*.

En el Algoritmo 3, definiendo un nodo remitente para el cuál se quiere encontrar el camino a los otros nodos, se cierran uno a uno todos los nodos existentes en el grafo evolutivo. Cerrar un nodo se define como calcular la distancia mínima hasta él, desde el nodo que fue seleccionado como remitente. El algoritmo inicializa estableciendo como infinita la distancia hasta cada nodo, menos al remitente, indicado como $\boldsymbol{n_r}$. En forma ordenada según la distancia encontrada en el proceso, se evalúa a cada nodo $\boldsymbol{n} \in \boldsymbol{N}$ como intermediario último entre $\boldsymbol{n_r}$ y los vecinos de $\boldsymbol{n}$. Se compara la distancia que existiría si se utiliza el camino a través de $\boldsymbol{n}$ en vez del ya almacenado y calculado anteriormente. En caso que el nuevo camino, en el cuál $\boldsymbol{n}$ es el último nodo intermedio, sea de una distancia menor al camino almacenado, prevalece éste nuevo camino encontrado en vez del almacenado anteriormente.

2.2.2. **Evolving Graph Shortest**

El algoritmo *Evolving Graph Shortest* (EG Shortest) es catalogado como proactivo, ya que el cálculo de los caminos se realiza antes de necesitar en forma inminente la información de ruteo. Define una función objetivo que minimiza la cantidad de saltos realizados por los paquetes para viajar desde el nodo remitente hasta el nodo destinatario. Basándose en la información del grafo evolutivo, calcula los caminos óptimos para cada par de nodos *{remitente; destinatario}* en cada una de las eras.

Al igual que el algoritmo *Evolving Graph Foremost*, este algoritmo es una adaptación realizada por Monteiro [18] del de Dijkstra para Caminos de Costo Mínimo (ver Anexo I), con la diferencia de que en vez de priorizar la era de llegada de los paquetes, intenta minimizar la cantidad de saltos ejecutados entre nodos a través de los enlaces.

El algoritmo *EG Shortest* puede identificar como mejor opción el almacenar un paquete en memoria por cierta cantidad definida de eras. En la Figura 6 se presentó un ejemplo de red dinámica periódica, mostrando el grafo evolutivo que la define. En dicha figura, utilizando el camino [A B C D E F] un paquete puede llegar en la era 1 con 5 saltos, mientras con el camino [A E F] el mismo paquete puede llegar en la era 4 con 2 saltos. El algoritmo *EG Shortest*, en este caso, optaría por utilizar el camino [A E F] aun cuando para ello el paquete deba ser almacenado, aguardando a la era 4 para iniciar el viaje.

El ejemplo brindado por la Figura 7 muestra a tres supuestos paquetes. Todos ellos se encuentran disponibles para ser enviados desde la era 1, los tres tienen al mismo nodo como remitente y al mismo nodo destinatario. El *paquete 1* realizó un camino de 5 saltos para arribar

al destino, mientras que el *paquete 2* necesitó 2 saltos y el *paquete 3* necesitó únicamente 1 salto. El algoritmo *Evolving Graph Shortest* opta por el camino que realizó el *paquete 3* ya que necesitó menor cantidad de saltos que los otros dos paquetes, aunque haya llegado en una era muy posterior al *paquete 1*.

El Algoritmo 4 muestra el pseudocódigo de *Evolving Graph Shortest* incluyendo la modificación al algoritmo de Dijkstra para caminos mínimos realizada por Monteiro en [18].

```
/*

Variables:

N: conjunto de nodos existentes en el grafo evolutivo
n_r: nodo remitente
e: era en que está el sistema
T: árbol de caminos más cortos con elementos <n', e'>
V_G: conjunto de enlaces existentes en el grafo evolutivo
location: tabla indexada por los nodos n∈N
location(n): puntero a T que indica el nodo anterior en el camino a n
earliest: tabla indexada por los nodos n∈N
earliest(n): contiene información sobre la menor era de llegada a n
d: contador de profundidad en el árbol T

*/

   1. Inicializar T ← <n_r, e_now>
   2. Inicializar earliest(n_r) ← e_now, earliest(n) ← ∞  ∀n∈V_G n≠n_r
   3. d ← 1
   4. location(n_r) ← <n_r, e_now>
   5. Mientras exista n ∈ V_G tal que location(n) = ⊥, hacer:
      a. Para los pares <n, e> en el árbol T a una profundidad d, hacer:
         • Seleccionar los pares <n', e'> vecinos de <n, e>
         • Si location(n') = ⊥, entonces location(n') ← <n', e'>
         • Si earliest(n') > e'
            o earliest(n') ← e' y <n', e'> es hijo del nodo <n, e> de T
         • Fin
      b. Fin
      c. d ← d + 1
   6. Fin
   7. Retornar el árbol T y el vector location
```

Algoritmo 4. Pseudocódigo del algoritmo *EG Shortest*.

En el Algoritmo 4, el conjunto $\boldsymbol{V_G}$ está formado por las aristas o enlaces existentes en el grafo evolutivo. Luego de elegir un nodo como remitente y denominarlo $\boldsymbol{n_r}$, se calculan los caminos de costo mínimo para llegar desde $\boldsymbol{n_r}$ hasta cualquier otro nodo que forma parte del grafo evolutivo, y que es un extremo de algún enlace del conjunto $\boldsymbol{V_G}$. El árbol $\boldsymbol{T}$ organiza a los nodos de manera tal, que los caminos que se forman entre la raíz y las hojas, son los caminos óptimos buscados. Cada hoja de dicho árbol está formada por un nodo y una era, representados por el par ordenado <$\boldsymbol{n}$, $\boldsymbol{e}$>. Se recorre el árbol por niveles, utilizando para tal efecto al contador $\boldsymbol{d}$. Para cada nodo del árbol en el nivel $\boldsymbol{d}$, se toman los nodos adyacentes en la era en que el enlace entre ellos se encuentra activo, formando así el par ordenado <$\boldsymbol{n'}$, $\boldsymbol{e'}$>.

El árbol $\boldsymbol{T}$ contiene la descripción del camino óptimo (i.e. el de menor cantidad de saltos) desde el nodo remitente $\boldsymbol{n_r}$ en la era $\boldsymbol{e_{now}}$ hasta el nodo destinatario $\boldsymbol{n}$. El árbol debe interpretarse comenzando desde el nodo apuntado por la tabla ***location(n)***, atravesando cada nodo padre intermedio, hasta llegar a la raíz. Cabe destacarse que existe más de un nodo en el árbol para cada nodo $\boldsymbol{n}$, ya que no es indiferente la era en la que se encuentra la red dinámica, ya que de ello depende la conectividad de los nodos.

3. Problema de Optimización Multi-objetivo

Los *Problemas de Optimización Multi-objetivo* (MOP) se dan en la vida diaria y no son exclusivos del área de ingeniería. Cotidianamente, las personas lidian con ellos en forma natural posiblemente sin ser plenamente consientes de ello. Algunas veces, las decisiones son apenas consientes y pasan inadvertidas; en otras ocasiones, es necesario recabar información para realizar un razonamiento más elaborado y tomar una decisión.

Un ejemplo de planteamiento de un problema como *MOP*, es cuando una persona se moviliza en un automóvil propio, y por ende tiene dominio exacto sobre el camino que realiza, debe conducir desde un punto de una ciudad, hasta otro punto de la ciudad. En esta situación, posiblemente los objetivos a optimizar sean: la cantidad de combustible necesitado, el tiempo de viaje, la cantidad de kilómetros de recorrido e inclusive los beneficios de pasar por otros puntos intermedios específicos (e.g. realizar una compra en un negocio en particular). El conductor planifica el camino que realizará intentando minimizar cada uno de los costos (i.e. tiempo, costo y distancia del viaje) además de maximizar los beneficios de puntos específicos en el camino. Para ambos ejemplos propuestos, las posibles soluciones deben respetar ciertas restricciones como son las leyes de la física, las capacidades de la persona, entre otras. En los dos casos, existen objetivos que se encuentran en conflicto.

En el área de ingeniería, un problema puede ser modelado como un *Problema de Optimización Multi-objetivo (MOP)* si se definen las funciones que deben ser minimizadas o maximizadas. Dichas funciones probablemente se encuentran en conflicto, además de posiblemente utilizar unidades de medida diferentes. Deben observarse las restricciones propias del problema.

En el paradigma multi-objetivo se busca una solución representada por un vector x de dimensión n, que optimice una función vectorial y = F(x) compuesta por k objetivos, es decir que $F(x) : \Re^n \rightarrow \Re^k$. Dicha solución está sujeta a las restricciones G(x). Formalmente, lo anterior puede definirse de la siguiente manera [27]:

$$y = F(x) = [f_1(x), \ldots, f_k(x)] \tag{3.1}$$

$$G(x) = [g_1(x), \ldots, g_m(x)] \geq 0 \tag{3.2}$$

donde el vector de decisión *x* es de dimensión *n*, y el vector resultante de las funciones objetivo es de dimensión *k* con valores reales como elementos.

$$x = [x_1, \ldots, x_n] \in X \subseteq \Re^n \tag{3.3}$$

$$y = F(x) = [y_1, \ldots, y_k] \in Y \subseteq \Re^k \tag{3.4}$$

La fórmula (3.3) muestra una solución compuesta por valores reales. No obstante, los cálculos son similares si en vez de números reales otros dominios son utilizados. Para ello, las funciones objetivo deben conocer a dichos dominios, de forma tal a utilizar en forma consistente sus valores. Si este fuera el caso, cada dimensión del vector x podría tener un dominio específico, pudiendo ser éste el conjunto de caracteres ASCII, el conjunto de las cadenas de texto, direcciones de memoria, entre otros.

Al modelar un problema como un Problema de Optimización Multi-objetivo, se deben identificar las funciones objetivo (3.1) y evaluar cada una de las soluciones dadas por (3.3) que se ajustan al conjunto de restricciones (3.2) del problema. De esta forma cada solución tiene asociado un vector objetivo *y*, dado por (3.4).

La Dominancia Pareto es utilizada para comparar dos soluciones. Para ello, deben evaluarse cada uno de sus respectivos objetivos. Una solución domina a otra en el sentido Pareto sí y solo sí todos sus elementos son mejores o iguales a los elementos respectivos de la otra, y al menos uno es estrictamente mejor [27]. Para el caso de las soluciones *x'* y *x''* donde lo que se busca es una optimización por maximización, la solución *x'* domina a *x''* si y solo si:

$$\underset{0 < i \leq k}{\forall i}\, f_i(x') \succcurlyeq f_i(x'') \quad \text{y} \quad \underset{0 < j \leq k}{\exists j}\, f_j(x') \succ f_j(x'') \tag{3.5}$$

Haciendo uso de la Dominancia Pareto, el Conjunto Pareto se define como el conjunto de todas las soluciones factibles no dominadas por ninguna otra solución. A su vez, considerando la reciente definición, la imagen del Conjunto Pareto en el espacio objetivo se conoce como Frente Pareto [16, 27].

Problema de ruteo definido como MOP

Las ventajas de utilizar grafos evolutivos para representar redes dinámicas periódicas, fueron presentadas en [3, 14, 18, 19]. También se hizo una descripción de algunos algoritmos conocidos, que inicialmente se utilizaron para realizar el ruteo de paquetes en redes dinámicas. Ellos son: *Destination-Sequenced Distance-Vector* (DSDV) [20], *Dynamic Source Routing* (DSR) [12] y *Ad-hoc On-Demand Distance Vector* (AODV) [26] (ver sección 2.1.).

En [18], Monteiro demostró que dos algoritmos mono-objetivo: *Evolving Graph Foremost* y *Evolving Graph Shortest*, son capaces de encontrar una solución óptima para el contexto mono-objetivo, cada uno en su función objetivo. Estas soluciones óptimas son igualmente buenas o inclusive mejores que aquellas soluciones encontradas por los algoritmos tradicionales mencionados anteriormente. El algoritmo *EG Shortest* minimiza la cantidad de saltos mientras que el algoritmo *EG Foremost* encuentra la era de llegada más temprana posible.

Hasta el presente trabajo, todas las publicaciones encaran el problema tratándolo en un contexto mono-objetivo, minimizando una de las siguientes mediciones del viaje, según son definidas en [18]:

a) La suma de saltos. Se considera que el paquete da un salto cada vez que es transmitido de un nodo a otro utilizando un enlace. Se desea minimizar la cantidad de saltos requeridos por los paquetes para viajar desde el nodo remitente hasta el nodo destinatario.

b) La era más temprana de llegada. Cada paquete es creado cuando existe información suficiente para ser transmitida. Es deseable que los paquetes culminen su viaje llegando al nodo destinatario en la era más temprana posible.

c) La cantidad de eras viajadas. Los paquetes se encuentran viajando a través de la red durante un período de eras, desde que partieron de sus respectivos nodos remitente dando el primer salto, hasta que arribaron a los nodos destinatario. En este sentido, la rapidez se relaciona con la cantidad de eras que dura el viaje.

Este trabajo propone considerar los tres objetivos (funciones de costo) en forma simultánea, en un contexto absolutamente multi-objetivo. Presentar el problema de ruteo como un *Problema de Optimización Multi-objetivo*, o por sus siglas en inglés, MOP (*Multiobjective Optimization Problem*), permite hallar un conjunto Pareto de soluciones no dominadas que simultáneamente optimicen las tres funciones objetivo, y donde dichas funciones pueden estar en conflicto [18, 27]. Las soluciones de compromiso son de gran valor para el tomador de decisiones.

De esta forma, un conjunto de soluciones Pareto puede contener soluciones que son tan buenas como las optimas calculadas por los algoritmos *Evolving Graph Foremost* y *Evolving Graph Shortest* cuando se considera un único objetivo. Sin embargo, se puede encontrar en el conjunto Pareto otras soluciones incluso mejores cuando las tres funciones de costo, recientemente señaladas como (a), (b) y (c), son consideradas en forma simultánea, lo cual no es realizado en el contexto mono-objetivo.

En resumen, este trabajo es el primero en abordar el problema de encaminamiento en redes dinámicas periódicas, modelándolo como un *Problema de Optimización Multi-objetivo* (MOP). En la siguiente sección se propone un *Algoritmo Evolutivo Multi-Objetivo*, capaz hallar un conjunto de soluciones Pareto que no sólo contienen individuos competitivos con las soluciones calculadas por otros algoritmos que forman parte del estado del arte, sino que además, se logra calcular individuos que dominan a las soluciones consideradas como óptimas cuando una sola función objetivo es considerada. Esto se debe a que dicho enfoque propuesto es capaz de mejorar, en un contexto multi-objetivo, a las soluciones que pueden ser consideradas como óptimas en un entorno mono-objetivo.

4. Algoritmo Evolutivo Multi-Objetivo

En el año 1859, el naturista inglés Charles Robert Darwin publica la obra "*El origen de las especies*". En ella se plantea la evolución de las especies a través de la selección natural y se la explica como *"un mecanismo evolutivo que se define como la reproducción diferencial de los genotipos en el seno de una población biológica"* (Javier Sampedro, Deconstruyendo a Darwin, página 22).

Gregor Johan Mendel, monje católico y naturista, nacido en Austria, realizó estudios y publicaciones (entre los años 1865 y 1866) definiendo lo que hoy se denomina "*Leyes de Mendel*" y tratan acerca de la herencia genética, las cuales son más bien reglas ya que no se cumplen en todos los casos.

Basándose en los modelos de evolución estudiados y publicados por Charles Robert Darwin y Gregor Johan Mendel Mendel, un Algoritmo Evolutivo o por sus siglas en inglés EA (*Evolutionary Algorithm*), realiza cruzamiento selectivo entre pares de individuos de un grupo (al que se llama población) para hacerlo evolucionar, en un intento de crear mejores individuos en cada generación. Los individuos pueden ser comparados entre ellos aplicándose la Dominancia Pareto (ver fórmula 3.5) como fue definida en la sección 3, de forma tal a asegurarse que las soluciones calculadas son buenas [27]. Un Algoritmo Evolutivo (EA) en el cual se define más de una función objetivo a optimizar, es considerado como un Algoritmo Evolutivo Multi-Objetivo, o por sus siglas en ingles como MOEA (*Multi-Objective Evolutionary Algorithm*).

Se denomina *cromosoma* a una estructura de datos que contiene información genética, la cual en un entorno computacional puede ser almacenada como una cadena de bits, un arreglo de enteros, una matriz o inclusive utilizando otro tipo de estructuras, según sea la complejidad de lo que se necesita modelar. El conjunto de valores que conforma el cromosoma y representa a los genes, es entendido como el *genotipo* (ver fórmula 3.3). La Figura 8 muestra un posible cromosoma donde cada gen tiene un significado particular, ello demuestra la existencia de diversas codificaciones válidas, siendo meramente opcional utilizar una misma codificación para todos los genes. En el ejemplo, los genes 1, 2, 4 y 8 son codificados como números enteros; el gen 3 es un valor booleano que puede adoptar los valores de falso o verdadero. A su vez, el gen 5 está codificado como un byte, el cual es interpretado como un carácter ASCII. El

gen 6 es un apuntador a una dirección de memoria en formato hexadecimal y, por último, el gen 7 es una cadena de texto.

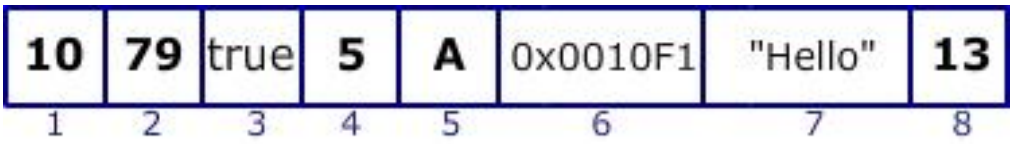


Figura 8. Cromosoma con genes. La información se encuentra codificada en formas diferentes.

Mientras que el genotipo se encuentra en el dominio de valores que puede tomar la solución, el *fenotipo* se encuentra en el espacio objetivo o rango (ver fórmula 3.4) y está compuesto por valores que denotan el desempeño de la solución en cada función objetivo.

El *fitness* es un valor que indica la aptitud de un cromosoma o su calidad respecto a otros cromosomas. El cromosoma con mejores aptitudes de entre un grupo, es el que obtendrá el mayor valor de *fitness*. Los criterios de evaluación son diferentes en cada Algoritmo Evolutivo Multi-Objetivo (MOEA).

Se denomina *individuo* a una estructura que contiene un cromosoma (i.e. un genotipo) y opcionalmente información asociada al mismo, como el valor de *fitness* y otras características propias. Una *población* está compuesta por individuos. Una población es denotada por P, y los individuos integrantes por P_i . La cantidad de individuos que integran una población es $||P||$. El valor de fitness que cada individuo P_i posee, es indicado por $fitness_i$.

Los Algoritmo Genéticos logran hacer evolucionar a una población a través de la aplicación de operadores genéticos. Por lo general, los Algoritmos Genéticos dan una importancia principal a la operación de cruzamiento, mientras que dan una importancia secundaria y de baja probabilidad de ocurrencia a la operación de mutación. La operación de selección es clave en la evolución de una población. A continuación se muestra en forma gráfica la evolución y luego se describen las tres operaciones genéticas.

Sea k la cantidad de veces que el Algoritmo Evolutivo realizó el proceso de hacer evolucionar a una población P. La población P en I_0 es la población inicial; de igual forma, la población P en I_k es la población luego de haberse ejecutado la evolución de la misma k veces. La Figura 9 muestra que una población evoluciona al aplicársele los operadores genéticos.

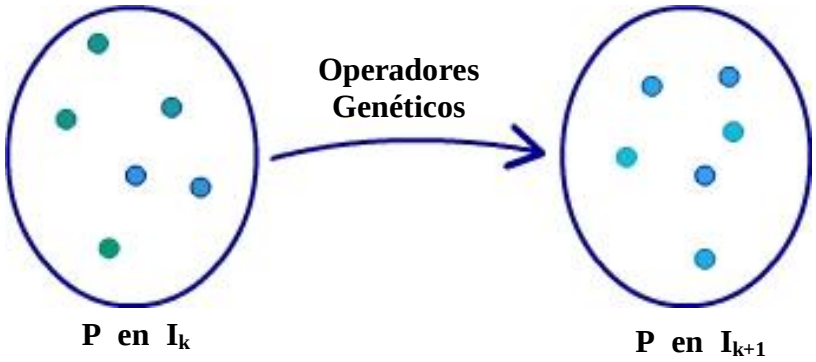


Figura 9. La población *P* luego de evolucionar I_k veces, se convierte en la población *P'* al evolucionar una vez más.

La operación de selección se realiza principalmente a través de dos estrategias: utilizando una ruleta (*roulette-whell selection*) o por torneo (t*ournament selection*) [27]. Para el primer caso, luego de calcular el valor de adaptabilidad *fitness* de cada individuo, se simula una ruleta donde se asigna una celda o casilla a cada individuo. Dicha casilla será para cada individuo de un ancho proporcional al valor de su *fitness*. De esta forma, la probabilidad de que al girar la ruleta sea elegida una casilla en particular, depende de la calidad del individuo (i.e. su *fitness*). La probabilidad de elegir a un individuo P_i en particular a través del método de ruleta es:

$$probabilidad_i \quad \frac{fitness_i}{\sum_{j=1}^{||P||} fitness_j} \tag{4.1}$$

El método de selección denominado *por torneo* (t*ournament selection*), consta de dos fases. En la primera, se eligen al azar una cantidad definida de individuos pertenecientes a la población *P*; en la segunda fase se hace competir a los individuos seleccionados en forma análoga a una competencia de deportes. Al finalizar este método, el individuo que ganó el torneo (i.e. el que tuvo mejor desempeño) será elegido.

El operador genético de cruzamiento requiere haber previamente seleccionado a los dos individuos que se desempeñarán como padres. Supóngase que inicialmente fueron seleccionados los individuos P_i y P_j de la población *P*, utilizando alguna estrategia de selección. El operador de cruzamiento utiliza la información genética de ambos padres pertenecientes a la generación I_k para construir o crear un nuevo individuo, el cual pasará a formar parte de la siguiente generación I_{k+1}.

Existen diversas formas de efectuar la operación de cruzamiento, y la más comúnmente implementada [27] es aquella en la que para dos cadenas de información, se toma como límite un punto en la longitud de la cadena. El individuo nuevo es conformado copiando la información genética que se encuentra en la cadena de un padre, hasta el punto límite (o punto de cruce) que fue elegido previamente. Cruzando la frontera establecida por dicho punto, se copia la información de la cadena del segundo padre a la cadena del individuo nuevo (ver Figura 10).

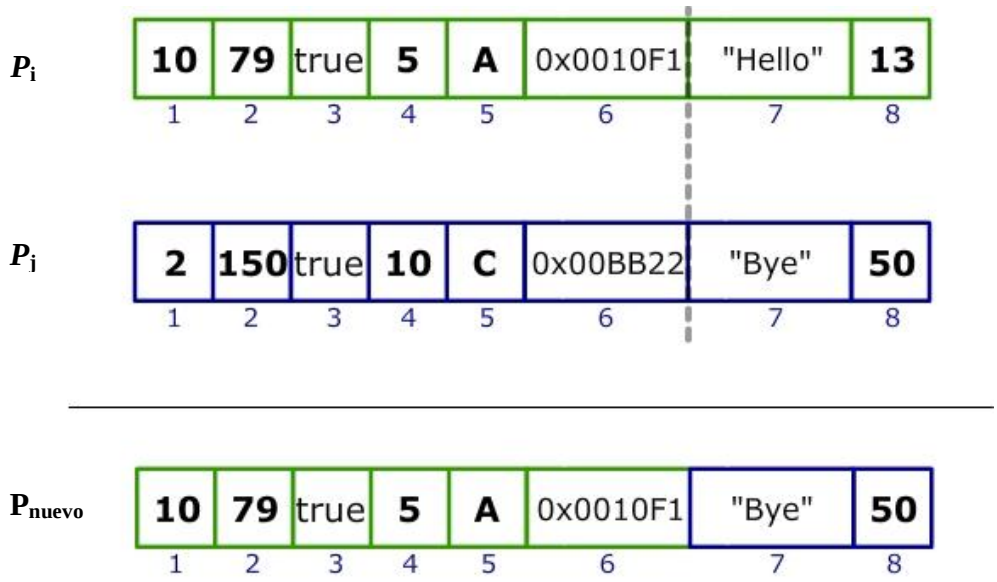


Figura 10. Ejemplo de efectuar la operación genética de cruzamiento.

La mutación es utilizada en los Algoritmos Genéticos como un operador cuyo propósito es el de explorar, en forma aleatoria, otras regiones del espacio de búsqueda. Esto es necesario ya que la operación de cruzamiento transfiere únicamente información genética ya existente en los padres sin introducir nuevas variaciones. En el caso de cadenas binarias, el operador de mutación invierte el valor del bit en la posición indicada dentro de la cadena. En la Figura 11 se muestra la aplicación del operador de mutación a un individuo cuyo gen no contiene información binaria. En el caso referido, el nuevo valor que contendrá un gen es tomado en forma aleatoria del dominio válido para el mismo (mostrado en la Figura 10 encima de cada uno).

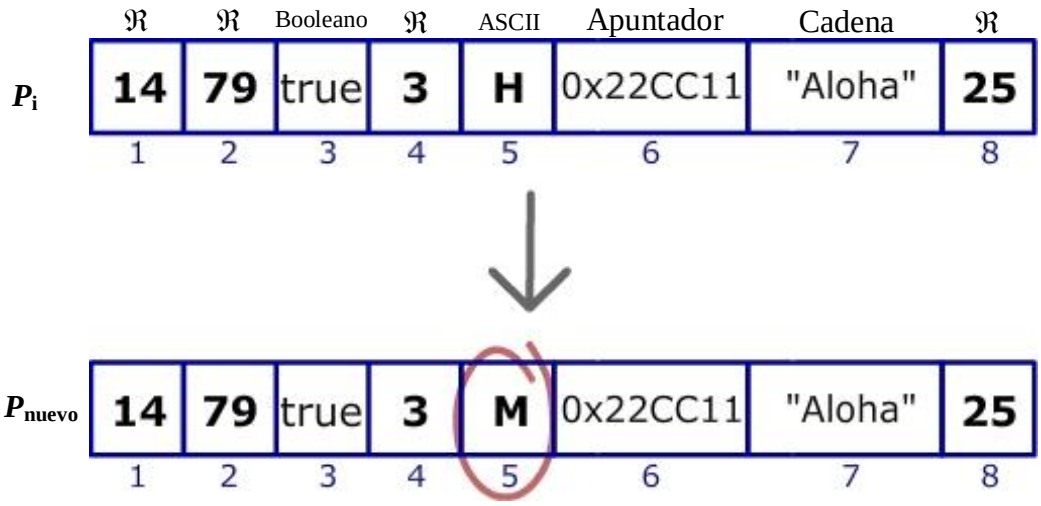


Figura 11. Ejemplo de efectuar la operación genética de mutación.

Los Algoritmos Evolutivos efectúan operaciones genéticas en forma iterativa. Un buen individuo creado durante alguna de las generaciones, puede ser modificado en una generación posterior, causando una pérdida de calidad. Algunos Algoritmos Evolutivos implementan el mecanismo de *elitismo* para asegurar que los cromosomas o individuos más aptos de una población P en I_k no sean alterados en su paso a la generación P en I_{k+1} (i.e. no sean afectados por los operadores genéticos) [27]. Una manera de implementar el mecanismo de elitismo, es mantener una población externa P' a la que se copian los individuos considerados como la élite de P, en caso no existan previamente en P' individuos de mayor calidad que dichos nuevos individuos. Luego de la inclusión de un nuevo individuo a P', es necesario quitar de dicha población a los individuos que son de menor calidad que el nuevo individuo incorporado.

A continuación se presenta un Algoritmo Genético simple, cuyo diagrama de flujo se muestra en la Figura 12. Dicho algoritmo no implementa el mecanismo de elitismo. En un comienzo, se inicializa la población P creando los individuos P_i donde $0 < i \leq ||P||$. Esta primera población contiene individuos creados en forma aleatoria, tomando cada uno de sus genes un valor cualquiera de su dominio respectivo. Una vez inicializada la población inicial P con la información requerida, se ingresa al ciclo a ser ejecutado mientras la condición de parada así lo indique. Dicha condición de parada puede utilizar cada uno de los siguientes criterio o una combinación de ellos: cantidad de iteraciones realizadas, mejoría promedio de los individuos, tiempo de ejecución del algoritmo, cuando el fenotipo de los individuos hayan alcanzado cierto umbral, entre otras opciones.

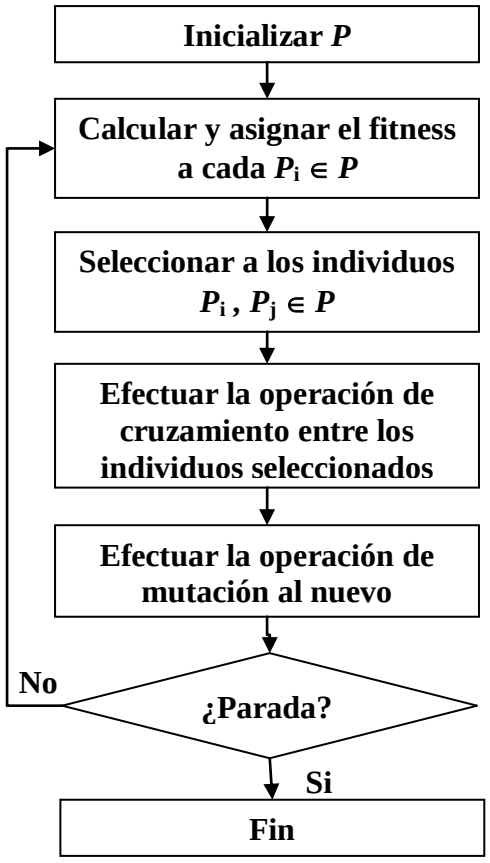


Figura 12. Diagrama de flujo de un Algoritmo Evolutivo simple.

Dentro del ciclo, se calcula el valor de fitness o adaptabilidad de cada individuo $P_i \in P$, almacenándolo en $fitness_i$. Conociendo la calidad de un individuo a través de su valor de fitness, se procede a realizar la selección de dos individuos según alguna de las estrategias descriptas anteriormente. Siendo P_i , $P_j \in P$ los individuos seleccionados en el paso anterior, se procede a la operación genética de cruzamiento (e.g. implementándola como se muestra en la Figura 10, al copiar la información genética de cada padre al nuevo individuo particionando el espacio en dos, sin que exista solapamiento). Habiendo obtenido un nuevo individuo en la etapa anterior, se realiza la operación de mutación sobre el mismo en forma similar al ejemplo propuesto en la Figura 11. El algoritmo reinicia el ciclo, siempre y cuando el criterio de parada así lo indique, sumando uno al valor de k y pasando a la generación I_k.

Una vez concluido el ciclo exitosamente, y siendo que la población evolucionó I_k veces, el algoritmo finaliza su ejecución presentando los $||P||$ individuos que fueron evolucionando en las sucesivas generaciones, hasta alcanzar el estado actual.

Existen diversos buenos Algoritmos Evolutivos Multi-Objetivo (MOEA) [17, 27] de los que se puede hacer uso para resolver el problema de encaminamiento en redes dinámicas periódicas. Entre dichos algoritmos se encuentra el *Non-dominated Sorting Genetic Algorithm II* (NSGA2) en el cual, antes de realizar el proceso de selección, se ordena a los individuos según su nivel de no dominancia formando un primer frente Pareto, luego se deja de tomar en cuenta las soluciones de ese frente para formar un segundo frente, y así sucesivamente. El algoritmo NSGA2 [16] realiza una estimación de densidad midiendo la distancia promedio entre individuos adyacentes en cada objetivo; por último, compara la concentración del individuo teniendo en cuenta a qué frente pertenece el mismo.

Otro algoritmo muy utilizado es el *Strength Pareto Evolutionary Algorithm* (SPEA) [16, 27], el cual mantiene una población externa que representa el frente de soluciones no dominadas encontradas en todas las generaciones de la población. Este algoritmo utiliza la dominancia Pareto para valorar la adaptación de cada individuo en forma escalar y si es necesario aplica el concepto de *clustering* para reducir el número de individuos en la población externa. El algoritmo SPEA2 se basa en el *Strength Pareto Evolutionary Algorithm* antes descripto, y se diferencia en la forma de calcular el valor de adaptación o *fitness*. El valor de adaptación de cada individuo no solo se relaciona con la cantidad de individuos que lo dominan, sino además, a cuántos individuos domina el mismo.

En el presente trabajo se implementa únicamente el *Strength Pareto Evolutionary Algorithm* (SPEA), que se describe a continuación. En secciones sucesivas se establecen las métricas utilizadas y el esquema de simulación, donde se utiliza una misma topología de red dinámica periódica y un mismo tráfico en la ejecución de cada algoritmo implementado. Finalmente, se compara el desempeño de cada algoritmo a través de la calidad de las soluciones encontradas.

4.1. Strength Pareto Evolutionary Algorithm

El *Strength Pareto Evolutionary Algorithm* (SPEA) presentado en [29], implementa elitismo de forma a mantener una población externa de individuos no dominados y que son denominados soluciones óptimas, en el contexto del frente Pareto. La fuerza y la adaptabilidad, denominados en inglés *strength* y *fitness* respectivamente, son calculadas para cada individuo utilizando relaciones de dominancia Pareto como fueron definidas en la sección 3. El presente trabajo implementa el algoritmo *SPEA* basándose en el descripto en [27, 29] y modificándolo para hacer uso de grafos evolutivos que describen la topología de la red dinámica periódica en

cada era. La principal variación incorporada al algoritmo *SPEA* se basa en que un individuo se conforma por una estructura compleja y las modificaciones efectuadas por las operaciones genéticas pueden convertirlo en un individuo no válido. Una solución o individuo no válido es aquel incapaz de definir un camino que permita a un paquete ser transmitido desde el nodo remitente y ser entregado al nodo destinatario.

La población P está formada por una cantidad P_{max} de individuos. Para comparar dos individuos y concluir si uno domina al otro o ninguno de los dos es dominado por el otro, se aplica la definición de dominancia Pareto según la fórmula 3.5. En dicha fórmula, f_i representa el fenotipo del individuo P_i. Se denomina P' a la población externa que contiene a los individuos no dominados entre todas las generaciones. La fuerza de cada individuo $P_j \in P'$ se denota como $strength_j$ y es igual a la razón entre la cantidad de individuos de la población P a los que domina y la cantidad de individuos que conforman P más uno. Es decir:

$$strength_j \quad \frac{\| \{ i \mid 0<i\leq P_{max},\ f_i \prec f_i \} \|}{P_{max} + 1} \tag{4.2}$$

La adaptación (*fitness*) de cada individuo, es utilizada como probabilidad de que dicho individuo sea elegido como participante de las operaciones genéticas, brindando parte de la herencia genética necesaria para constituir a los individuos de la siguiente generación. El *fitness* de los individuos P_j pertenecientes a la población externa P' es calculado haciendo uso de la inversa de su *strength*:

$$fitness_j = \frac{1}{strength_j} \tag{4.3}$$

En cambio, el *strength* de los individuos P_i pertenecientes a la población P, es calculado de la siguiente forma. Se forma el conjunto de los individuos $P_j \in P'$ tal que P_j domina a P_i para el cual se está calculando el *strength* (i.e. $f_i \prec f_j$). Se suma uno a la sumatoria de los $strength_j$ del conjunto antes descripto. Es decir:

$$strength_i = 1 + \sum_{\forall i.\ 0<i<\|P'\| \ \mid\ fi \prec fi} strength_j \tag{4.4}$$

Nuevamente, el *fitness* de los individuos P_i pertenecientes a la población *P* es la inversa de su *strength*:

$$fitness_i = \frac{1}{strength_i} \tag{4.5}$$

Un individuo denota una solución. Cada individuo está compuesto por una estructura que contiene tablas de ruteo; específicamente, contiene un conjunto de tablas para cada nodo *n* existente en el grafo evolutivo. Cada conjunto de tablas almacena información para las eras definidas en la red dinámica periódica. Sea *n* un nodo intermedio (i.e. forma parte de un camino), el cual recibe un paquete en la era *e,* o un nodo remitente el cual crea un paquete de datos en la era *e*. En ambos casos, dicho nodo utiliza la sección de la tabla correspondiente a dicha era *e* y obtiene información acerca de a qué nodo vecino debe transmitir el paquete y en qué era estará activo el enlace a ser utilizado. Aun cuando un enlace entre dos nodos se encuentra activo en varias eras distintas, es posible que utilizar la era más cercana a aquella en la que el paquete se encuentra disponible para ser enviado, no sea la mejor acción a realizar.

La Figura 13 muestra las tablas de encaminamiento contenidas en una posible solución al problema de encaminar paquetes en una red dinámica periódica pequeña como la mostrada más adelante en la Figura 17. En el ejemplo, la red define 15 eras y está compuesta por 5 nodos identificados como “a”, ”b”, “c”, “d” y “e”. Existe una tabla por cada una de las 15 eras, donde cada fila corresponde a la información competente a cada uno de los nodos antes citados y que se identifican en la primera columna. La información útil para cada nodo es únicamente aquella que se encuentra en la fila que comienza con su identificador, en cada era. Las tablas mostradas en la Figura 13 podrían ser particionadas y recombinadas para que cada nodo contenga una única tabla donde las filas están indexadas por el número de era, y las columnas por el nodo destinatario. A modo de ejemplo, si el nodo “c” recibe un paquete en la era 5, debe extraer de la cabecera del paquete el identificador del nodo destinatario. Luego, dicho nodo “c” debe recuperar la tabla rotulada con el número 5 y extraer la información que se encuentra en la intersección de la fila rotulada con el identificador “c” y la columna rotulada con el identificador del nodo destinatario. La información contenida en cada celda de la tabla es una dupla *<n’, e’>*, donde *n’* es un nodo adyacente a *n*, y existe entre los nodos *n* y *n’* un enlace activo en la era *e’*, entre otras eras. De esta forma, el nodo *n* calendariza al paquete para ser retransmitido al nodo *n’* en la era *e’*.

1	a	b	c	d	e
a		b3	c1	b3	c7
b	a3		c7	d1	e1
c	a1	b7		b10	e3
d	b1	b1	e10		e6
e	c7	b1	c3	d6	

6	a	b	c	d	e
a		b10	c1	b10	c7
b	a10		c7	d10	e12
c	a7	b7		b10	e7
d	b10	b10	b10		e6
e	c7	b12	c7	d15	

11	a	b	c	d	e
a		b3	c1	b3	c1
b	a3		c10	d1	e12
c	a13	b7		e11	e11
d	b1	b1	e15		e15
e	c11	b12	c11	d15	

2	a	b	c	d	e
a		b3	c7	b10	b10
b	a3		c7	d5	e12
c	a7	b7		b10	e3
d	e6	b5	e6		e6
e	c7	b12	c3	d6	

7	a	b	c	d	e
a		b10	c7	b10	c7
b	a10		c10	d10	e12
c	a7	b7		b7	e7
d	b10	b10	b10		e15
e	c7	b12	c7	d10	

12	a	b	c	d	e
a		b3	c13	b3	c1
b	a3		c7	d1	e12
c	a13	b7		b10	e3
d	b1	b1	b10		e15
e	c7	b12	c3	d15	

3	a	b	c	d	e
a		b3	c7	b3	c7
b	a3		c7	d5	e12
c	a7	b7		b10	e7
d	b5	b5	b10		e6
e	c7	b12	c3	d10	

8	a	b	c	d	e
a		b10	c13	b10	b10
b	a10		c10	d10	e12
c	a13	b10		b10	e11
d	b10	b5	b10		e10
e	d10	b12	c11	d10	

13	a	b	c	d	e
a		b3	c1	b3	c7
b	a3		e1	d1	e1
c	a13	b7		b10	e3
d	b1	b1	e6		e6
e	c7	b1	c3	d15	

4	a	b	c	d	e
a		b10	c7	b10	c7
b	a10		c7	d5	e12
c	a7	b10		b7	e7
d	b10	b5	b10		e6
e	c7	b12	c7	d6	

9	a	b	c	d	e
a		b10	c13	b10	c1
b	a10		c10	d10	e12
c	a1	b10		e11	e11
d	b10	b10	b10		e10
e	b1	b12	c11	d10	

14	a	b	c	d	e
a		b3	c1	b10	c7
b	a3		c7	d1	e1
c	a1	b7		b10	e3
d	b10	b1	b10		e15
e	b1	b1	c3	d15	

5	a	b	c	d	e
a		b10	c7	b10	c7
b	a10		c7	d5	e12
c	a7	b10		e7	e11
d	b10	b5	e6		e6
e	c7	b1	c7	d6	

10	a	b	c	d	e
a		b10	c13	b10	b10
b	a10		c10	d10	e12
c	a13	b10		b10	e11
d	b10	b10	b10		e10
e	c11	b12	c11	d10	

15	a	b	c	d	e
a		b3	c1	b3	c1
b	a3		c7	d1	e1
c	a1	b10		b10	e3
d	b1	b1	b10		e15
e	b1	b1	c3	d15	

Figura 13. Posible solución al problema de encaminar paquetes en una red dinámica periódica compuesta por 15 eras y 5 nodos (ver Figura 17).

El diagrama de flujo que se muestra en la Figura 14, corresponde a la implementación del algoritmo *SPEA* modificado, propuesto en el presente trabajo. La población *P* está compuesta por individuos cuya cantidad es constante y definida desde un comienzo. Dicha cantidad se denota como $||P||$ o P_{max}.

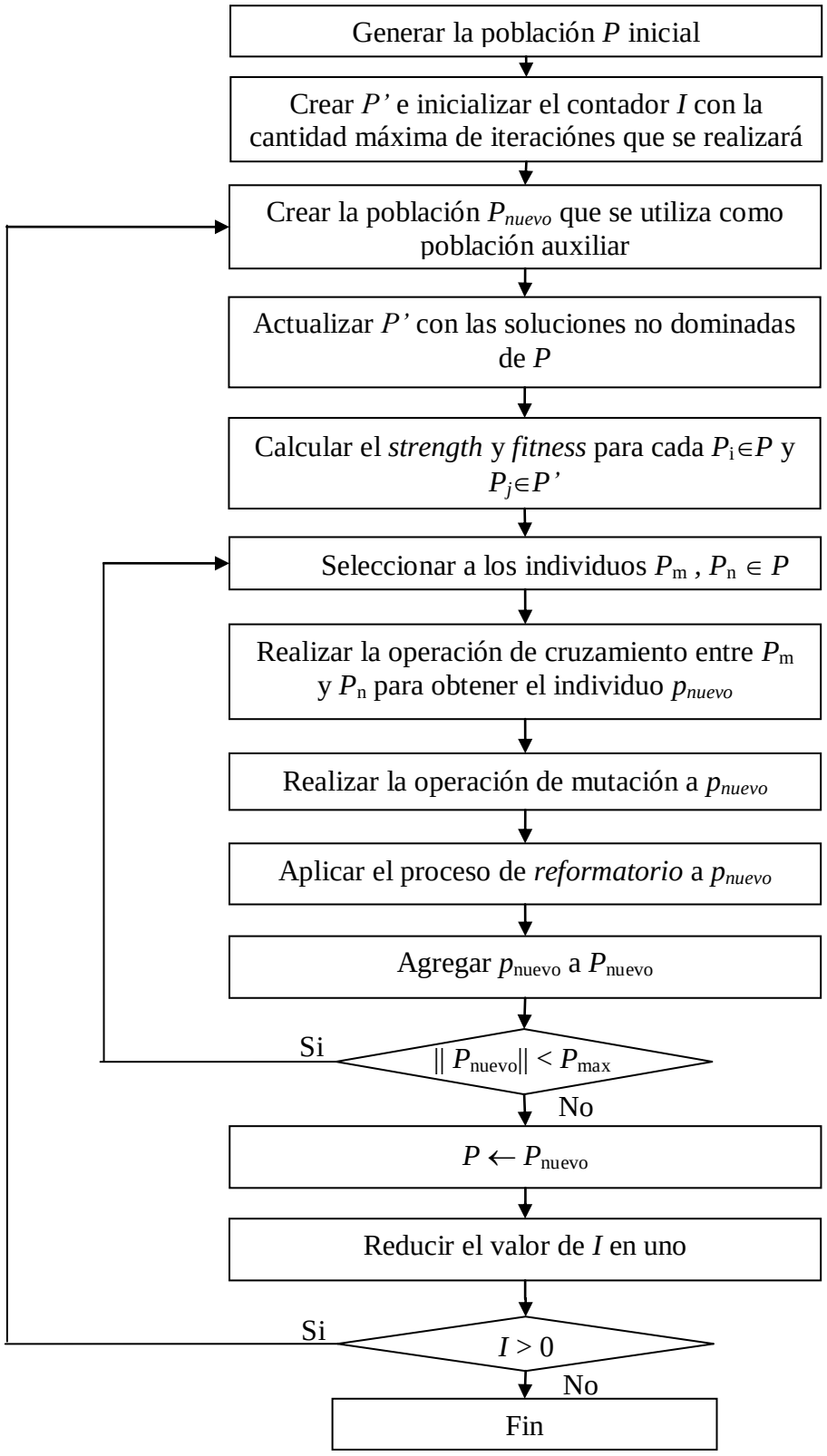


Figura 14. Diagrama del algoritmo *SPEA* implementado en el presente trabajo.

Al inicio del algoritmo se crea la población *P* con individuos generados en forma aleatoria. El algoritmo realiza una cantidad definida de iteraciones para que la población evolucione.

Dicha cantidad de iteraciones se asigna a la variable *I* en un inicio, cuyo valor se reduce en cada iteración del ciclo hasta alcanzar cero. En el siguiente paso, se crea la población externa *P'*, que en un comienzo se encuentra vacía, y es donde se mantiene el conjunto de soluciones no dominadas que conforman frente Pareto de todos los individuos que existieron en la ejecución del algoritmo. Luego, el algoritmo da inicio al ciclo de evolución. Comenzando dicha etapa, se crea e inicializa el conjunto vacio P_{nueva}, que funciona como un contenedor temporal y se actualiza la población externa *P'* copiando a ella los individuos no dominados de la población *P*, y eliminando de la misma a los individuos que existiendo previamente en *P'*, son dominados por algún individuo que se insertó recientemente.

En el siguiente paso, se calculan los valores de fuerza (*strength*) y adaptabilidad (*fitness*) para cada individuo P_i perteneciente a la población *P* y para cada individuo P_j perteneciente a la población externa *P'*. Para ello, se utilizan las fórmulas (4.2) a (4.5).

La etapa de selección hace uso de los valores de *fitness* calculados en los pasos anteriores. Se utiliza una *ruleta* como método de selección para elegir a los individuos de los que se copiará parte de la información genética de manera a crear nuevos individuos. La probabilidad de seleccionar a un individuo está dada por la fórmula (4.1). Dos individuos, denominados P_m y P_n, son elegidos de entre el conjunto resultante de la unión de la población *P* y la población externa *P'*.

Haciendo uso de la operación genética de cruzamiento, se crea el individuo p_{nuevo} tomando información de los individuos P_m y P_n antes seleccionados (ver Figura 10). Luego, dicho individuo p_{nuevo} es sometido a la operación de mutación, donde con cierta probabilidad, se incluyen alteraciones al valor almacenado en sus genes (ver Figura 11).

Al inicio de la sub-sección se introdujo el concepto de validez de un individuo. Las operaciones genéticas de cruzamiento y mutación efectúan cambios en el genotipo de un individuo considerando ciertos aspectos fundamentales pero dejando otros sin consideración. De esta forma se permite amplia diversidad y variedad en las alteraciones posiblemente realizadas y cuya aplicación resulte en cambios profundos en el individuo. Para el caso particular del problema de encaminar paquetes en una red dinámica periódica, los cambios introducidos deben respetar reglas como, por ejemplo, que un nodo no puede pretender enviar un paquete en forma directa a otro nodo, si el mismo no es adyacente (i.e. no comparten un

enlace activo), y que las eras posibles para intercambiar información entre dos nodos, son únicamente aquellas en las que existe un enlace activo entre dichos nodos.

Un individuo es considerado como no válido, si no consigue establecer un camino que permita a un paquete de datos ir desde cualquier nodo remitente hasta cualquier otro nodo destinatario, y habiéndose creado dicho paquete en cualquiera de las eras definidas en el grafo evolutivo que describe a la red dinámica periódica. Además, si un individuo propone un camino desde un nodo remitente hasta un nodo destinatario en una era definida, y dicho camino contiene un bucle, entonces el individuo es considerado como no válido.

Siguiendo con la descripción del diagrama de flujo presentado en la Figura 14, el individuo p_{nuevo}, que es el resultado de la aplicación de las operaciones genéticas antes explicadas, puede ser o no válido. El individuo atraviesa por un proceso que verifica su validez, y modifica la información en los genes hasta convertirlo en un individuo válido, en caso que no lo sea. El presente trabajo propone e implementa un proceso llamado *reformatorio*, que es el encargado de diferenciar a un individuo válido de uno no válido, y además, realizar las modificaciones necesarias a su información genética para convertirlo en un individuo válido (ver sección 4.2.).

Una vez que el individuo p_{nuevo} fue creado y la función *reformatorio* asegura que es un individuo válido, p_{nuevo} es incluido a la población auxiliar P_{nuevo}. El proceso es repetido hasta que la población P_{nuevo} contenga una cantidad P_{max} de individuos (i.e. la selección con el método *ruleta*, las operaciones genéticas de cruzamiento y mutación, y el proceso de *reformatorio* se realizan P_{max} veces).

Luego de finalizada la etapa de creación de la población auxiliar P_{nuevo}, se considera que dicha población es la evolución de la población P. A continuación, se descarta a los individuos que componen la población P reemplazándolos con los individuos de P_{nuevo}. Por último, se resta uno al valor del contador de iteraciones I. Mientras el valor de I sea mayor a cero, el algoritmo sigue evolucionando a la población P, para lo cual continúa desde donde se creó e inicializó al contenedor de individuos P_{nuevo} como conjunto vacío. Al término de la ejecución del algoritmo, se presentan los individuos que conforman la población externa P' como el conjunto de soluciones encontradas.

Dentro del dominio del problema de encaminar paquetes en una red dinámica periódica, se describe a continuación lo relativo al modelado e implementación del algoritmo *SPEA* con sus modificaciones, en el presente trabajo.

Se denomina *N* al conjunto de nodos y *E* al conjunto de eras que forman parte de un grafo evolutivo, o por sus siglas en inglés *EG* (Evolving Graph). Se denomina n_r al nodo remitente y n_d al nodo destinatario, ambos pertenecientes a *N*. Se denota $e \in E$ como una era en particular. La terna conformada por $<n_r, n_d, e>$ representa a un paquete en el nodo n_r, el cual debe ser entregado al nodo n_d siendo *e* la era actual en el grafo evolutivo. En la descripción ternaria de un paquete, es indistinto si n_r es el nodo que creó a dicho paquete de datos en un primer momento, o si es un nodo interno del camino recorrido por el paquete, y funciona como ente de reenvío.

Cada individuo incorpora una estructura de datos donde se almacenan tablas conteniendo información acerca de cómo encaminar cualquier paquete $<n_r, n_d, e>$ siendo n_r y n_d cualesquiera dos nodos diferentes pertenecientes a *N*, en cualquier era *e* perteneciente a *E* en que se precise realizar una acción. Dicha información de ruteo es representada por la dupla $<n', e'>$, donde n' pertenece a *N* y e' pertenece a *E*. En la dupla, n' es el identificador del siguiente nodo en el camino hacia el nodo destinatario y e' es la era durante la cual el enlace entre los nodos n_r y n_d debe utilizarse para transmitir el paquete. El nodo n' es un vecino de *n* (i.e. nodo adyacente con quien comparte un enlace activo). El párrafo anterior advierte que es indiferente si n_r es un nodo de reenvío o es el nodo donde se creó el paquete de datos. La aseveración se basa en lo siguiente: cada nodo conoce únicamente a qué nodo vecino debe transmitir el paquete y en qué era, pero no posee una visión global del camino. Dicho nodo de reenvío no conoce qué otros nodos serán visitados posteriormente por el paquete, ni cuál es la distancia restante por recorrer.

En los algoritmos evolutivos, es imprescindible evaluar el desempeño de un individuo; a tal efecto, en este trabajo se simula el envío de un paquete por cada combinación posible de nodo remitente, nodo destinatario y era de creación del paquete. El algoritmo *SPEA* implementado en este trabajo, define las siguientes funciones objetivo:

a) La suma de saltos necesaria para que todos los paquetes sean entregados, cada uno a su respectivo nodo destinatario.

b) La suma de eras transcurridas entre la creación de los paquetes en el nodo remitente, y la era de arribo de todos los paquetes a sus respectivos nodos destinatario. Esta función objetivo es también llamada suma de eras de vida.

c) La suma de eras transcurridas entre la transmisión de los paquetes utilizando cada uno un enlace activo (i.e. realizando el primer salto), hasta que cada paquete terminó de viajar a través de la red llegando a su nodo destinatario. Esta Función también puede denominarse como la suma de eras viajadas.

Habiendo descripto las funciones objetivo, el camino más corto es el que utiliza la cantidad mínima de saltos; es decir, minimiza la función objetivo (a). El camino que permite la llegada más temprana de paquetes a destino, es aquel donde la suma de eras vividas es mínima; es decir, minimiza la función objetivo (b). Por último, el camino más rápido es aquel cuya suma de eras viajadas es mínima; es decir, minimiza la función objetivo (c).

A continuación, se presenta el Algoritmo 5 que es la implementación del algoritmo *SPEA* modificado cuyo diagrama de flujo se muestra en la Figura 14.

```
/*
Variables:

P: población de individuos, de tamaño constante P_max = ||P||
P': externa de individuos no dominados, de tamaño ||P'|| ≤ P'_max
P_i: individuos pertenecientes a la población P
P_j: individuos pertenecientes a la población P'
I_max: Cantidad de iteraciones a realizar
I_k: Cada una de las iteraciónes considerando 0 < k ≤ I_max

*/
   1. Inicializar P' ← ∅
   2. Generar p_i, 0 < i ≤ P_max, conformando la población inicial P
   3. k ← 1
   4. Mientras k ≤ Imax hacer:
      a. Actualizar P' con los individuos no dominados de P
      b. Mientras ||P'|| > P'_max
         • Elegir al azar j, 0 < j ≤ ||P'||
         • Eliminar P_j de P'
      c. Fin
      d. ∀P_i∈P calcular fitness_i, ∀P_j∈P' calcular fitness_j
      e. Inicializar P_nuevo ← ∅, una nueva población
      f. Repetir P_max veces
         • p_nuevo ← cruzamiento entre dos individuos elegidos de P∪P'
         • Aplicar mutación y rectificación a p_nuevo
         • Agregar p_nuevo a P_nuevo
      g. Fin
      h. k = k + 1
      i. p ← p_nuevo
   5. Retornar la población P' encontrada
   6. Fin
```

Algoritmo 5. Pseudocódigo del algoritmo *SPEA* implementado en este trabajo.

Anteriormente se definió el calificativo de válido o no válido, atribuyéndolo a un individuo. Ahora se especifica su significado en el contexto del encaminamiento en redes dinámicas periódicas. Un individuo es válido si, de acuerdo a la información de ruteo que posee, se asegura la existencia de una ruta desde el nodo remitente hasta el nodo destinatario para cada uno de los posibles paquetes enviados, cualquiera sea la era.

En toda generación de una población, cada individuo pasa a través de un proceso de cruzamiento y de mutación, utilizados para obtener nuevas soluciones evolucionadas. Como resultado de dicho proceso, la información de ruteo en dichos individuos nuevos puede contener ciclos innecesarios en el camino. Esto pasa cuando un nodo es visitado más de una vez en un mismo camino. Otro problema que potencialmente puede surgir es la fragmentación de la red, imposibilitando la llegada del paquete hasta el destino. La fragmentación de la red se da si respetando la restricción de la no existencia de ciclos en un camino, ya se recorrió la red diferenciando dos regiones aisladas entre si, como se muestra en la Figura 15.

En efecto, la Figura 15 muestra una situación donde el paquete ya recorrió [A C E G]. La línea punteada color verde indica los enlaces ya utilizados. Es decir que los nodos extremo en cada uno de dichos enlaces, ya fue visitado. Respetando la no inclusión de ciclos en el camino, si el paquete se encuentra en una de las regiones diferenciadas, y el nodo destinatario se encuentra en la otra región, no existe forma de llegar a destino. En la figura, el paquete se encuentra actualmente en el nodo G, y no existe camino posible que le permita llegar al nodo I, sin pasar a través de un nodo que forma parte de la ruta ya utilizada (i.e. generando un ciclo). En este caso, se dice que la red se encuentra fragmentada o particionada.

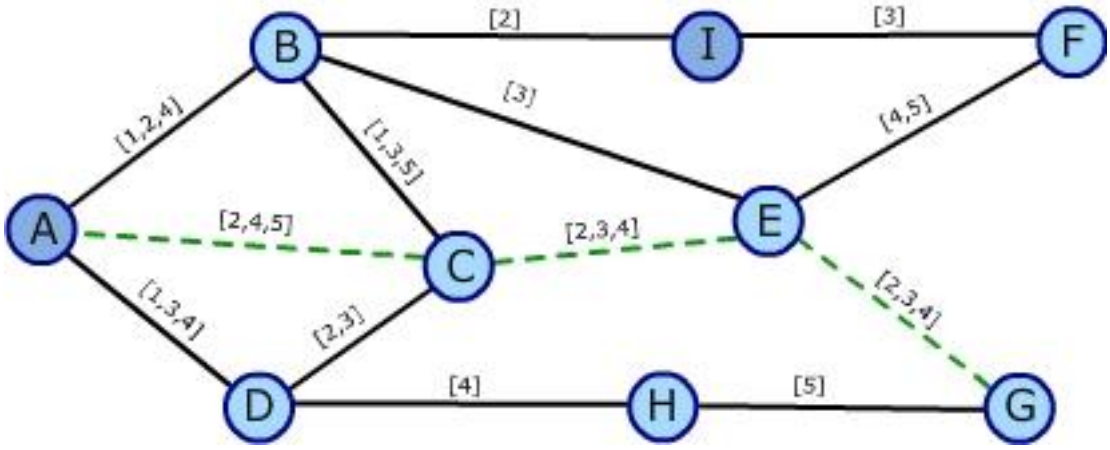


Figura 15. Una red fragmentada. El paquete creado por el nodo A tiene como destinatario al nodo I. Los enlaces ya utilizados son punteados y de color verde. El paquete se encuentra actualmente en el nodo G.

Para que un *Algoritmo Evolutivo Multi-Objetivo* (MOEA) sea realmente de utilidad, los nuevos individuos deben ser reparados para convertirse en soluciones válidas. Consecuentemente, el presente trabajo propone un método novedoso llamado *reformatorio* para reparar a un individuo, obteniendo de esta forma un individuo válido que es capas de solucionar el problema de encaminamiento en el grafo evolutivo estudiado.

4.2. Reformatorio

Cada población *P* es calculada en una iteración del algoritmo, y a cada iteración se la denota como I_k. En la iteración I_{k+1} se utilizan individuos de la iteración I_k para realizar el proceso de generación de una nueva población *P*. Cada individuo que conformará la población es el resultado de tres operaciones genéticas: cruzamiento, mutación y rectificación (ver Figura 14 y Algoritmo 5). En la sección anterior se definió el significado de un ciclo o bucle en el camino, así como de la fragmentación de la red (ver Figura 15).

Es necesario verificar a cada individuo y convertirlo en una solución válida si no lo es. Para ello, se inspeccionan las tablas de ruteo que forman parte del mismo. Se envían paquetes de a uno por vez, el conjunto de paquetes incluye todas las combinaciones posibles de paquetes $<n_r, n_d, e>$, donde n_r y n_d, son los nodos remitente y destinatario respectivamente, y *e* es la era en la que el paquete es creado.

Internamente, el algoritmo hace uso de dos listas ordenadas a los efectos de registrar el proceso de construcción o verificación de cada camino. Una lista es denominada *tabú* y cumple la función de almacenar el identificador de los nodos ya visitados por el paquete en el supuesto camino desde n_r hasta n_d. En cada salto simulado, se corrobora que el siguiente nodo al cual debe ser transmitido el paquete, no figure en la lista *tabú*. Si este no fuera el caso, y el identificador del nodo ya se encuentra en dicha lista, entonces se indica la detección de un ciclo o bucle y se toman las acciones necesarias.

La segunda lista es denominada *tabú_n* y su función es la de almacenar el conjunto de duplas *<n', e>* ya utilizadas por el nodo *n* en el camino bajo verificación. Dentro de la simulación del viaje, se denota a *n* como el nodo donde se encuentra el paquete, y la dupla antes nombrada indica al nodo *n'* que es vecino de *n* y con el cuál establece un enlace activo en la era *e*, momento en el cual se transmite el paquete al siguiente nodo del camino. La lista *tabú_n* está indexada por el identificador de los nodos incluidos en el trayecto o que ya fueron probados y se los excluyó del camino.

Cuando se detecta un problema cualquiera en el camino, y si la solución al mismo implica buscar un reemplazo al salto que aparentemente generó el problema, el algoritmo debe poder sugerir un nodo adyacente a *n* y una era en la cual exista un enlace activo entre ambos nodos, para seguir el camino hacia el nodo destinatario (i.e. se sugiere una dupla *<n', e>*). Una restricción que debe respetarse al construir una sugerencia de siguiente salto, es que visitar al nodo *n'* no genere un ciclo; esto se verifica haciendo uso de la lista *tabú*. En el caso de que el algoritmo sugiera la continuación del camino a través de *n'*, resta por definirse en qué era se realizará el envío del paquete. Puede sugerirse un mismo nodo *n'* variando la era *e*. El algoritmo se asegura de la no repetición de una misma sugerencia, inspeccionando las opciones ya realizadas las cuales se encuentran en una lista apuntada por *tabú_n* y es indexada por el identificador del nodo *n*. Si para un nodo *n* ya se intentó continuar el trayecto utilizando cualquiera de los nodos adyacentes, en cualquiera de las eras en que dichos nodos mantienen un enlace activo, entonces se informa acerca de la detección de una partición y se realizan acciones para intentar solucionar el problema.

Cuando el algoritmo detecta un ciclo o una partición, realiza un proceso de retroceso (*rollback*) restaurando el estado de ciertas variables y manteniendo en forma consistente las listas *tabú* y *tabú_n*, deshaciendo de esta forma parte del camino en construcción. Una vez finalizado dicho retroceso, el algoritmo debe sugerir la siguiente dupla *<n', e>* a ser utilizada como prueba de siguiente salto en el camino hasta el nodo destinatario. Para ello, se elige aleatoriamente una de tres opciones: utilizar información calculada con el método utilizado en el algoritmo *Evolving Graph Shortest*, la calculada con el método sugerido en el algoritmo *Evolving Graph Foremost* (ambos algoritmos fueron implementados en [18] y descriptos en las secciones 2.2.1. y 2.2.2.); o como tercera opción, elegir un nodo adyacente a *n* que no se encuentre en la lista *tabú*, y a continuación elegir una era *e* en la que exista un enlace activo entre *n* y *n'* observando que *e* no se encuentre en el conjunto de eras que ya fueron sugeridas sin éxito para el nodo vecino *n'*. Esta última información es recabada de *tabú_n*.

El Algoritmo 6 muestra el pseudocódigo de este método novedoso, definido e implementado en el presente trabajo: el *reformatorio*. En dicho pseudocódigo, se simula el viaje que realizaría un paquete para cualquier par de nodos remitente y destinatario, cualquiera sea la era de creación de dicho paquete, utilizando la información que contiene el individuo.

```
/*

Variables:

tabú: conjunto ordenado de nodos ya visitados en un camino
tabú_n[n]: conjunto de duplas <n',e> donde n' es un nodo adyacente a n y
e es la era en la que ya se sugirió el enlace
n_r: nodo remitente
n_d: nodo destinatario
e_rd: era en que está disponible el paquete para ser enviado
ind: arreglo indexado por un paquete y que contiene la información <n,e>
para el siguiente salto

Sea <n_r,n_d,e_rd> un paquete a ser enviado de n_r a n_d y la era actual es e_rd

*/

1. Para cada par de nodos y cada era, marcar la información como prueba
2. Mientras exista información sin confirmar como correcta hacer:
   A) Para cada par de nodos y cada era <n_r,n_d,e> hacer:
      a. Mientras n≠n_d hacer:
        1) Si la información en ind[<n,n_d,e>] está marcada como prueba
            • Marcar la información de ind[<n,n_d,e>] como sugerido
            • <n',e'> ← enlace indicado en ind[<n,n_d,e>]
            • Si n' está en tabú o <n',e'> está en tabú_n[n],
              retroceder y probar otra opción
        2) Si la información en ind[<n,n_d,e>] está marcada como correcta
            • Si n' está en tabú o <n',e'> está en tabú_n[n],
              retroceder y probar otra opción
        3) Si la información en ind[<n,n_d,e>] está marcada como sugerido
            • Calcular el conjunto de nodos adyacentes a n que no están
              en tabú, y las eras disponibles que no están en tabú_n[n]
            • Si no quedan nodos adyacentes y eras disponibles,
              entonces retroceder y probar otra opción
            • Escoger al azar una de las tres posibilidades: asignar a
              ind[<n,nd,e>] la información extraída de EG shortest, de
              EG foremost, o buscar en forma aleatoria un enlace
        4) Agregar n al tope de tabú e ind[<n,n_d,e>] a tabú_n[n]
        5) <n, e> ← ind[<n,n_d,e>]
      b. Fin
   B) Fin
   C) Marcar la información como correcta para cada <n,n_d,e> que forma
      el camino
3. Fin
```

Algoritmo 6. Pseudocódigo de la función reformatorio.

Inicialmente, toda la información de encaminamiento es rotulada como ***prueba*** para asegurar su posterior control durante el Algoritmo 6. En caso surjan dificultades con el camino formado por la información contenida en forma previa, el algoritmo sistemáticamente altera dicha información de ruteo y la rotula como ***sugerida***. La sugerencia de información permite

que la simulación siga adelante con la exploración del camino hasta que el paquete llega al destinatario. En caso de detectarse un ciclo o una partición en la red y no encontrar una opción válida de modificación, se aplica un retroceso (*rollback*) el cual deshace el camino hasta donde corresponda, para optar por un camino alternativo.

La información de siguiente salto sugerida por el algoritmo, inicialmente es válida dada la corroboración de que el siguiente nodo sugerido *n'* no se encuentra en *tabú*, y que <*n'*, *e*> no se encuentra en *tabú_n* para el nodo *n*. Dicho control, no garantiza que la sugerencia resuelva el problema. Con motivo de la complejidad del problema, se puede estar generando una partición sin saberlo. Esto ocurre en el ejemplo de la Figura 15, cuando el algoritmo sugiere que el siguiente salto, luego del nodo identificado como E, sea el nodo identificado como G. Allí, la partición fue establecida pero todavía no es detectable.

Habiendo alcanzado el paquete en la simulación al nodo destinatario, la información de encaminamiento involucrada en el viaje (i.e. todas las duplas <*n'*, *e*> involucradas en el camino) es rotulada como ***correcta***.

El Algoritmo 6 finaliza cuando toda la información es confirmada como ***correcta***. En dicho caso, puede asegurarse que cualquiera sean los nodos remitente y destinatario, y cualquiera sea la era *e* en que un paquete es creado para ser enviado, dicho paquete <n_r, n_d, *e*> atravesará con éxito la red culminando su camino.

5. Métricas de Eficiencia

Los algoritmos que no utilizan grafos evolutivos para representar la topología de las redes dinámicas periódicas, pueden no lograr construir un camino entre los nodos remitente y destinatario. Esto ocurre en las simulaciones realizadas y datos mostrados en [15, 18, 21]. En una red dinámica periódica donde no existen nodos aislados (ver Figura 16), necesariamente existe un camino a través del cual un paquete puede viajar desde el remitente hasta el destinatario [18].

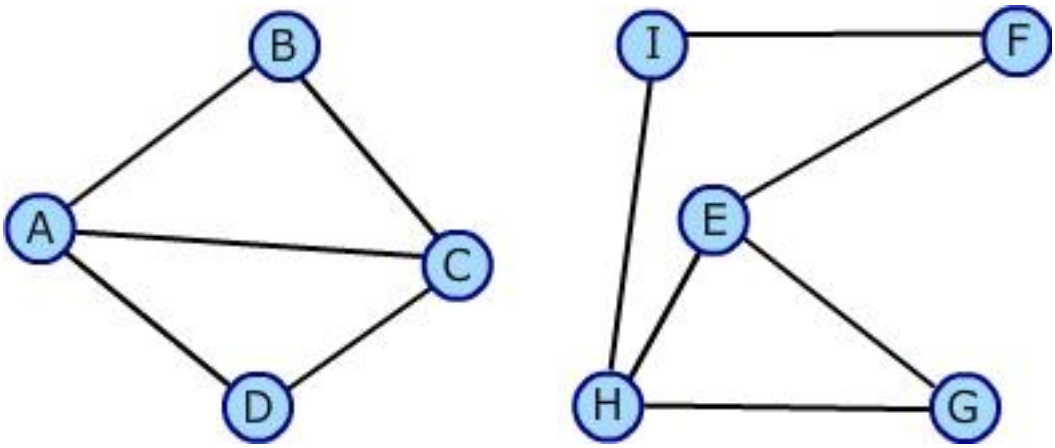


Figura 16. Una red con islas. Una red dinámica periódica compuesta por dos grupos aislados de nodos. No existe forma de interconectar los nodos A y G.

Los algoritmos *Destination-Sequenced Distance-Vector* (DSDV) [20], *Dynamic Source Routing* (DSR) [12], *Ad-hoc On-Demand Distance Vector* (AODV) [26], *Evolving Graph Foremost* (EG Foremost) y *Evolving Graph Shortest* (EG Shortest) [18], fueron implementados en el presente trabajo. En todos los casos, los algoritmos utilizaron las métricas de eficiencia (objetivos) introducidas en la sección 4.1. y detalladas a continuación.

Cantidad de Saltos

Un salto es el evento ocurrido cuando un nodo transmite un paquete a otro nodo adyacente, que lo recibe. La cantidad de saltos mide cuántos nodos atravesó un paquete para llegar desde el nodo remitente hasta el nodo destinatario.

Cada camino entre un par de nodos *{remitente; destinatario}* tiene una cantidad de saltos mayor a cero. El conjunto de caminos más cortos es el de los caminos cuya cantidad de saltos es mínima.

Cantidad de Eras de Vida

La cantidad de eras de vida es la suma de eras durante las cuales un paquete se encuentra en la memoria de uno o más nodos. Se contabiliza desde la era en la cual el paquete es creado por el nodo remitente, hasta su recepción por el destinatario donde se extraen los datos que contiene y el paquete como tal, deja de existir. La cantidad de eras de vida puede ser cero o nula, este es el caso si suceden los siguientes hechos en una misma era: el paquete es creado, comienza su viaje y por último, llega hasta el nodo destinatario; es decir, la topología de la red no sufrió cambios, por lo que el grafo evolutivo no realizó ninguna transición de eras.

Cada camino entre un par de nodos *{remitente; destinatario}* tiene una cantidad de eras de vida. El conjunto de caminos que permiten al paquete llegar al nodo destinatario en la era más temprana, es el de los caminos cuya cantidad de eras de vida es mínima.

Cantidad de Eras de Viaje

La cantidad de eras de viaje es la suma de eras durante las cuales el paquete se encuentra atravesando efectivamente los nodos intermedios. Se contabiliza desde la era en que el remitente envía el paquete al primer nodo intermedio, hasta que dicho paquete es recibido por el destinatario. La cantidad de eras de viaje puede ser cero o nula si la topología de la red dinámica periódica indica que todos los enlaces que conforman al camino existen (i.e. se encuentran activos) en una misma era. El paquete es enviado por el nodo remitente al primer nodo de reenvío, y luego es transmitido entre cada par de nodos de reenvío hasta alcanzar al nodo destinatario. Cabe destacarse que, para todo viaje realizado, la cantidad de eras de viaje debe necesariamente ser menor o igual a la cantidad de eras de vida.

Cada camino entre un par de nodos *{remitente; destinatario}* tiene una cantidad de eras de viaje. El conjunto de caminos más rápidos es el de los caminos cuya cantidad de eras de viaje es mínima.

En resumen, las métricas de eficiencia de dividen en tres: cantidad de saltos, cantidad de eras de vida y cantidad de eras viajadas. El Algoritmo Evolutivo Multi-Objetivo propuesto en el presente trabajo para resolver el problema de encaminamiento de paquetes en redes dinámicas periódicas, intenta minimizar las tres funciones objetivo definidas en la sección 4.1 haciendo uso de las métricas anteriormente citadas.

6. Simulaciones

Para asegurarse de que un Algoritmo Evolutivo Multi-Objetivo (MOEA) en particular, es una buena alternativa en la resolución del problema de encaminamiento en redes dinámicas periódicas, se llevan a cabo muy extensas simulaciones entre dicho algoritmo propuesto en este trabajo y los algoritmos anteriormente descriptos: *Destination-Sequenced Distance-Vector* (DSDV), *Dynamic Source Routing* (DSR), *Ad-hoc On-Demand Distance Vector* (AODV), *Evolving Graph Foremost* (EG Foremost) y *Evolving Graph Shortest* (EG Shortest) ya desarrollados en las secciones 2.1. y 2.2.

En el presente trabajo, las simulaciones se desarrollan utilizando tres escenarios diferentes, los cuales se describen más adelante. Cada escenario está compuesto por una topología de red y una definición de tráfico. La topología de red especifica los nodos existentes, los enlaces entre ellos y las eras en que se encuentran activos, mientras que la definición de carga de tráfico proporciona información detallada acerca de cada paquete a ser transmitido.

6.1. Topología de la Red

Una topología de red especifica cada nodo *n* perteneciente al conjunto de nodos *N*, y cada uno de los enlaces establecidos entre dos nodos n_i y n_j en cada era *e* definida en la red dinámica periódica. En el presente trabajo se utilizan tres tamaños de red, las cuales son especificadas más adelante. Para todas las simulaciones realizadas, las redes dinámicas periódicas definen únicamente quince eras. A continuación, la Figura 17 muestra una red denominada pequeña.

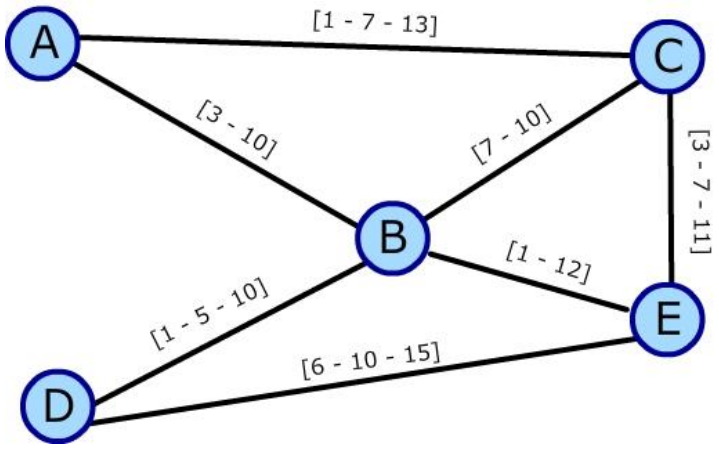


Figura 17. Una red pequeña. Grafo evolutivo que representa la topología de una red pequeña.

La Figura 17 muestra un grafo evolutivo formado por cinco nodos identificados como A, B, C, D y E. Además contiene siete enlaces. Las eras en que cada enlace se encuentra activo es indicado por los número encerrados entre corchetes encima del enlace (e.g. entre los nodos A y B existe un enlace activo en las eras 3 y 10, entre C y E existe un enlace activo en las eras 3, 7 y 11). El enlace entre los nodos D y E se encuentra activo únicamente en tres de las quince eras definidas en la red dinámica periódica.

Basándose en la topología de la red pequeña mostrada en la Figura 17, el presente trabajo define una red considerada de tamaño mediano. El grafo evolutivo que describe a dicha red, se muestra en la Figura 18. La red mediana está conformada por trece nodos y veinte enlaces.

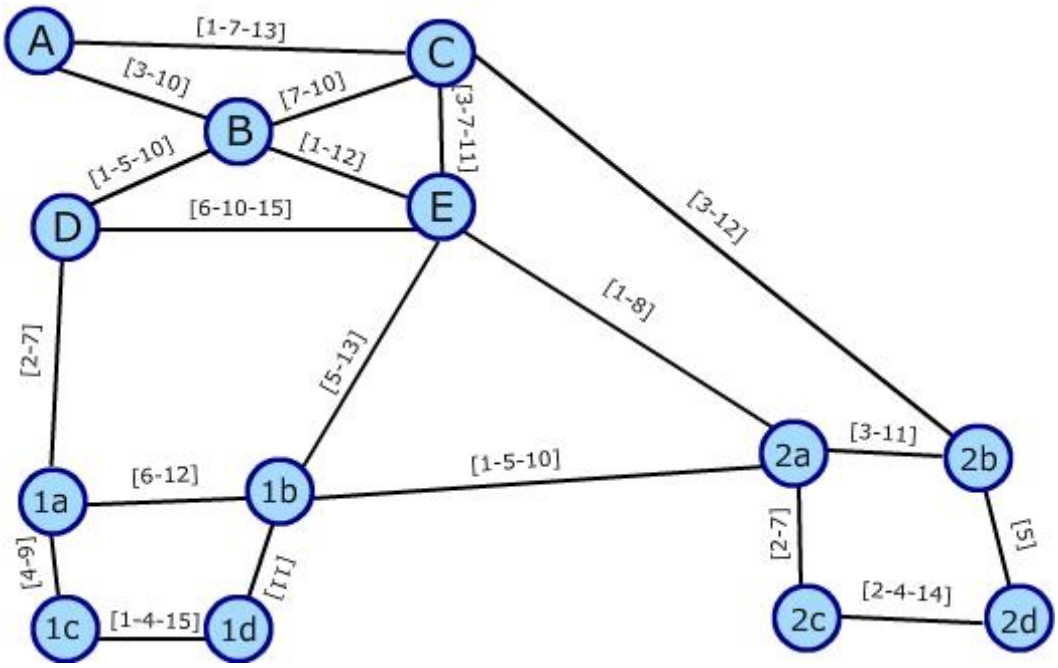


Figura 18. Una red mediana. Grafo evolutivo que representa la topología de una red mediana.

Para facilitar la comprensión de la ubicación relativa del nodo dentro de esta red en particular, el identificador de cada uno está relacionado con el subgrupo al que pertenece. De esta forma, los nodos identificados por A, B, C, D y E forman un núcleo, mientras que los nodos 1a, 1b, 1c y 1d conceptualmente forman una isla y los nodos 2a, 2b, 2c y 2d forman una segunda isla. En este caso, y en adelante, la denominación de *isla* se refiere a la conectividad o concentración de un grupo de nodos pertenecientes a la red dinámica periódica, y no a la acepción utilizada en la unidad 5 (ver Figura 16).

Se presenta además, otra red basada en la denominada pequeña (ver Figura 17). Esta red es denominada grande y se muestra en la Figura 19. La misma contiene veinticuatro nodos y treinta y siete enlaces que los conectan.

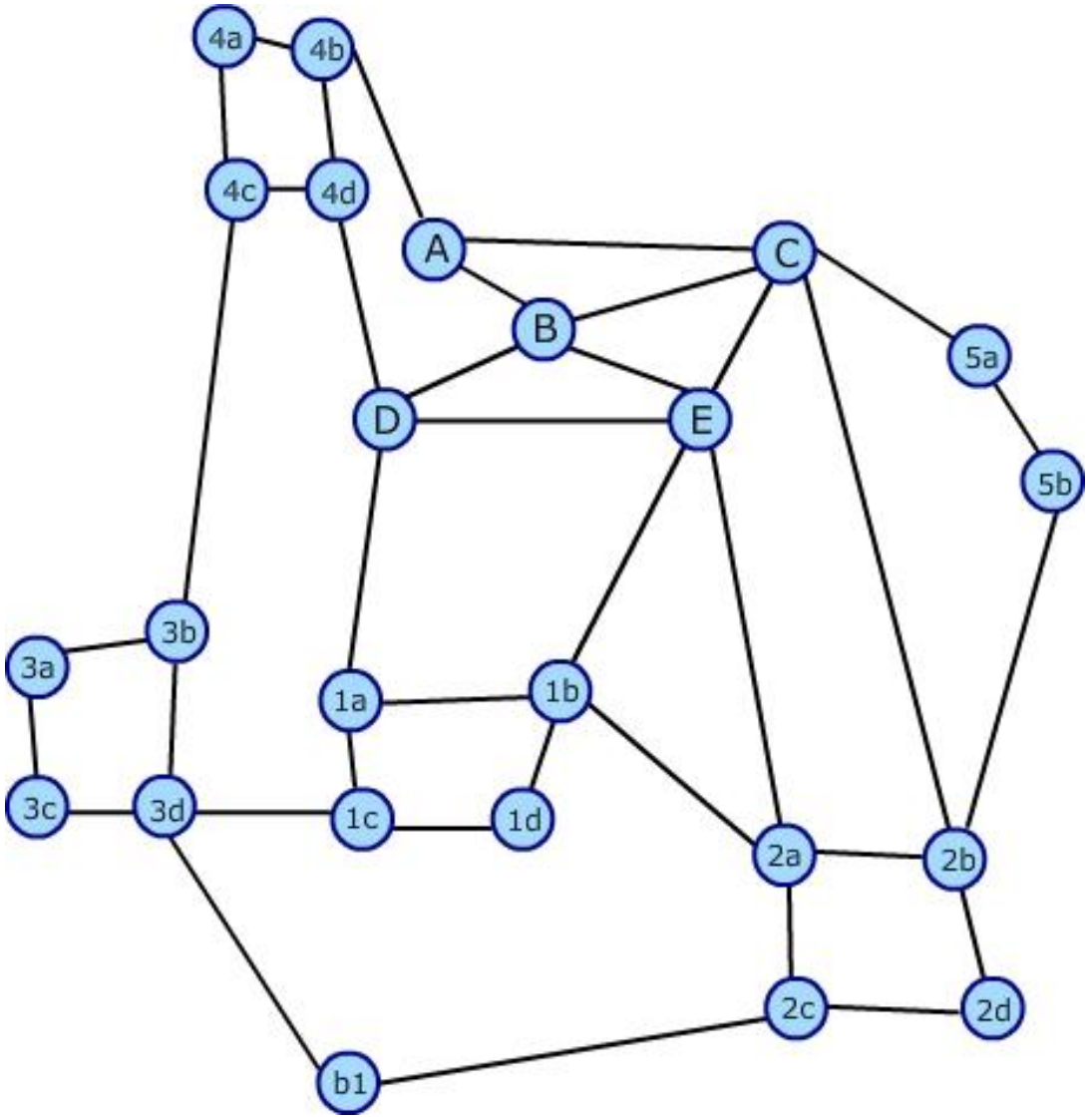


Figura 19. Una red grande. Grafo evolutivo que representa la topología de una red grande.

El método de nomenclaturas utilizado en la red mediana (ver Figura 18), es puesto en práctica nuevamente en la red grande. El nodo cuyo identificador comienza con un número, pertenece a una isla (i.e. pertenece a un conjunto de nodos agrupados conceptualmente, aunque los mismos se encuentran interconectados con todos los otros nodos de la red en forma directa o indirecta). Dentro de una isla, los nodos que la componen tienen como identificador letras (e.g.

el nodo 3c pertenece a la isla 3, dentro de la cual puede ser denominado c). Los nodos cuyo identificador inicia con una letra en mayúscula, son considerados parte del núcleo (backbone) que puede ser visto como una isla central en la red dinámica periódica. Por último, si el identificador de un nodo comienza con la letra "b" en minúscula, el nodo cumple conceptualmente la función de nodo puente uniendo a dos islas (e.g. el nodo identificado por b1 es un puente entre las islas 2 y 3).

Con objeto de facilitar la visualización de la red grande, en la Figura 19 fueron omitidas las eras en que cada enlace se encuentra activo. Dicha información se encuentra en la Tabla 1, la cual muestra un enlace por fila, donde cada fila contiene una columna por cada nodo extremo y una columna que lista las eras separadas por guiones.

Nodo	Nodo	Eras
A	B	3-5
A	C	1-2-4
B	C	1-3-5
B	D	1-2-3
B	E	3-4-5
C	E	1-4-5
D	E	1-2-4
D	1a	1-3-5
E	1b	1-2-4
1a	1b	3-5
1a	1c	2-3
1b	1d	5
1c	1d	3-4-5
2a	E	1-3-5
2b	C	1-2-4
2a	2c	1-2-3
2a	1b	1-2-3-4-5
2a	2b	3-5
2b	2d	5

Nodo	Nodo	Eras
2c	2d	3-4-5
3a	3c	2-3
3a	3b	3-5
3b	3d	5
3c	3d	3-4-5
3d	1c	3-4-5
4a	4b	3-5
4a	4c	2-3
4b	4d	5
4b	A	3-4-5
4c	4d	3-4
4d	D	4-5
4c	3b	2-4
b1	3d	1-2-3-4-5
b1	2c	1-2-3-4-5
5a	5b	1-2-3-4-5
5a	C	3-5
5b	2b	2-4

Tabla 1. Eras en las que los enlaces de la Figura 19 se encuentran activos.

6.2. Carga de Tráfico

El tráfico es considerado como la especificación de la carga en una red, i.e. la exacta descripción de cada paquete a ser transmitido. Un paquete es representado por la siguiente terna: $<n_r, n_d, e>$. En ella, n_r y n_d son los nodos remitente y destinatario respectivamente. El paquete es creado en la era e por el nodo n_r y es calendarizado para ser enviado de acuerdo a la información de la tabla de ruteo del nodo remitente.

El presente trabajo presenta simulaciones utilizando una carga pesada de paquetes de datos en la red. La cantidad de paquetes de datos considerada pesada, es calculada como el 90 % del

número de nodos multiplicado por el número de enlaces que constituyen la red dinámica periódica. En la Tabla 2 se presentan los treinta y un paquetes aleatoriamente elegidos para conformar el tráfico a ser utilizado en las simulaciones con la red pequeña (ver Figura 17). La tabla muestra la terna $<n_r, n_d, e>$, interpretando a la primera columna como el identificador del nodo remitente, en la segunda columna el identificador del nodo destinatario, y en la última columna la era en que el paquete de datos es creado en el nodo remitente.

Remitente	Destinatario	Era
A	D	1
A	D	2
A	D	3
A	D	4
A	D	5
A	D	6
A	D	7
A	D	8
A	D	9
A	D	10
A	D	11
B	E	4
B	E	6
B	E	7
B	E	8
B	E	9

Remitente	Destinatario	Era
B	E	10
B	E	11
B	E	12
B	E	13
B	E	14
E	A	1
E	A	3
E	A	5
E	A	7
E	A	8
E	A	11
E	A	12
E	A	13
E	A	14
E	A	15

Tabla 2. Tráfico pesado o congestionado para la red pequeña de la Figura 17.

Para efectuar una simulación de la red mediana (ver Figura 18) densamente cargada, nuevamente se elige un volumen de paquetes calculado como el 90 % de la cantidad de nodos multiplicado por la cantidad de enlaces, el tráfico pesado o denso para la red dinámica periódica de la Figura 18, contiene 234 paquetes. Por cuestiones de espacio, dichos paquetes son mostrados en el Anexo II. Para realizar las simulaciones en una red dinámica periódica grande, la cual se muestra en la Figura 19, se define un tráfico compuesto por 630 paquetes mostrado en el Anexo III.

6.3. Parámetros del Algoritmo SPEA

Para ejecutar el *Strength Pareto Evolutionary Algorithm* (SPEA) con las adaptaciones implementadas en el presente trabajo, como es el *reformatorio* (ver sección 4.2.), deben definirse los valores de los parámetros a ser utilizados en las simulaciones. La Tabla 3 muestra, a modo de ejemplo, el valor que toma cada uno de los parámetros para ejecutar el algoritmo *SPEA* y resolver el problema del encaminamiento en redes dinámicas periódicas, en este caso, para la red definida como pequeña, y que fue presentada en la Figura 17.

Tamaño	P_{max}	I_{max}	P'_{max}	*EG inserción*	*P_EGf*	*P_EGs*	*P_rand*
Pequeño	10	1000	10000	1	0,4	0,4	0,2

Tabla 3. Ejemplo de valores asignados a los parámetros utilizados en el algoritmo *SPEA* para la red dinámica periódica pequeña.

En la tabla 3, *Tamaño* da una estimación de la cantidad de nodos que componen la red dinámica periódica, en este caso, el valor asignado es *pequeño*. Las simulaciones realizadas en la investigación precedente a este trabajo, incluyen redes pequeñas (ver Figura 17), medianas (ver Figura 18) y grandes (ver Figura 19) [18].

El parámetro P_{max} indica el tamaño constante de la población evolutiva, en cada generación, mientras que P'_{max} representa el tamaño máximo que puede tener la población externa donde las soluciones óptimas no dominadas son almacenadas. La cantidad de iteraciones que el algoritmo lleva a cabo, está regida por I_{max}. Se denomina I_k a cada una de las iteraciones, donde $0 < k \leq I_{max}$. Los experimentos realizados incluyen uno de los siguientes valores de I_{max}: 10, 100, 1000 ó 10000.

Los algoritmos *Evolving Graph Shortest* y *Evolving Graph Foremost* son óptimos, dentro del contexto mono-objetivo [18]. Al considerar un contexto multi-objetivo, existen variaciones de las soluciones encontradas por los algoritmos *EG Foremost* y *EG Shortest*, que llegan a ser mejores logrando superar el desempeño. Las soluciones de los algoritmos *EG Foremost* y *EG Shortest*, pueden ser insertadas en la población evolutiva *P* luego de una cantidad de iteraciones definidas en el parámetro *EG inserción*. De esta modo, mejores soluciones pueden ser fácilmente encontradas cuando las tres funciones objetivo son consideradas simultáneamente: minimizar la cantidad de saltos, minimizar la cantidad de eras de vida y minimizar la cantidad de eras de viaje (ver sección 5.).

La inserción de las soluciones proporcionadas por los algoritmos *EG Foremost* y *EG Shortest* en la población externa *P'* tiene el efecto de acelerar la búsqueda de soluciones no dominadas. Además, cuando es posible, permite encontrar otras soluciones que, partiendo de dichas soluciones óptimas en el contexto mono-objetivo, las dominen en el contexto multi-objetivo.

Cuando el procedimiento de *reformatorio* halla un problema en la tabla de encaminamiento de un individuo, dicho procedimiento intenta reparar la información cambiando el valor que genera el inconveniente. Para ello, en el *reformatorio* se consulta a las

tablas de encaminamiento pre-calculadas por el algoritmo *Evolving Graph Foremost* con una probabilidad *P_EGf*, o en vez de ello, se consulta a las tablas pre-calculadas por el algoritmo *Evolving Graph Shortest* con una probabilidad *P_EGs*. Una tercera opción es, con una probabilidad *P_rand*, seleccionar en forma aleatoria a un nodo vecino válido y una era válida en la cual el enlace que llega a dicho vecino se encuentre activo. En consecuencia, el *reformatorio* escoge en forma aleatoria la estrategia a seguir para reparar la información de encaminamiento que detecta como problemática, utilizando las probabilidades que se muestran en la Tabla 3. Los valores de las probabilidades se encuentran normalizados y, por lo tanto, sus valores sumados resultan en uno. Es decir:

$$P_EGf + P_EGs + P_rand = 1 \tag{6.1}$$

En las simulaciones realizadas, la probabilidad normalizada de utilizar el método aleatorio toma uno de los siguientes valores: 0.2, 0.4 ó 0.6. La probabilidad normalizada de utilizar el método de algoritmo *Evolving Graph foremost* o *Evolving Graph shortest* es la mitad de la diferencia de uno menos la probabilidad aleatoria, es decir 0.4, 0.3 ó 0.2 respectivamente, esto es:

$$P_EGf = P_EGs = \frac{1 - P_rand}{2} \tag{6.2}$$

7. Resultados Experimentales

En todas las pruebas y simulaciones realizadas, el *Strength Pareto Evolutionary Algorithm* (SPEA) encontró siempre buenas soluciones, al un precio de un mayor tiempo de ejecución. La cantidad de buenas soluciones, las cuales compiten en calidad y, cuando es posible, superan a las soluciones calculadas por los otros algoritmos implementados en este trabajo, depende principalmente de dos factores. Por un lado, el tamaño constante de la población evolutiva *P*, y por el otro lado la cantidad de iteraciones I_k que efectúa el algoritmo, como será mostrado más adelante.

Las secciones que siguen, presentan resultados experimentales para las tres topologías de redes dinámicas periódicas que se muestran en la sección 6.1., específicamente en la Figuras 17, la Figura 18 y la Figura 19. Estos resultados son obtenidos utilizando los seis algoritmos nombrados a continuación:

1. *Destination-Sequenced Distance-Vector* (DSDV).
2. *Dynamic Source Routing* (DSR).
3. *Ad-Hoc Distance-Vector* (AODV).
4. *Evolving Graph Shortest* (EG s).
5. *Evolving Graph Foremost* (EG f).
6. *Strength Pareto Evolutionary Algorithm* (SPEA).

7.1. Simulaciones en Redes Pequeñas

Las simulaciones realizadas con la red dinámica periódica presentada en la Figura 17 dieron los resultados que se muestran en la Figura 20. Para permitir un fácil entendimiento al lector, la Figura 20 muestra el valor promedio de únicamente dos funciones objetivo: (a) cantidad de saltos y (b) cantidad de eras vividas. El valor de la tercera función objetivo (c) cantidad de eras viajadas, no se muestra para simplificar la comprensión, pero puede ser apreciado considerando el gran número de soluciones de compromiso encontradas por el

algoritmo *SPEA*, utilizando los parámetros presentados en la Tabla 3. La leyenda al lado derecho de la Figura 20 indica qué algoritmos fueron simulados.

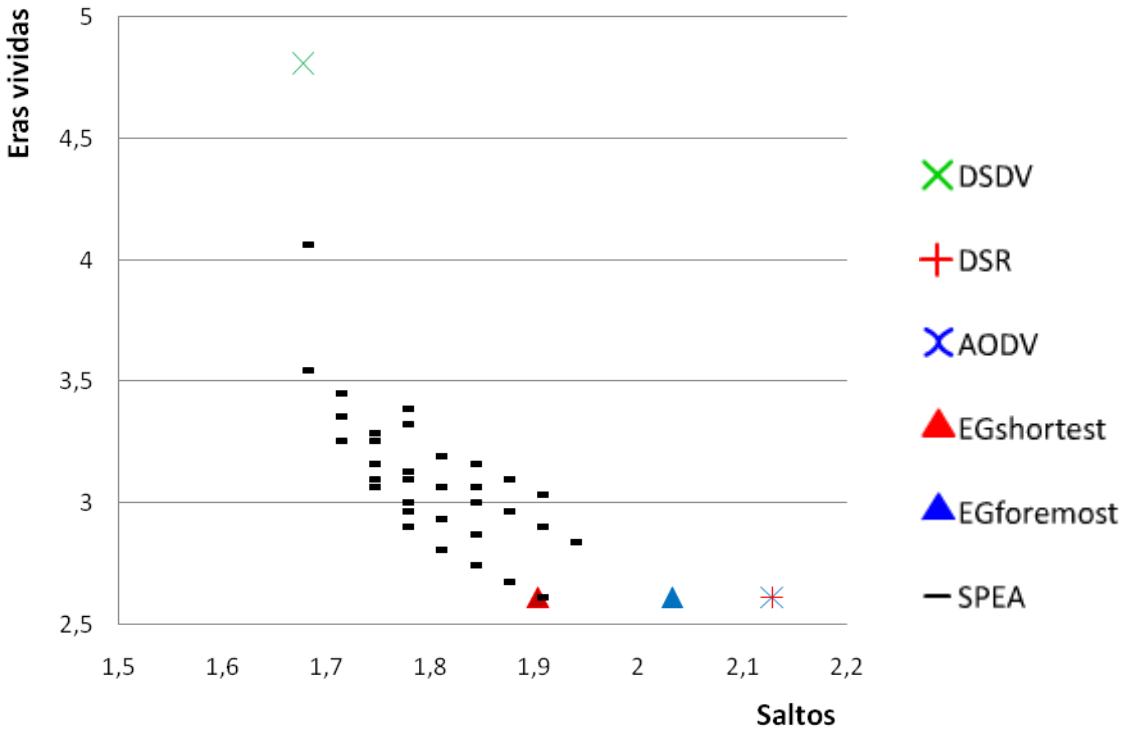


Figura 20. Soluciones para una red pequeña. Ejemplo de un conjunto de soluciones para la red pequeña de la Figura 17.

En la Tabla 4 se presenta un breve resumen de las tres funciones objetivo calculadas por el algoritmo *Evolving Graph Foremost*, *Evolving Graph Shortest* y dos soluciones elegidas de entre las 168 encontradas por el *Strength Pareto Evolutionary Algorithm* (SPEA). Como puede apreciarse en la columna de comentarios de la tabla, una solución calculada con el algoritmo *SPEA*, a la cual se nombra SPEA 1, domina a la solución encontrada por el algoritmo *Evolving Graph Foremost* debido a que ambos tienen el mismo valor promedio de eras de viaje hasta el nodo destinatario (cuyo valor óptimo es 2.61) pero la solución SPEA 1 tiene un menor valor promedio en las otras dos funciones objetivo que no fueron considerados por el algoritmo *EG Foremost*. Al mismo tiempo, otra solución calculada utilizando el *SPEA*, a la que se denota como SPEA 2, domina a la solución calculada por el algoritmo *Evolving Graph Shortest*; esto ocurre, porque ambos necesitan una cantidad promedio de 1.68 saltos para terminar el camino llegando al nodo destinatario, pero la solución SPEA 2 precisa, en promedio, una menor cantidad de eras para hacerlo (i.e. tiene mejores resultados en las funciones objetivo (b) y (c) antes descriptas).

Algoritmo	(a) Cantidad de saltos	(b) Cantidad de eras vividas	(c) Cantidad de eras viajadas	Comentario
EG Foremost	2.03	2.61	1.32	Dominado por SPEA 1
EG Shortest	1.68	4.07	1.16	Dominado por SPEA 2
SPEA 1	1.90	2.61	0.94	Domina a EG Foremost
SPEA 2	1.68	3.55	0.90	Domina a EG Shortest

Tabla 4. Valor promedio de las funciones objetivo calculadas para las soluciones obtenidas en la simulación utilizando la red pequeña mostrada en la Figura 17.

Cabe recordar que los algoritmos *Evolving Graph Shortest* y *Evolving Graph Foremost* calculan soluciones óptimas en un contexto mono-objetivo [18]; por lo tanto, es notable que el algoritmo *SPEA* pueda encontrar mejores soluciones que dichos algoritmos basados en grafos evolutivos, aun siendo éstos óptimos en el contexto mono-objetivo.

Otra ventaja del algoritmo *SPEA* propuesto en el presente trabajo, es su habilidad de hallar todo un conjunto de soluciones Pareto que pueden ser utilizadas para decidir cuál solución atiende mejor a las necesidades de encaminamiento de los usuarios. Como ejemplo, para la red dinámica periódica pequeña que fue utilizada en la comparación recién presentada, fueron encontradas 168 soluciones de compromiso (frente Pareto), como se indica en la Figura 20 en forma de guiones.

Por último, debe ser mencionado que las soluciones de los algoritmos *Evolving Graph Shortest* y *Evolving Graph Foremost* siempre dominan a las soluciones de los algoritmos *Destination-Sequenced Distance-Vector* (DSDV), *Dynamic Source Routing* (DSR) y *Ad-hoc On-Demand Distance Vector* (AODV), según se demuestra en [18], lo que resulta predecible dado que estos no fueron especialmente diseñados para redes dinámicas periódicas. Por otro lado, el algoritmo *Strength Pareto Evolutionary Algorithm* es capaz de calcular soluciones que a su vez dominan a las soluciones encontradas por todos los algoritmos antes citados. Este hecho es enfatizado en la siguiente sub-sección, donde los algoritmos nombrados son comparados en cada dominio de medición. De las 168 soluciones halladas por SPEA, 3 de ellas fueron seleccionadas para facilitar la comparación y son denotadas como SPEA 1, SPEA 2 y SPEA 3.

7.1.1. **Saltos Hasta Llegar a Destino**

Como es de esperarse, el algoritmo *Evolving Graph Shortest* encuentra siempre el camino más corto para entregar un paquete. La Figura 21 muestra la cantidad de saltos necesitados por cada uno de los treinta y un paquetes que conforman la definición de tráfico congestionado para

la red dinámica periódica, para viajar hasta el nodo destinatario. La definición de dicho tráfico se encuentra en la Tabla 2 y la red dinámica periódica pequeña en la Figura 17.

Si se considera únicamente la función objetivo referente a la cantidad de saltos, ninguna de las soluciones calculadas por los algoritmos nombrados en la leyenda de la Figura 21, se desempeñó mejor que la construida por el algoritmo *EG Shortest*. Esto es, porque dicho algoritmo es una adaptación del algoritmo de Dijkstra para Caminos de Costo Mínimo (ver Anexo I), el cual es óptimo [18].

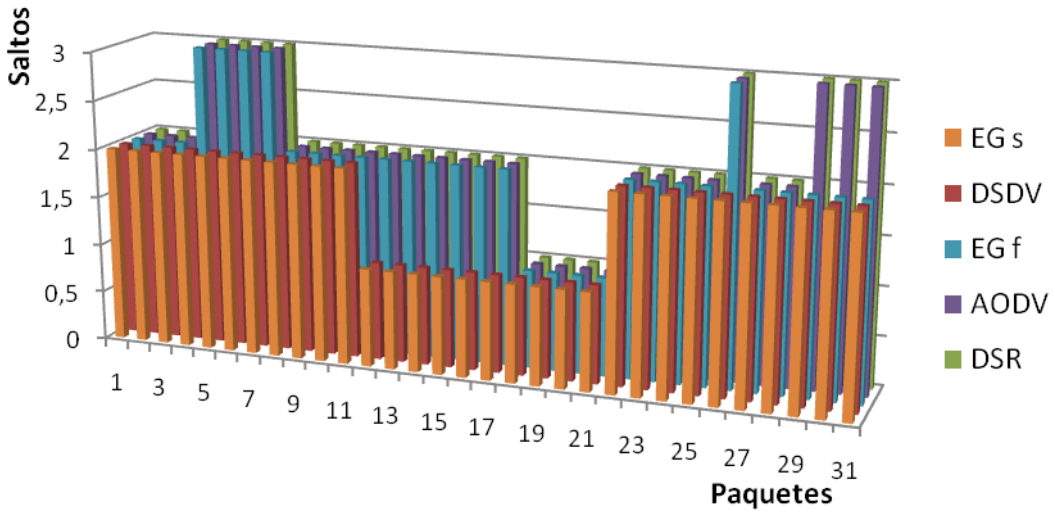


Figura 21. Saltos en el camino. Saltos realizados por cada paquete para arribar al nodo destinatario.

El algoritmo *Strength Pareto Evolutionary Algorithm* encontró una solución, a la que se denominó anteriormente SPEA 2, y que se desempeña tan bien como la solución calculada por el algoritmo *Evolving Graph Shortest* considerando la cantidad de saltos (ver Figura 22). De todas formas, como se muestra en la Tabla 4, la solución SPEA 2 se desempeña mejor que dicha solución *EG Shortest*, logrando que los viajes sean en promedio más rápidos y los paquetes vivan en promedio menor cantidad de eras. Por lo tanto, SPEA 2 es una solución mejor que la calculada por el algoritmo *EG Shortest*.

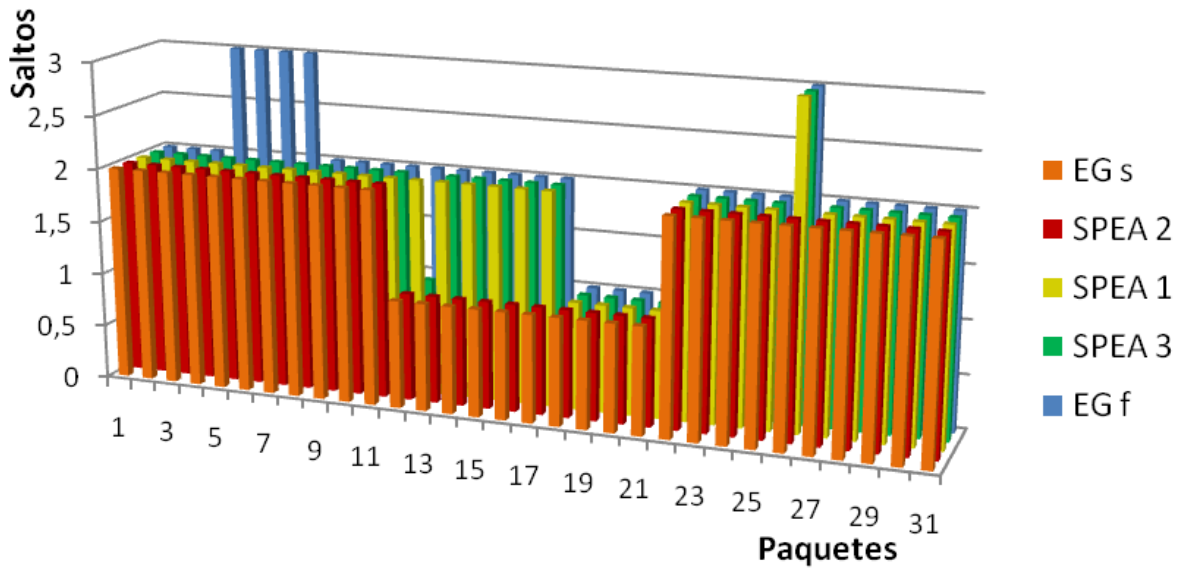


Figura 22. Saltos en el camino con *SPEA*. Saltos realizados por las soluciones del algoritmo *SPEA*.

7.1.2. **Eras Vividas Hasta Llegar a Destino**

Un paquete es creado cuando el nodo remitente tiene información preparada y suficiente para ser enviada hasta un nodo destinatario. La vida del paquete es considerada como el transcurso de eras entre que el nodo remitente crea el paquete y la era en que el mismo llega al nodo destinatario, donde los datos son extraídos y el paquete es borrado de memoria.

El algoritmo *Evolving Graph Foremost* es una adaptación del algoritmo de Dijkstra para Caminos de Costo Mínimo (ver Anexo I). En esta oportunidad, lo minimizado es la cantidad de eras que vive un paquete, en vez de la extensión de un camino [18]. Si se considera únicamente la función objetivo referente a la cantidad de eras vividas por los paquetes, el algoritmo *EG Foremost* brinda soluciones óptimas.

La Figura 23 muestra una comparación entre los cinco algoritmos mono-objetivo listados en la leyenda del gráfico. Como es de esperarse, el algoritmo *Evolving Graph Foremost* se desempeña mejor que los algoritmos *Evolving Graph Shortest* y *Destination-Sequenced Distance-Vector*. Resulta interesante notar que dicho algoritmo tiene un desempeño igual de bueno que los algoritmos *Dynamic Source Routing* y *Ad-hoc On-Demand Distance Vector* para la topología definida en la Figura 17 y el tráfico mostrado en la Tabla 2.

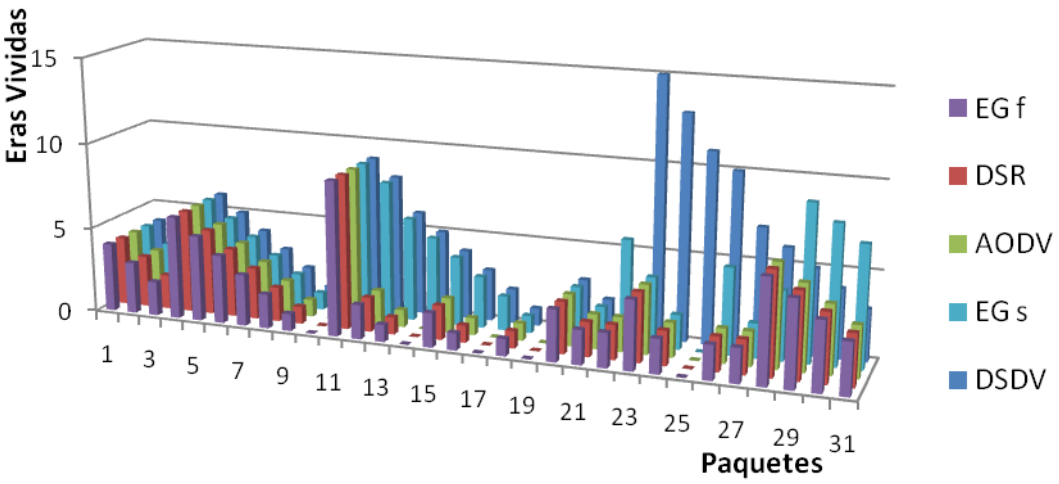


Figura 23. Eras vividas. Eras vividas por cada paquete hasta arribar al nodo destinatario.

El algoritmo *SPEA* encontró la solución que fue denominada SPEA 1, la cual se desempeña tan bien como la solución encontrada por el algoritmo *EG Foremost* cuando se considera únicamente el número promedio de eras vividas por los paquetes (ver Figura 24). De todas maneras, como se muestra en la Tabla 4, dicha solución SPEA 1 se desempeña mejor que la solución calculada por *EG Foremost* al entrar en consideración además las otras dos funciones objetivo. Por lo tanto, la solución SPEA 1 es mejor que las otras mostradas en la Figura 23.

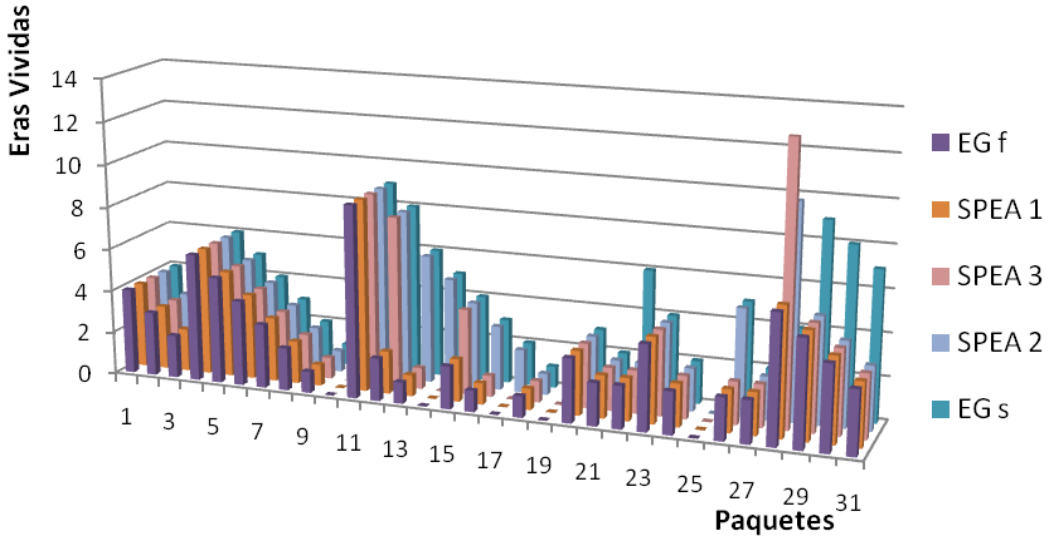


Figura 24. Eras vividas con *SPEA*. Eras vividas por cada paquete siendo encaminados con las soluciones del algoritmo *SPEA*.

7.1.3. **Eras Viajadas Hasta Llegar a Destino**

Un viaje es más rápido que otro si toma menor cantidad de eras atravesar el camino hasta el nodo destinatario. La cantidad de eras viajadas por un paquete es obtenida calculando la diferencia entre la era en que dicho paquete llegó al nodo destinatario, menos la era en que el paquete fue transmitido por el nodo remitente realizando el primer salto.

Considérese la función objetivo definida como (c) en la sección 5. Se sugiere en [18] la creación de un algoritmo mono-objetivo, el mismo sería una adaptación del algoritmo de Dijkstra para Caminos de Costo Mínimo (ver Anexo I) minimizando la cantidad de eras de viaje. Hasta el presente trabajo, dicho algoritmo mono-objetivo no fue publicado ni implementado. En efecto, conforme a lo expresado en sub-secciones anteriores, el supuesto algoritmo *Evolving Graph Fastest* encontraría los caminos óptimos para encaminar paquetes en la forma más rápida posible desde el nodo remitente hasta el nodo destinatario, en un ambiente mono-objetivo. Además, es de esperarse que en un contexto multi-objetivo, la solución calculada por el algoritmo *EG Fastest* sea dominada por soluciones encontradas por el algoritmo *SPEA*. Como mínimo, se espera que una solución del *SPEA* se desempeñe tan bien como la solución de *EG Fastest*.

Las tres soluciones, calculadas por el algoritmo *SPEA*, que se muestran en la figura 25 se desempeñan mejor que las soluciones calculadas por los algoritmos *EG Shortest*, *EG Foremost*, *DSDV*, *DSR* y *AODV* al considerar las eras viajadas.

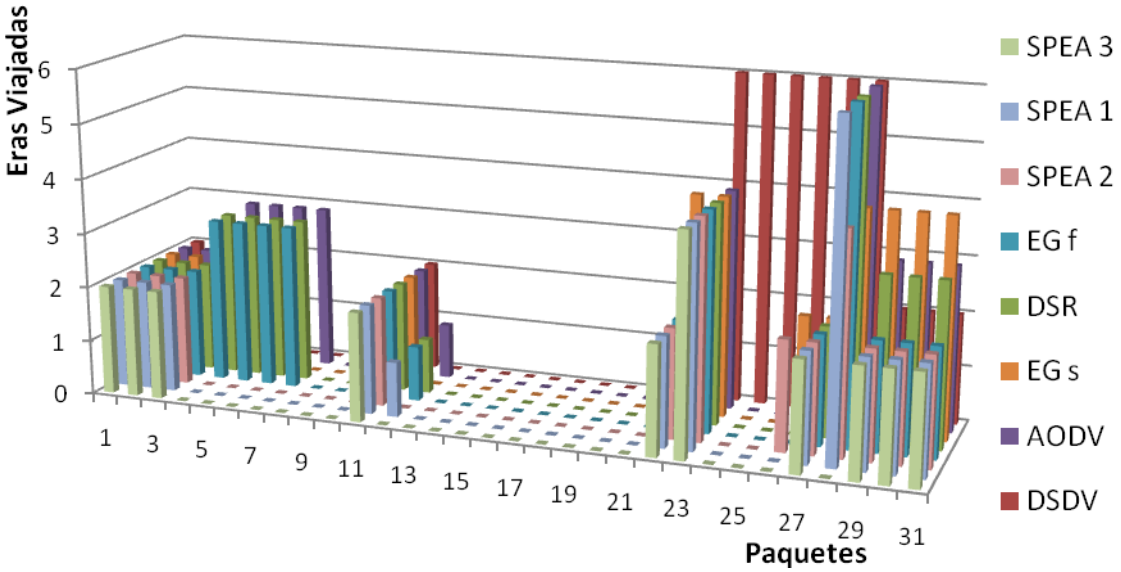


Figura 25. Eras viajadas. Eras viajadas por cada paquete hasta arribar al nodo destinatario.

7.2. Simulaciones en Redes Medianas

Simulaciones realizadas utilizando la red dinámica periódica de tamaño mediano (ver Figura 18), son resumidas en la Figura 26. En ella, el algoritmo *Strength Pareto Evolutionary Algorithm* asigna a los parámetros, los valores que se muestran en la Tabla 5.

Tamaño	P_{max}	I_{max}	P'_{max}	*EG inserción*	*P_EGf*	*P_EGs*	*P_rand*
Medium	10	1000	10000	1	0,3	0,3	0,4

Tabla 5. Ejemplo de valores asignados a los parámetros utilizados en el algoritmo *SPEA* para la red dinámica periódica mediana.

La Tabla 6 presenta el valor promedio que resulta de las tres funciones objetivo previamente definidas como (a), (b) y (c) en la sección 5. La primera columna de la tabla especifica el nombre del algoritmo que calculó la solución que se describe o bien, el identificador de la solución propiamente dicha en el caso del algoritmo *SPEA*. En el caso de este último algoritmo, 2 soluciones fueron seleccionadas de entre 53, y son denotadas como SPEA' 1 y SPEA' 2.

En esta oportunidad, la solución SPEA' 2 y la calculada por el algoritmo *Evolving Graph Foremost*, tienen los mismos valores en sus respectivas funciones objetivo es decir, encuentran la misma solución que no solo es óptima en eras vividas, sino también en el contexto multi-objetivo. Las soluciones SPEA' 1 y la propuesta por el algoritmo *Evolving Graph Shortest*, tienen la misma cantidad promedio de saltos en cada viaje (y es de 2.927), pero la solución SPEA' 1 logra en promedio que los viajes sean más rápidos que en la solución del algoritmo EG Shortest.

Algoritmo	*(a) Cantidad de saltos*	*(b) Cantidad de eras vividas*	*(c) Cantidad de eras viajadas*	*Comentario*
EG foremost	3.526	6.482	4.897	
EG shortest	2.927	10.667	7.449	Dominado por *SPEA' 1*
SPEA' 1	2.927	10.517	7.376	Domina a EG shortest
SPEA' 2	3.526	6.482	4.897	

Tabla 6. Valor promedio de las funciones objetivo calculadas para las soluciones obtenidas en la simulación utilizando la red mediana mostrada en la Figura 18.

La Figura 26 muestra a las soluciones calculadas por los algoritmos que se listan a la derecha del gráfico, en la leyenda. El conjunto de 53 soluciones no dominadas encontradas por el algoritmo *Strength Pareto Evolutionary Algorithm*, incluye a la solución calculada por el algoritmo *Evolving Graph Foremost* y además, incluye una solución diferente a la ya

nombrada, que domina a la solución calculada por el algoritmo *Evolving Graph Shortest*, superándolo una vez más (ver Tabla 6).

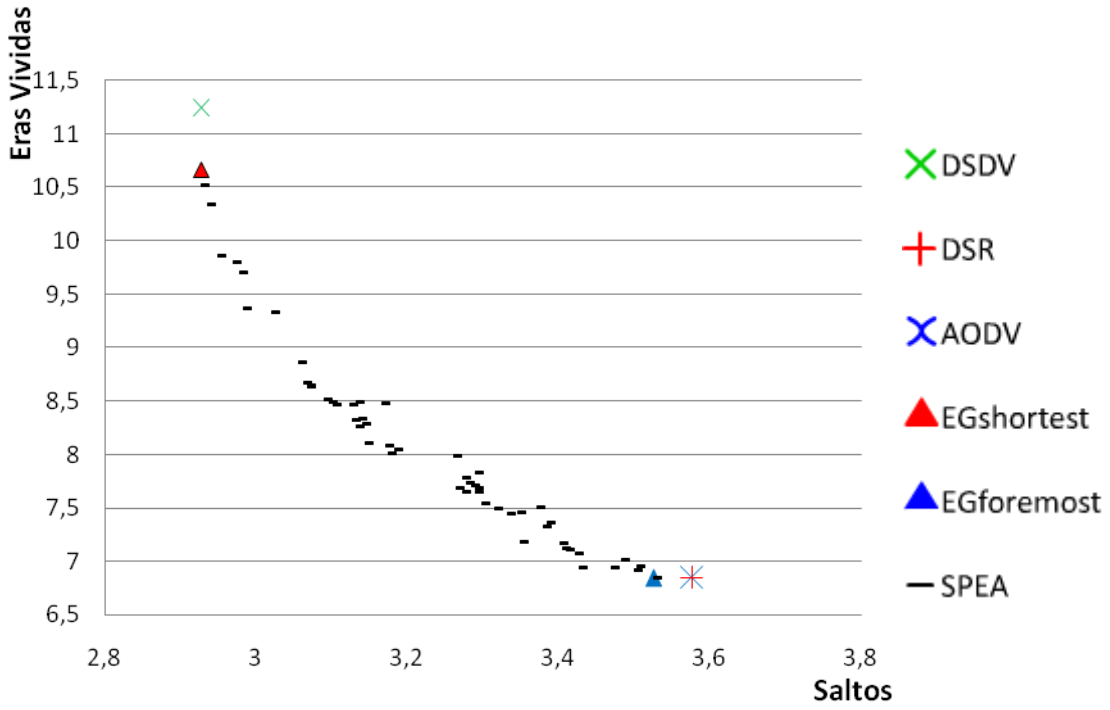


Figura 26. Soluciones para una red mediana. Ejemplo de un conjunto de soluciones para la red mediana de la Figura 18.

7.3. Simulaciones en Redes Grandes

Una red compuesta por veinticuatro nodos fue propuesta y presentada en la Figura 19. En la simulación que se describe, el algoritmo *Strength Pareto Evolutionary Algorithm* utiliza los valores expuestos en la Tabla 7 en sus parámetros. El tráfico de datos considerado está formado por 630 paquetes. Los resultados experimentales son presentados en la Figura 27.

Size	P_{max}	I_{max}	P'_{max}	*EG inserción*	*P_EGf*	*P_EGs*	*P_rand*
Large	10	100	10000	1	0,3	0,3	0,4

Tabla 7. Ejemplo de valores asignados a los parámetros utilizados en el algoritmo *SPEA* para la red dinámica periódica grande.

La solución calculada por el algoritmo *Evolving Graph Shortest* tuvo un desempeño de cantidad promedio de saltos que necesitan los paquetes para llegar al nodo destinatario, igualmente bueno al alcanzado por la solución propuesta por el algoritmo *Destination-Sequenced Distance-Vector*. La solución presentada por el algoritmo *Evolving Graph Foremost* sobrepasa en desempaño a las soluciones propuestas por los algoritmos *Dynamic Source*

Routing y *Ad-hoc On-Demand Distance Vector*, logrando que en promedio, los paquetes vivan una menor cantidad de eras hasta llegar al nodo destinatario.

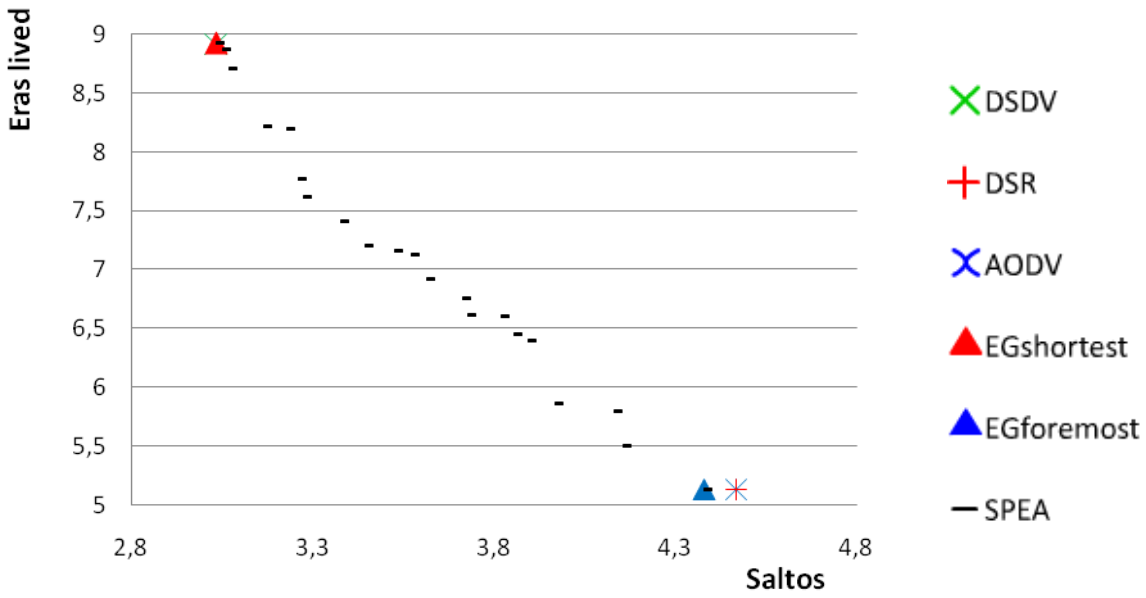


Figura 27. Soluciones para una red grande. Ejemplo de un conjunto de soluciones para la red grande de la Figura 19.

En la presente simulación, el algoritmo *SPEA* logró encontrar 22 soluciones no dominadas. A continuación, se detalla el desempeño de 3 soluciones distintas que pertenecen al conjunto de soluciones recién citadas. Las 3 soluciones son denotadas por SPEA'' 1, SPEA'' 2 y SPEA'' 3.

Algoritmo	***(a) Cantidad de saltos***	***(b) Cantidad de eras vividas***	***(c) Cantidad de eras viajadas***
EG foremost	4.379	0.548	4.897
EG shortest	3.033	8.927	3.27
SPEA'' 1	3.033	8.927	3.27
SPEA'' 2	4.379	0.548	4.897
SPEA'' 3	3.443	1.825	7.203

Tabla 8. Valor promedio de las funciones objetivo calculadas para las soluciones obtenidas en la simulación utilizando la red grande mostrada en la Figura 19.

Los valores promedio de las funciones objetivo se encuentran resumidas en la Tabla 8. El conjunto de soluciones calculado por el algoritmo *SPEA* incluye a aquellas soluciones que fueron propuestas por los algoritmos *Evolving Graph Shortest* y *Evolving Graph Foremost*; específicamente, estas soluciones son SPEA'' 1 y SPEA'' 2. Dicho conjunto de soluciones también incluye a la solución SPEA'' 3, que tiene una relación de compromiso entre las tres

funciones objetivo: (a) cantidad de saltos, (b) cantidad de eras vividas y por último, (c) cantidad de eras viajadas.

7.4. Variación de los Parámetros del Algoritmo SPEA

Las observaciones realizadas en la presente sección, están basadas en pruebas muy extensas en las que se utiliza la red dinámica periódica considerada de tamaño pequeño y definida en la Figura 17, además del tráfico compuesto por treinta y un paquetes especificado en la Tabla 2. Los resultados experimentales son mostrados en la Tabla 9, para diferentes combinaciones de valores asignados a los parámetros.

La Tabla 9 es similar a las Tablas 3, 5 y 7, pero a diferencia de estas, incluye algunas columnas adicionales que se definen a continuación. La columna denominada **Número de Soluciones**, indica el número de individuos que componen el frente Pareto calculado por el algoritmo *Strength Pareto Evolutionary Algorithm* luego de haber realizado I_{max} iteraciones (i.e. la población P evolucionó I_{max} veces). La columna llamada **Soluciones No Dominadas** contiene el número de soluciones encontradas por el algoritmo *Strength Pareto Evolutionary Algorithm* que no son dominadas por ninguna otra solución calculada por los otros algoritmos simulados: *DSDV*, *DSR*, *AODV*, *EG Shortest* y *EG Foremost*.

Las columnas **Domina a EGf** y **Domina a EGs** indican si existe al menos una solución en el conjunto de soluciones calculadas por el algoritmo *SPEA*, que se desempeña mejor y domina a las soluciones propuestas por los algoritmos *EG Foremost* y *EG Shortest*, respectivamente.

Para facilitar la referencia, cada instancia de experimento es referida por el índice de la Tabla 9, que se encuentra a la izquierda, en la primera columna. Un conjunto de instancias indexadas por la primera columna, se expresan en forma de una lista encerrada entre llaves. A modo de ejemplo, las instancias 7, 8 y 9 son referenciadas como {7, 8, 9}.

Finalmente, cabe mencionarse que el número de **soluciones** encontradas por el algoritmo *SPEA* en sus iteraciones sucesivas, no asegura la calidad de dichas soluciones. Por ejemplo, utilizando las funciones objetivo (a), (b) y (c) definidas en la sección 5, dos soluciones ineficientes pueden presentarse como parte del frente Pareto, y un conjunto de muchas soluciones pueden estar distribuidas en el frente entre ambas soluciones ineficientes. En contrapartida, la cantidad de **soluciones no dominadas** expresa en forma fiel el desempeño del algoritmo encontrando buenas soluciones; esto es, porque las soluciones no dominadas son parte del frente Pareto que se forma considerando, entre otras, a las soluciones provistas por los

algoritmos *EG Shortest*, *EG Foremost*, *DSDV*, *DSR* y *AODV*. Se expresó en secciones anteriores, que los algoritmos *EG Shortest* y *EG Foremost* son óptimos si se considera solamente la función objetivo (a) para el primero y la (b) para el segundo, en un contexto mono-objetivo.

Nº	P_{max}	I_{max}	EG inserción	P_EGf	P_EGs	P_rand	Número de Soluciones	Soluciones No Dominadas	Domina a EGf	Domina a EGs
1	20	100	0	0,3	0,3	0,4	15	1	No	No
2	10	100	0	0,4	0,4	0,2	2	0	No	No
3	10	1000	0	0,4	0,4	0,2	51	49	No	No
4	10	10000	0	0,4	0,4	0,2	9999	2105	No	No
5	20	100	0	0,4	0,4	0,2	12	0	No	No
6	50	100	0	0,4	0,4	0,2	11	8	No	No
7	10	100	0	0,2	0,2	0,6	3	0	No	No
8	20	100	0	0,2	0,2	0,6	12	1	No	No
9	50	100	0	0,2	0,2	0,6	10	10	No	No
10	10	100	1	0,3	0,3	0,4	82	22	Yes	Yes
11	20	100	1	0,3	0,3	0,4	106	25	Yes	Yes
12	50	100	1	0,3	0,3	0,4	167	24	Yes	Yes
13	10	100	50	0,3	0,3	0,4	77	32	Yes	No
14	20	100	50	0,4	0,4	0,2	86	21	No	Yes
15	10	100	1	0,4	0,4	0,2	94	15	Yes	No
16	10	100	1	0,2	0,2	0,6	88	19	Yes	Yes
17	50	100	1	0,4	0,4	0,2	148	56	Yes	Yes
18	50	100	1	0,2	0,2	0,6	142	40	No	No

Tabla 9. Simulaciones en una red pequeña. Simulaciones realizadas utilizando la red dinámica periódica de la Figura 17 y el tráfico de la Tabla 2.

7.4.1. **Tamaño de la Población**

Cuando se mantienen constantes todos los otros parámetros, incrementar el tamaño de la población tiene el efecto de mejorar los resultados de la exploración y, por lo tanto, una mayor cantidad de soluciones no dominadas pueden ser encontradas. Como ejemplo, considérese el conjunto de instancias {7, 8, 9} de la Tabla 9, donde el *Strength Pareto Evolutionary Algorithm* varía el tamaño de la población evolutiva y ello tiene el efecto de mejorar notablemente la calidad de las soluciones halladas. Además, en {9} el frente Pareto está formado por una menor cantidad de soluciones que en {8} y sin embargo, la calidad de sus soluciones es mayor. Esto puede ser visto a través de la enorme mejoría en el número de soluciones no dominadas.

La magnitud de la mejoría que introduce el aumento de la población evolutiva, en las soluciones no dominadas, disminuye cuanto más cerca se encuentra el valor del parámetro *EG inserción* a la cantidad máxima de iteraciones I_{max}. El menor desempeño es visto cuando se asigna el valor cero al parámetro *EG inserción*. La mejoría máxima es vista cuando *EG inserción* tiene un valor pequeño como uno, como se ve con las instancias {10, 11, 12}, o

cuando la población externa incluye a soluciones que dominan a las propuestas por los algoritmos *Evolving Graph Shortest* o *Evolving Graph Foremost*.

7.4.2. **Cantidad de Iteraciones**

Incrementar el número de iteraciones que el algoritmo realiza, implica que la población *P* evoluciona un mayor número de veces (ver Algoritmo 5). Esto a su vez, tiene influencia directa en el número de *soluciones no dominadas*, como puede verse en el conjunto {2, 3, 4}. Por consiguiente, incrementar el número de iteraciones tiene como consecuencia un efecto similar al de incrementar el tamaño de la población (ver sección 7.5.) en la columna de *soluciones no dominadas* de la Tabla 9. Esto resulta muy natural dado que a la población externa *P'* únicamente ingresan soluciones no dominadas que son tan buenas o mejores que las contenidas previamente en dicha población externa. Es decir que la población externa *P'* contiene individuos cada vez mejores o a lo sumo igual de buenos, de acuerdo va evolucionando la población *P*, pero de ninguna forma dichas soluciones pertenecientes a *P'* pueden empeorar.

7.4.3. **Probabilidades en el Reformatorio**

En general, las simulaciones hacen uso de una misma probabilidad para los parámetros *P_EGf* y *P_EGs* de forma a evitar posibles sesgos. Los algoritmos mono-objetivo *Evolving Graph Shortest* y *Evolving Graph Foremost* utilizan una función objetivo cada uno, las cuales están definidas en la sección 5.1 como (a) y (b) respectivamente. Es igualmente deseable que los caminos sean cortos y que los paquetes lleguen al nodo destinatario en la era más temprana posible, y al asignar igual probabilidad a dichos parámetros *P_EGf* y *P_EGs*, se evita favorecer a una característica por sobre la otra.

La mayor parte de las veces, los mejores resultados experimentales fueron obtenidos cuando tanto *P_EGf* como *P_EGs* tomaban el valor de 0.3, y al parámetro *P_rand* se asignaba 0.4. Esto puede ser apreciado cuando se consideran los resultados de los experimentos referidos en {10, 15, 16} y en {1, 5, 8}.

Por último, puede notarse que no siempre la mejor distribución es la sugerida anteriormente (*P_EGf* = 0.3, *P_EGs* = 0.3 y P_rand = 0.4), un ejemplo de ello son las soluciones que presenta la Tabla 9 para el conjunto de simulaciones {12, 17, 18}. En efecto, puede notarse que para dicho conjunto de simulaciones, la distribución de probabilidades utilizada en {17} dio el mejor resultado.

8. Conclusiones y Trabajos Futuros

Como fue identificado en el presente trabajo, las redes dinámicas son de gran utilidad para modelar el comportamiento periódico de algunas aplicaciones existentes. Como por ejemplo, las redes de sensores, los satélites de órbita baja o por sus nombres en inglés *Low Orbit Satellite* y *Low Earth Orbit* (LEO). Además, las redes dinámicas logran modelar redes del tipo ad-hoc y otros sistemas de comunicación. Resulta de gran utilidad estudiar dichas redes haciendo uso de los Grafos Evolutivos (*Evolving Graphs* - EG), como se demuestra en [18].

Hasta el momento, el problema de encaminar paquetes en redes dinámicas periódicas encarado en la bibliografía publicada, ha sido estudiado considerando una única función objetivo como: (a) la cantidad promedio de saltos efectuados por los paquetes para llegar al nodo destinatario; (b) la cantidad promedio de eras vividas por los paquetes hasta que todos ellos culminan su camino; (c) la cantidad promedio de eras durante las cuales los paquetes viajan a través de la red hasta llegar a destino. El presente trabajo modela, por primera vez, el problema de encaminamiento en redes dinámicas periódicas en el contexto de un Problema Multi-Objetivo (en inglés Multi-Objective Problem - MOP).

Para resolver este problema multi criterio, se desarrolla un Algoritmo Evolutivo Multi-Objetivo, en inglés *Multi-Objective Evolutionary Algorithm* (MOEA), inspirado en el algoritmo *Strength Pareto Evolutionary Algorithm* (SPEA). Dicho algoritmo es especialmente adaptado en este trabajo para satisfacer las necesidades específicas del problema de ruteo. Para ello, se desarrolla un procedimiento especializado denominado *Reformatorio*, descripto en la sección 4.2, que convierte soluciones no factibles en tablas de encaminamiento que permiten encaminar todos los paquetes que puedan ser generados.

De esta forma y con el enfoque propuesto, es posible hallar un conjunto de soluciones Pareto que no sólo contienen individuos competitivos con las soluciones calculadas por otros algoritmos que forman parte del estado del arte, sino que además, se logra calcular individuos que dominan a las soluciones consideradas como óptimas cuando una sola función objetivo es considerada. Esto se debe a que el enfoque aquí propuesto es capaz de mejorar, en un contexto multi-objetivo, a las soluciones que pueden ser consideradas como óptimas en un entorno mono-objetivo al considerar los demás objetivos no contemplados por un algoritmo mono-objetivo. Además, es importante resaltar las ventajas de calcular todo un conjunto de soluciones en conflicto en solo una corrida del algoritmo, dando al tomador de decisiones la habilidad de

escoger la alternativa que mejor satisfaga a las necesidades del problema, conociendo sus alternativas (lo que gana y lo que pierde) antes de tomar sus decisiones.

En resumen, se puede decir que el principal aporte de este trabajo es la posibilidad de optimizar en un contexto multi-objetivo, tablas de ruteo hasta ahora consideradas óptimas ante la inexistencia de publicaciones que consideren el contexto multi-objetivo propuesto en este trabajo.

Trabajos futuros pueden incluir desviaciones en el comportamiento planificado de cada nodo perteneciente a la red dinámica periódica, utilizando un grafo evolutivo como guía o estimación de la topología y comportamiento de la red en cada era. De esta forma, los nodos deberían incluir algún mecanismo de control que detecte las diferencias entre la topología que sugiere el grafo evolutivo, y la topología real en el instante en que se quiera enviar un paquete. Las diferencias entre la red planificada y la real pueden deberse, entre otras cosas, a efectos climáticos repentinos que impidan la concreción de un enlace.

La calidad de servicio, en inglés *Quality of Service* (QoS) es una posible área de trabajo a ser considerada. Desde que muchos servicios utilizan como plataforma de interconexión a diferentes tipos de redes, deben poder asegurarse rangos aceptables o límites de la cantidad y calidad de la transmisión de paquetes a través de la red, del tiempo que toma su traslado, y la disponibilidad de recursos en los nodos intermedios. Además, con el crecimiento de las redes y del uso que se les da como plataforma para la transmisión de datos, interesa contar con alternativas para evitar la congestión y proponer un mecanismo de control de flujo.

Cabe recordar que este trabajo consideró que todas las eras duran un mismo intervalo de tiempo, por lo que una extensión natural sería asignar un tiempo distinto de duración a cada era.

9. Referencias

[1] Ash, G. Dynamic Network Evolution, with Examples from AT&TS Evolving Dynamic Network. 1995.

[2] Bahi, J. M., Couturier, R. y Vernier, F. “Synchronous Distributed Load Balancing on Totally Dynamic Networks”, 2007.

[3] Bhadra, S. y Ferreira, A. Complexity of Connected Components in Evolving Graphs and the Computation of Multicast Trees in Dynamic Networks. In Proceedings of Adhoc-Now’03, volume 2865 of Lecture Notes in Computer Science, pág. 259–270. Springer Verlag, Octubre 2003.

[4] Bui Xuan, B., Ferreira, A. y Jarry, A., “Computing Shortest, Fastest, and Foremost Journeys in Dynamic Networks”, 2004.

[5] Burns, B., Brock, O., Levine, B. N. “MV Routing and Capacity Building in Disruption Tolerant Networks”, Department of Computer Science, University of Massachusetts Amherst, 2005.

[6] Campanha, R., Bletsas, M., Claudio, L. y Magalhães, S. “Mesh Networks for Digital Inclusion - Testing OLPC's XO Mesh Implementation”, 2007.

[7] Corson, S. y Macker, J. Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. RFC 2501, IETF, Enero 1999.

[8] Ferreira, A., Galtier, J. y Penna, P. Topological design, routing and hand-over in satellite networks. In I. Stojmenovic, editor, Handbook of Wireless Networks and Mobile Computing, pág. 473–493. John Wiley and Sons, New York, NY, USA, 2002.

[9] Ford, L. R. y Fulkerson, D. R. “Constructing Maximal Dynamic Flow from Static Flows”. The Journal of the Operations Research Society of America, 6:419–433, 1958.

[10] Hanbin, L., Mingyi, W., Shaowen, Y. y Mingtian, Z. “Routing in dynamic network based on the fuzzy knowledge exchange among multi agents”, 2003.

[11] Jarry, A. y Lotker, Z. “Connectivity in Evolving Graph with Geometric Properties”, 2004.

[12] Johnson, D., Maltz, D. Dynamic Source Routing in Ad Hoc Wireless Networks. Imielinski and Korth, editors, Mobile Computing, vol. 353, cap. 5, pág. 153–181. Kluwer Academic Publishers, 1996.

[13] Kastner, W. y Leupold, M., "How Dynamic Network works: A Short Tutorial on Spontaneous Networks", 2001.

[14] Köhler, E., Langkau, K. y Skutella, M. "Time-Expanded Graphs for Flow-Dependent Transit Times". Proceedings of the 10th annual European Symposium on Algorithms (ESA'02), pages 599–611, Londres, Gran Bretaña, 2002.

[15] Mahesh. M. "AODV code for CMU Wireless and Mobility Extensions to NS2". http://www.cs.sunysb.edu/ mahesh/aodv/.

[16] Martínez, F. J. Un Nuevo Enfoque para Resolver Problemas de Optimización Multi-Objetivo Utilizando Programación Genética Lineal. Tesis de grado, Universidad Católica "Nuestra Señora de la Asunción". 2008.

[17] Mendel, G. Experiments in plant Hybridization. Expuesto en las reuniones del Brünn Natural History Society con fechas 8 de febrero y 8 de marzo de 1865.

[18] Monteiro, J. "Uso de Grafos Evolutivos no Roteamento em Redes Dinâmicas: algoritmos, fluxos e limites". Tesis de maestría. Instituto de Matemática y Estadística, Universidad de São Paulo. 2007.

[19] Monteiro, J., Goldman, A. y Ferreira, A. Performance Evaluation of Dynamic Networks using an Evolving Graph Combinatorial Model. In Proceedings of the 2nd IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob'06), pág. 173–180, Montreal, CA, enero 2006. Best Student Paper Award.

[20] Perkins, C y Bhagwat, P. "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers". Proceedings of the Conference on Communications Architectures, Protocols and Applications (ACM SIGCOMM'94), pág. 234–244, Sep 1994.

[21] Rice University Monarch Project. The CMU Monarch Wireless and Mobility Extensions to NS2. http://www.monarch.cs.rice.edu/.

[22] Sampedro, J. Deconstruyendo a Darwin. Editorial Crítica. Año 2002. ISBN 978-84-8432-910-7.

[23] Siqueira, I. G., Ruiz, L. B., Loureiro, A. F. y Nogueira, J. M. Coverage area management for wireless sensor networks. International Journal of Network Management. 17(1):17–31, Año 2007.

[24] Stojmenovic, I., Nayak, A. y Kuruvila, J. Design guidelines for routing protocols in ad-hoc and sensor networks with a realistic physical layer. IEEE Communications Magazine (Ad Hoc and Sensor Networks Series), 43(3):101–106, Marzo 2005.

[25] Tanenbaum, A. S. Computer Networks. Prentice-Hall. 4th Edition. ISBN-13: 9780130661029. 2003.

[26] Tomar, G.S., "Modified Routing Algorithm for AODV in Constrained Conditions", Modeling & Simulation, 2008. AICMS 08. Second Asia International Conference on Volume , Issue , 13-15 Mayo 2008 pág (s):219 – 224 Digital Object Identifier 10.1109/AMS.2008.196.

[27] von Lücken, C., "Algoritmos Evolutivos para Optimización Multiobjetivo: un estudio comparativo en un ambiente paralelo asíncrono", Tesis de maestría, Universidad Nacional de Asunción, 2003.

[28] Zhang, Z. Routing in Intermittently Connected Mobile Ad Hoc Networks and Delay Tolerant Networks: Overview and Challenges. San Diego Research Center. IEEE Communications Survey. 1st. Quarter 2006, Vol. 8, Nº 1.

[29] Zitzler, E. y Thiele, L. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. IEEE Transactions on Evolutionary Computation, 3(4):257–271, 1999.

Anexo I. Dijkstra Shortest Path Algorithm

```
/*

Variables:

G=(V,A): un grafo dirigido
a: vértice origen
z: vértice destino
C: conjunto de vértices todavía no cerrados
D: vector que almacena las distancias desde a hasta otros vértices
P: vector que contiene el vértice que precede a cada nodo en el camino

*/

   1. C ← V
   2. Para todo vértice i ∈ C, i ≠ a, se establece Di ← ∞; Da ← 0
   3. Para todo vértice i ∈ C se establece Pi = a
   4. Se obtiene el vértice s ∈ C tal que ¬∃ w ∈ C tal que Dw < Ds
      Si s = z entonces se ha terminado el algoritmo
   5. Se elimina de C el vértice s: C ← C-{s}
   6. Para cada arista e ∈ A de longitud l, que une el vértice s con algún
      otro vértice t∈C, hacer:
         a. Si l+Ds < Dt, entonces:
            • Se establece Dt ← l+Ds
            • Se establece Pt ← s
         b. Fin
   7. Fin
   8. Se regresa a 4.
```

Puede encontrarse un ejemplo gráfico dinámico en:

http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/DijkstraApp.shtml?demo3

Anexo II. Tráfico pesado o congestionado para la red mediana de la Figura 18

Remi-tente	Desti-natario	Era
A	2a	1
A	2a	3
A	2a	5
A	2a	7
A	2a	9
A	2b	1
A	2b	2
A	2b	3
A	2b	4
A	2b	5
A	2c	6
A	2c	8
A	2c	10
A	2c	12
A	2c	14
2a	A	1
2a	A	3
2a	A	5
2a	A	7
2a	A	9
2b	A	1
2b	A	2
2b	A	3
2b	A	4
2b	A	5
2c	A	6
2c	A	8
2c	A	10
2c	A	12
2c	A	14
D	2a	4
D	2a	5
D	2a	6
D	2a	7
D	2a	8
D	2b	9
D	2b	10
D	2b	11
1b	E	8
1c	E	9

Remi-tente	Desti-natario	Era
D	2b	12
D	2b	13
D	2c	14
D	2c	15
D	2c	1
D	2c	2
D	2c	3
2a	D	4
2a	D	5
2a	D	6
2a	D	7
2a	D	8
2b	D	9
2b	D	10
2b	D	11
2b	D	12
2b	D	13
2c	D	14
2c	D	15
2c	D	1
2c	D	2
2c	D	3
C	1a	5
C	1a	7
C	1a	9
C	1a	11
C	1a	1
C	1b	3
C	1b	5
C	1b	7
C	1b	9
C	1b	1
C	1c	2
C	1c	3
C	1c	6
C	1c	9
C	1c	15
1a	C	5
1d	A	8
1d	B	10

Remi-tente	Desti-natario	Era
1a	C	7
1a	C	9
1a	C	11
1a	C	1
1b	C	3
1b	C	5
1b	C	7
1b	C	9
1b	C	1
1c	C	2
1c	C	3
1c	C	6
1c	C	9
1c	C	15
E	1a	2
E	1a	4
E	1a	6
E	1a	8
E	1a	10
E	1b	12
E	1b	14
E	1b	6
E	1b	7
E	1b	8
E	1c	9
E	1c	10
E	1c	2
E	1c	6
E	1c	8
1a	E	2
1a	E	4
1a	E	6
1a	E	8
1a	E	10
1b	E	12
1b	E	14
1b	E	6
1b	E	7
2d	E	4
A	2d	5

1c	E	10
1c	E	2
1c	E	6
1c	E	8
1a	A	1
1a	A	2
1a	A	3
1a	A	4
1a	A	5
1a	B	6
1a	B	7
1a	B	8
1a	B	9
1a	B	10
1a	C	11
1a	C	12
1a	C	13
1a	C	14
1a	C	15
A	1a	1
A	1a	2
A	1a	3
A	1a	4
A	1a	5
B	1a	6
B	1a	7
B	1a	8
B	1a	9
B	1a	10
C	1a	11
C	1a	12
C	1a	13
C	1a	14
C	1a	15
1d	A	2
1d	A	4
1d	A	5
1d	A	7

1d	B	11
1d	B	13
1d	B	1
1d	B	2
1d	C	4
1d	C	5
1d	C	7
1d	C	8
1d	C	10
A	1d	2
A	1d	4
A	1d	5
A	1d	7
A	1d	8
A	1d	10
A	1d	11
A	1d	13
A	1d	1
A	1d	2
A	1d	4
A	1d	5
A	1d	7
A	1d	8
A	1d	10
2d	A	5
2d	A	6
2d	A	7
2d	A	8
2d	A	9
2d	D	1
2d	D	2
2d	D	3
2d	D	4
2d	D	5
2d	E	12
2d	E	13
2d	E	14
2d	E	15

A	2d	6
A	2d	7
A	2d	8
A	2d	9
D	2d	1
D	2d	2
D	2d	3
D	2d	4
D	2d	5
E	2d	12
E	2d	13
E	2d	14
E	2d	15
E	2d	4
2c	A	11
2c	A	13
2c	A	1
2c	A	2
2c	A	3
2c	D	6
2c	D	11
2c	D	2
2c	D	4
2c	D	8
2c	E	9
2c	E	12
2c	E	14
2c	E	15
2c	E	2
1c	2b	1
1c	2b	2
1c	2b	6
1c	2b	7
1c	2b	13
1c	2b	1
1c	2b	2
1c	2b	6
1c	2b	7

Anexo III. Tráfico pesado o congestionado para la red grande de la Figura 19

Remi-tente	Desti-natario	Era
3a	4d	8
1c	4b	1
C	1b	12
2a	A	11
4a	2a	8
2a	2c	5
4a	1a	2
1c	C	9
4b	1d	5
3a	B	3
3d	D	12
3c	1b	2
1d	4d	11
A	4b	7
1c	1d	2
2b	4c	5
C	3d	14
2d	4b	3
4a	3a	10
2d	4b	3
1c	4d	5
4c	3b	11
1d	3c	9
A	3d	2
2d	1b	12
4b	2d	8
D	3b	1
2a	1a	8
2d	4c	10
4d	2d	12
1d	4b	12
3c	3d	14
2a	3b	5
2d	C	2
3d	2b	10
3b	1d	2
3a	1d	13
1d	3a	9
1a	3b	10
2a	3b	14

Remi-tente	Desti-natario	Era
4d	4b	8
3c	2a	8
4b	4c	10
3a	2b	5
3a	2a	5
1a	B	10
B	2d	12
4d	D	14
3b	1a	12
4c	D	6
1a	1b	9
3a	B	9
C	1c	14
3b	B	8
C	2b	6
1d	3d	9
1d	4c	14
4b	1c	8
4c	4a	13
1b	B	13
2b	D	1
2a	1d	13
2a	B	7
2a	B	7
3b	4c	9
3d	3b	7
3c	2a	1
C	2b	1
2b	3c	1
1b	4c	14
2b	1a	5
A	D	2
2b	2a	14
1c	4c	10
3d	3b	11
4a	A	6
1d	D	3
4d	3c	3
D	4b	2
4a	1b	9

Remi-tente	Desti-natario	Era
4a	1b	9
1a	4d	4
4b	4d	6
1a	2c	13
B	4b	7
2b	3d	8
4d	2a	8
4c	1b	4
2b	C	4
3a	1c	5
2d	D	1
2c	B	6
B	4c	1
2b	3c	10
D	3b	9
4a	3c	8
2c	2d	3
4b	1a	5
3d	4d	12
C	3c	12
2b	3d	11
3a	4c	14
3a	2c	8
D	4d	10
B	4c	4
2b	4b	11
2c	2b	8
B	3b	7
4d	2d	10
1d	3a	9
1a	C	4
A	D	1
1a	3a	13
1b	2d	6
4d	1a	12
4a	D	7
4c	A	11
4d	2c	1
3d	4a	14
A	D	11

1a	2d	11
3b	3a	3
2c	2b	4
1d	1b	13
B	4d	4
1c	2b	9
2c	1a	7
3d	C	7
1d	1a	11
4a	2b	12
2c	2b	2
D	2b	5
2a	B	7
4d	2d	14
2d	3a	5
2a	4d	12
4c	1a	13
3b	C	9
3a	4d	7
2c	4a	3
D	B	4
1b	D	13
4b	D	1
1b	2a	8
3c	1c	1
D	C	11
A	3a	4
3b	2d	11
4b	1c	10
1a	4b	10
3a	2c	11
4a	D	13
1c	4a	14
3d	1b	2
B	2c	10
2d	D	6
1b	1a	10
2d	B	5
2c	3a	8
1d	2b	4
4d	D	12
4a	B	13
4c	3d	13
1a	1b	9
3c	1a	7
4c	3b	13
2d	3c	9
D	2d	13

2a	4d	8
3b	4c	8
1b	4a	5
1d	3b	3
3a	2c	4
A	4b	2
1a	A	10
1a	1c	10
3c	4c	14
3b	A	7
3a	D	1
2a	2c	9
4b	1d	8
1a	4d	1
2a	4c	12
2b	2a	13
1b	C	3
1c	B	13
4a	2c	12
2b	2a	10
B	1b	4
A	2a	9
C	3c	11
1d	4b	7
2d	4d	13
3a	4b	7
2b	A	11
1d	D	4
2a	D	8
2a	3c	13
A	2b	4
3c	B	11
1d	C	5
1d	C	13
1c	D	9
1a	2c	6
1c	4c	10
1a	4d	8
3d	A	10
1d	A	7
1c	3d	14
1d	C	3
1d	4a	3
3c	4b	8
4b	1a	9
1c	1a	10
3c	3b	14
4d	3d	14

C	C	12
4b	3b	10
4a	A	4
3a	A	1
3a	B	11
4a	D	4
1b	4a	4
1a	3c	13
3c	A	11
D	1b	6
3b	2c	3
B	4b	6
3a	4b	12
1b	2d	2
3b	3d	14
3a	1a	3
4b	3b	12
2a	A	12
1d	C	12
D	3c	10
1d	2d	1
4c	C	8
A	2d	8
A	2a	3
4c	3d	9
3b	4c	4
2a	1a	11
4c	2b	1
4d	2d	6
C	A	1
2d	4c	12
4a	D	13
2d	2c	11
3c	3a	7
3c	3b	9
2a	1d	9
1c	4c	7
1b	3b	1
2d	2b	2
3a	3b	5
3d	3b	5
4b	1a	9
2b	2a	8
4a	4d	8
4d	3a	6
4d	4b	5
4b	A	2
3b	2a	12

2d	1d	4
1c	4a	5
A	4a	7
1c	2b	4
3b	1c	2
B	1b	9
2d	4c	14
C	4b	12
2a	4d	3
2b	B	6
2b	4d	2
C	3a	9
3a	2a	10
1c	4b	14
A	1a	11
2d	2a	1
5d	3c	14
2a	3a	4
1c	C	6
3d	2c	10
C	3b	6
2a	2b	11
2d	3a	12
3d	1b	10
4b	3d	4
3d	4c	2
1a	3a	1
4c	2d	12
2b	4c	7
2a	C	8
2c	1b	6
3b	3d	5
C	A	12
1d	1c	14
2a	D	14
3c	1d	3
3c	3d	1
1d	2a	13
3c	4a	13
4d	3a	2
2d	3b	11
A	3c	14
D	C	11
4d	4b	13
4c	1b	9
3d	4b	13
2d	1c	10
C	3c	1

4c	3b	7
3a	D	2
2c	A	11
2a	B	4
1c	2c	4
A	3d	7
1c	4a	1
C	2b	2
3b	2c	2
2d	4c	2
2b	B	12
2a	1d	6
D	4d	6
B	3b	1
1c	3b	9
4a	3d	14
2b	3b	5
A	2a	8
1c	4d	5
3c	B	6
C	2a	10
3a	2d	8
3d	3d	14
1b	3a	8
C	A	12
2c	1b	11
2a	A	1
4c	3b	6
4c	2d	14
2a	2c	6
4c	1b	1
1a	1c	1
1c	2b	7
2b	2c	6
4a	A	14
1b	2c	9
4d	2b	8
2a	4b	1
B	3d	6
D	3a	4
4d	4b	7
2b	1a	11
1d	2d	4
3d	2a	4
2d	B	5
B	A	12
D	2a	8
4a	3b	2

A	2c	9
2b	D	14
4a	2c	1
C	2d	8
1b	2d	12
1d	2b	14
1a	4c	3
4b	1c	7
3c	C	12
D	1c	10
3a	1d	1
2a	4a	12
4c	3c	3
C	1c	8
3d	4b	13
3a	D	2
1a	4c	14
1a	2d	9
3b	1c	13
3b	C	4
4b	B	8
B	2d	3
3c	A	5
1d	A	12
1a	B	4
2d	2a	2
1d	3c	1
1c	2a	14
2c	4b	8
A	2c	11
A	4a	12
3d	4b	6
4b	1a	8
3a	2c	2
B	D	8
3c	3d	6
3a	3b	1
D	2b	13
3d	2d	6
A	4b	12
3a	B	5
3b	D	11
C	4c	1
1a	1b	12
2c	B	2
A	D	6
4d	1d	1
3c	4b	3

1c	1a	5
B	2b	6
3a	3d	2
2a	D	13
3c	2d	2
1a	C	2
D	2c	6
1d	2b	1
1d	2a	1
1b	D	4
3a	3b	7
C	2c	4
2c	4d	10
4d	B	4
3c	B	2
D	4c	14
3d	4d	6
4c	2c	7
D	3c	8
1b	3b	7
C	B	6
4c	C	14
A	D	6
2a	A	4
2c	A	2
1b	4a	7
4d	A	11
C	3c	2
4b	1c	6
3c	4b	4
3c	2d	4
4a	2d	8
3d	4a	9
1a	2c	11
2c	2a	14
4b	4a	13
2c	2d	10
2d	B	5
1a	4a	12
2d	3c	2
4d	4c	6
2b	1c	9
3b	1c	4
1c	D	5
2d	1a	8
D	3d	8
4b	1c	13
1d	3c	5

1b	3a	5
4c	1b	5
2b	1c	6
1d	2c	3
3a	C	14
B	D	13
1d	4d	4
4b	1d	14
3b	1c	12
1d	C	3
4b	3d	7
1a	1c	14
3a	1b	14
2b	2d	14
2c	2d	10
4a	4b	8
4c	4a	3
3a	1b	4
4b	1a	7
2a	4c	3
2b	2d	12
B	2a	10
4c	3b	6
2c	1a	9
C	2b	3
2c	4a	12
1b	4b	1
2a	1d	6
2a	1a	7
C	2d	1
D	2d	10
4b	2b	3
4a	B	14
4d	D	4
3d	B	8
D	1a	1
B	1d	6
2b	4c	8
2d	D	11
4a	4b	10
C	2d	4
1a	2c	11
1b	D	4
1c	3d	4
D	1d	3
4c	2b	1
3a	1c	14
1a	D	14

1c	3d	6
3a	1c	6
2c	3c	14
1b	3c	10
1b	1d	13
C	3c	2
1b	B	10
2a	3c	11
4c	4b	13
1c	2a	6
4b	A	9
D	2b	11
A	4c	13
B	2d	4
2b	B	4
4b	2d	13
D	3c	4
4c	B	2
1d	2d	5
2c	3d	13
4d	1d	1
3b	D	3
1a	1c	10
3c	4c	5
2c	A	14
2c	2d	7
1d	4c	9
2a	1d	6
3a	2b	13
3a	2d	13
2a	2c	7
A	3a	5
C	4b	12
2b	4b	14
1a	3b	4
2b	A	4
1b	1d	12
3c	2a	4
1a	4b	13
1a	D	14
D	3c	13
3a	3d	9
2c	3b	12
2a	4a	6
1b	A	6
3c	4a	2
3c	3a	2
C	4d	14

1a	C	12
4b	4c	8
1d	2d	8
2d	3b	4
2a	A	3
4a	3c	7
B	1a	14
2b	4a	13
1b	1c	9
B	4d	6
C	3b	10
A	3d	13
1a	2a	12
2a	B	2
3c	A	12
2c	4a	10
D	1a	11
1c	2c	3
3d	4a	8
3b	4d	3
4b	1a	7
4b	B	6
3c	2b	2
3c	2d	4
1d	A	9
D	3a	4
4b	2a	5

3a	4c	3
2b	1a	1
B	C	7
4d	2b	5
3c	1a	6
3c	1a	13
1d	4d	8
C	4b	13
1c	4b	2
D	2b	12
1b	3c	1
3d	4c	6
C	4a	14
2c	D	8
4b	3d	8
A	B	12
4a	3d	5
1b	4b	10
3c	1a	10
3b	C	3
2c	3a	10
1d	2d	1
4d	4c	7
4c	2d	3
4d	1a	3
1c	3d	13

4d	2c	10
B	3c	8
1b	1d	4
1d	2d	10
4a	3d	14
2b	3a	10
4c	1c	11
4b	4d	3
A	1a	8
3c	1c	7
1a	4b	7
2b	2c	4
2c	2a	2
1a	4a	9
4b	C	3
1c	C	7
3d	2a	14
C	1a	10
1a	3d	13
2a	4c	7
3b	3c	4
1d	2c	1
2d	4c	14
2b	B	10
2a	4a	10
4b	2b	8