Web Service Composition Using The Bidirectional Dijkstra Algorithm

F. Moo, R. Hernández, V. Uc and F. Madera

***Abstract*— Web services are not always able to fulfill customer requirements on their own, so in those cases it can choose to make a composition of web services. However, this is a complex problem since it must take into account the large number of available services, performance requirements, and other factors related to quality of service (QoS). Representing the problem of web service composition as a graph, some methods were used that do not ensure optimum solution. In this work the bidirectional Dijkstra algorithm is proposed to solve the problem of web services composition. Experimental results show that, as the number of web service classes increases, proposed algorithm performance improves.**

***Keywords*— web service composition, Dijkstra algorithm, bidirectional search, dynamic programming.**

I. INTRODUCCIÓN

EL WORLD Wide Web Consortium define un servicio web como un sistema de software diseñado para soportar interacción interoperable maquina a máquina sobre una red. Tiene una interfaz descrita en un formato capaz de ser procesado por máquinas (específicamente WSDL [1]). Otros sistemas interactúan con el servicio web en una manera prescrita por su descripción usando mensajes SOAP, típicamente transportados usando HTTP con una serialización XML en conjunción con otros estándares relacionados con la Web [2]. Cuando servicios web individuales no son capaces de cumplir con requerimientos complejos pueden ser combinados para crear servicios compuestos que sí puedan cumplir con esos requerimientos.

Para crear un servicio web compuesto sería ideal descubrir y seleccionar el servicio más apto para las posibles solicitudes del cliente. Sin embargo, el número de servicios web se incrementa continuamente, por lo que encontrar el mejor servicio no es un problema trivial, especialmente si al seleccionar se toman en cuenta, además de las funcionalidades del servicio, parámetros de calidad de servicio (QoS) como disponibilidad, tiempo de ejecución, entre otros. Calidad de servicio se refiere a un conjunto de técnicas cuyo objetivo es hacer que coincidan las necesidades de solicitantes y proveedores de los servicios, ambas basadas en los recursos de red disponibles [3].

F. Moo, Universidad Autónoma de Yucatán, Mérida, Yucatán, México, mmena@correo.uady.mx

R. Hernández, Universidad Autónoma de Yucatán, Mérida, Yucatán, México, rahernandez188@gmail.com

V. Uc, Universidad Autónoma de Yucatán, Mérida, Yucatán, México, uccetina@correo.uady.mx

F. Madera, Universidad Autónoma de Yucatán, Mérida, Yucatán, México, mramirez@correo.uady.mx

La complejidad de la selección de servicios para la composición de servicios web incluye tres factores principales: 1) el gran número de servicios web con funcionalidad similar que pueden estar disponibles, 2) las diferentes posibilidades de integración de los servicios web simples en un servicio más complejo, y 3) Varios requerimientos de desempeño de un servicio complejo (por ejemplo, costo de los servicios, confiabilidad, etc.) [4].

Utilizando un enfoque para resolver el problema de composición de servicios web, en el que las posibles composiciones se representan mediante un grafo dirigido ponderado, se han utilizado algoritmos heurísticos simples [5] y bidireccionales [6,7]. Sin embargo, una desventaja de los algoritmos heurísticos es que no es posible asegurar que la mejor solución encontrada sea efectivamente la mejor solución para el problema. Además, determinar la función heurística a utilizar puede ser muy complicado. Además, una vez que se tiene el grafo con todas las composiciones posibles, el algoritmo de Dijkstra proporciona una manera rápida de encontrar la mejor solución para el camino más corto en un grafo dirigido ponderado [8], sin embargo ha sido un método poco utilizado.

En este trabajo se propone el uso del algoritmo de Dijkstra bidireccional como una solución al problema de composición de servicios web y se le compara con otras soluciones. Para mostrar la viabilidad del algoritmo, se llevaron a cabo experimentos en los que se le compara con el algoritmo de Dijkstra clásico y tres algoritmos de programación dinámica. Aunque todos los algoritmos proveen una buena solución, el algoritmo de Dijkstra bidireccional muestra una ligera mejora en el tiempo respecto a los resultados del algoritmo de Dijkstra clásico y una considerable mejora en el tiempo obtenido por los algoritmos de programación dinámica.

El artículo está estructurado como sigue: en la sección 2 se presentan trabajos relacionados, en la sección 3 se presentan los aspectos básicos de los procesos de decisión Markov y en la sección 4 se presenta el algoritmo de Dijkstra. La sección 5 presenta los modelos que se utilizaron para resolver el problema de composición de servicios web. En la sección 6 se presentan los experimentos y sus resultados. Finalmente en la sección 7 se presentan las conclusiones del trabajo.

II. TRABAJOS RELACIONADOS

Yan, et. al [9] proponen un algoritmo de composición basado en una ontología de QoS. La ontología presentada, QoSHOnt, consiste de tres capas: La ontología superior (upper ontology) captura y define los conceptos más generales de QoS. La ontología media (middle ontology) se usa para incorporar características de calidad que son definidas con conceptos de la ontología superior y la ontología inferior contiene conceptos de aplicación específica del dominio,

derivados de la ontología media. Para cualquier servicio web solicitado, se puede obtener el mejor servicio sucesor inmediato de acuerdo a sus parámetros QoS. Este formará parte de la composición y el proceso se repite hasta completarla. La principal desventaja de este método es la posibilidad de una composición "vacía", es decir, que al final del proceso se hayan agregado servicios a la composición y por lo tanto ésta no se lleve a cabo. Los resultados muestran que el algoritmo puede asegurar la calidad de servicio y además reducir el tiempo de composición.

Zhang [6] propone un algoritmo heurístico bidireccional para realizar la composición de servicios web basado en el concepto de distancia de composición (composition distance), sobre un grafo que representa las distintas secuencias de composición. La distancia de composición es la distancia más corta entre dos servicios web a_1 y a_2 cuando éstos se componen juntos. El algoritmo utilizado es de búsqueda en espacio de estados. Los resultados de la experimentación muestran que el algoritmo encuentra una ruta eficiente de composición.

Wang, et al. [10] modelan el problema como un proceso de decisión Markov (MDP), de manera que múltiples servicios alternativos y flujos de trabajo puedan ser incorporados en una composición de servicios. Los estados del modelo representan los servicios web a partir de los cuales es posible invocar otros servicios; el llamar a un servicio implica recibir una recompensa. La optimización de la composición se realiza mediante aprendizaje por refuerzo. Para facilitar el proceso de aprendizaje se utiliza Q-learning para simular la recompensa acumulada. El algoritmo propuesto hace la distinción entre parámetros QoS positivos y negativos al calcular la recompensa. Los resultados muestran que el método propuesto se adapta bien a los cambios en el ambiente. Yu, et al. [11] utilizan un enfoque similar. De igual manera, los autores de [12] usan un MDP para la composición de servicios web; proponen usar un algoritmo basado en el algoritmo value iteration. Sin embargo únicamente hacen la propuesta sin hacer experimentos. El trabajo presentado en [13] usa un MDP y el algoritmo value iteration junto a los algoritmos iterative policy evalutation y policy iteration, todos presentados en [14], comparados con los algoritmos Sarsa y Q-learning. El artículo presenta experimentos realizados sobre un escenario artificial y uno real; los resultados muestran que los primeros tres algoritmos representan una mejora considerable en el tiempo que toma encontrar la ruta con la mayor recompensa, mientras que Sarsa y Q-learning tardan mucho tiempo en encontrar una solución buena pero que no es óptima.

Yang, et al. [15] presentan un algoritmo GraphPlan que toma en cuenta los parámetros QoS y que usa el principio del algoritmo de Dijkstra para hacer una búsqueda sistemática del mejor camino QoS en un grafo de planificación. El enfoque presentado es capaz de funcionar para diferentes criterios QoS. De igual manera, Guo et al. [16] utilizan una red de servicios para construir un grafo de relaciones semánticas en el que se utiliza el algoritmo del k-ésimo camino más corto implementado con búsqueda bidireccional para obtener las k cadenas de servicios con mejor calidad de servicio. El grafo se construye tomando en cuenta las entradas, las salidas y las interfaces de los servicios web; las interfaces se agrupan en interfaces abstractas de acuerdo a la función que llevan a cabo. El algoritmo para encontrar el camino más corto se basa en el algoritmo de Dijkstra. Tomando en cuenta únicamente tiempo de respuesta como parámetro de calidad de servicio, los experimentos llevados a cabo muestran la eficiencia y exactitud del método propuesto.

III. PROCESOS DE DESICIÓN MARKOV

El problema de composición de servicios web puede ser visto como un problema de selección de una secuencia de acciones, de manera que se maximice una función de evaluación. Este tipo de problemas de decisión secuencial puede ser resuelto mediante un proceso de decisión Markov (MDP) [13,14]. Un MDP es una tupla (S,A,P,γ, R), donde S es el conjunto de estados, A es el conjunto de acciones, $P(s_{t+1}|s_t,a_t)$ son probabilidades de transición de estado para todos los estados $s_t,s_{t+1} \in S$ y acciones $a \in A$, $\gamma \in [0,1)$ es un factor de descuento y $R: S \times A \to \Re$ es la función de recompensa. Un MDP funciona de la siguiente manera. Un agente en un estado $s_t \in S$ lleva a cabo una acción a_t, seleccionada del conjunto de acciones A. Como resultado de realizar la acción a_t, el agente recibe una recompensa con un valor esperado $R(s_t,a_t)$ y el estado actual del MDP hace una transición hacia otro estado sucesor s_{t+1}, de acuerdo a las probabilidad de transición $P(s_{t+1}|s_t,a_t)$. Una vez en el estado s_{t+1}, se elige y ejecuta una acción a_{t+1}, recibiendo una recompensa $R(s_t,a_t)$ y moviéndose al estado s_{t+2}. El agente continua eligiendo y ejecutando acciones creando una ruta de estados visitados $s_t,s_{t+1},s_{t+2},\ldots$. Conforme el agente pasa por los estados $s_t,s_{t+1},s_{t+2},\ldots$, obtiene las siguientes recompensas:

$$R(s_0,a_0) + \gamma R(s_1,a_1) + \gamma_2 R(s_2,a_2) + \cdots \quad (1)$$

La recompensa en el tiempo t se descuenta por el factor γ_t, de manera que el agente proporcione más importancia a las recompensas que se obtienen más rápido. En un proceso de decisión Markov, se trata de maximizar la suma de las recompensas esperadas obtenidas por el agente:

$$E[R(s_0,a_0) + \gamma R(s_1,a_1) + \gamma_2 R(s_2,a_2) + \ldots] \quad (2)$$

Una política es definida como función $\pi : S \to A$ que mapea los estados a las acciones. Una función de valor para una política π es la suma esperada de las recompensas descontadas, obtenida realizando siempre las acciones que la política π indica:

$$V^{\pi}(s) = E[R(s_0,\pi(s_0)) + \gamma R(s_1,\pi(s_1)) + \gamma_2 R(s_2,\pi(s_2)) + \ldots | s_0 = s,\pi] \quad (3)$$

V^{π} es la suma esperada de las recompensas descontadas que el agente recibirá si inicia en el estado s y toma las acciones indicadas por π. Dada una política fija π, su función de valor satisface la ecuación de Bellman:

$$V^{\pi}(s) = R\big(s,\pi(s)\big) + \gamma \sum_{s' \in S} \big(s' \big| s,\pi(s)\big) V^{\pi}(s') \quad (4)$$

La función de valor óptima está definida como

$$V*(s) = máx\, V\pi(s) \quad (5)$$

La función da la mejor suma esperada posible de las recompensas descontadas que puede ser obtenida usando cualquier política π. La ecuación de Bellman para la función de valor óptima es

$$V^*(s) = R(s,a) + \gamma \sum_{s' \epsilon S}(s'|s,a)V^*(s') \quad (6)$$

La función de valor óptima es tal en la que se tiene

$$V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s) \quad (7)$$

Programación dinámica para resolver MDP's. Cuando las probabilidades de transición se conocen, la programación dinámica puede usarse para encontrar la solución de (6). Los siguientes son tres algoritmos para resolver MDP's de estado finito usando programación dinámica [14]. El primero es iterative policy evaluation (algoritmo1). El segundo es el algoritmo policy value iteration (algoritmo 2), que calcula repetidamente el valor de la función para la política actual y actualiza la política usando la función de valor actual. El tercero, value iteration (algoritmo 3), es una actualización iterativa de la función de valor usando la ecuación de Bellman (6).

Algoritmo 1: Iterative policy evaluation [13,14]

1 **foreach** *state* **do**
2 $V(s) \leftarrow 0$
3 **repeat**
4 **Foreach** *state* **do**
5 $V_{i+1}(s) \leftarrow \sum_{a \in A} \pi(s,a)\ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_i(s')]$
6 **until** *convergence*;

Algoritmo 2: Policy iteration [13,14]

1 initialize π_0 randomly
2 **repeat**
3 $V_i(s) \leftarrow R(s,\pi_i(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi_i) V_i(s')$
4 **foreach** *state* **do**
5 $\pi_{i+1}(s) \leftarrow \arg máx_{a \in A} [R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_i(s')]$
6 **until** *convergence*;

Algoritmo 3: Value iteration [13,14]

1 **foreach** *state* **do**
2 $V(s) \leftarrow 0$
3 **end**
4 **repeat**
5 **foreach** *state* **do**
6 $V_{i+1}(s) \leftarrow máx_{a \in A} [R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_i(s')]$
7 **end**
8 **until** *convergence*;

IV. ALGORITMO DE DIJKSTRA

El problema de composición de servicios web puede ser visto como un problema del camino más corto en un grafo dirigido ponderado. Este problema puede ser resuelto con el algoritmo de Dijkstra (algoritmo 4), que resuelve el problema del camino más corto con una sola fuente en un grafo dirigido ponderado $G = (V,E)$ para el caso en el que todos los pesos de las aristas son no negativos, es decir, $w(u,v) \geq 0$ para cada arista (u,v). El algoritmo mantiene un conjunto S de vértices cuyos pesos, que representan el camino más corto respecto a la fuente o estado inicial s, ya han sido determinados. Se selecciona el vértice $u \in V - S$ con el mínimo camino más corto estimado, se agrega u a S y se relajan todas las aristas que salen de u.

Algoritmo 4: Algoritmo de Dijkstra [8].

Data: G: a weighted directed graph, w: all edge weights, s: source or initial state.
Result: The single-source shortest path.

1 **for** *each vertex* $v \in V[G]$**:**
2 $d[v] \leftarrow \infty$
3 $\pi[v] \leftarrow NIL$
4 $d[s] \leftarrow 0$
5 $S \leftarrow \emptyset$
6 $Q \leftarrow V[G]$
7 **while** $Q \neq \emptyset$**:**
8 $u \leftarrow$ EXTRACT-MIN(Q)
9 $S \leftarrow S \cup [u]$
10 **for** *each vertex* $v \in Adj[u]$**:**
11 **if** $d[u] > d[u] + w(u,v)$**:**
12 d[u] ← d[u] + w(u,v)
13 $\pi[u] \leftarrow u$

Las líneas 1-3 realizan la inicialización usual de los valores de d y π, y la línea 2 inicializa el conjunto S como vacío. El algoritmo mantiene invariante que $Q = V - S$ al principio de cada iteración del ciclo while de las líneas 7-13. Línea 6 inicializa Q con todos los vértices de G ($Q = V - S$ sique siendo verdadero ya que $S = \emptyset$). Cada vez que se ejecuta el ciclo while, un vértice u es extraído de $Q = V - S$ y añadido a S

manteniendo así la invariante. En la primera iteración, $u = s$, es decir, primero se toma la fuente o estado inicial. El vértice u seleccionado tiene el menor peso estimado, y en las líneas 10-13 se relajan las aristas que salen de u, actualizando los valores $d[v]$ y los predecesores $\pi[v]$ si el camino más corto a v puede ser mejorado a través de u.

V. MODELOS DE COMPOSICIÓN DE SERVICIOS WEB

Para los algoritmos de programación dinámica, es necesario definir la composición de servicios web como un proceso de decisión Markov. Sus elementos S,A,P y R se definen como sigue [13]:

- Estados. Dado un problema de composición de servicios web con C clases, S representa todas las posibles composiciones cuyo número de servicios es a lo más C. Además se tienen dos estados S y G que representan el estado inicial (ningún servicio web en la composición) y final (composición completada) respectivamente.
- Acciones. A es el conjunto de todas las acciones. Dado un estado s, el conjunto de acciones disponibles para s se denota por $A(s)$. Una acción consiste en seleccionar un servicio web para ser incluido en la composición. Si la composición es de largo $l = i$, todas las posibilidades de seleccionar un servicio web de clase $c = i+1$ constituyen el conjunto de acciones disponibles.
- Probabilidades de transición. $P(s^0|s,a)$ son las probabilidades de transición de estado para todos los estados $s,s^0 \in S$ y las acciones $a \in A$ que están disponibles para s y s^0. Solo es posible pasar de una composición de largo $l = i$ a una de largo $l = i + 1$, por lo que la probabilidad para estos casos será de 1. Para los demás casos será 0.
- Función de recompensa. $R(s^0|s,a)$ es la recompensa recibida cuando una acción a se ejecuta y se hace una transición de s a s^0. Para este modelo la función de recompensa toma en cuenta la disponibilidad, rendimiento y tiempo de ejecución del servicio web:

$$R(s) = \frac{av^s - av^{min}}{av^{max} - av^{min}} - \frac{time^s - time^{min}}{time^{max} - time^{min}} + \frac{tr^s - tr^{min}}{tr^{max} - tr^{min}} \quad (8)$$

donde av^s, $time^s$, tr^s son la disponibilidad, tiempo promedio de ejecución y rendimiento del servicio web más reciente añadido a la composición representada por el estado s. av^{min}, $time^{min}$, tr^{min} y av^{max}, $time^{max}$, tr^{max} son los valores mínimos y máximos de cada atributo para todos los servicios web.

Para el algoritmo de Dijkstra, la composición de servicios web puede ser representada por medio de un grafo dirigido cuyos componentes son los siguientes:

- Vértices. V es el conjunto de vértices. De la misma manera que en el MDP, V representa todas los posibles composiciones de, a lo más, C servicios. Se tienen también los vértices S y G, de significado equivalente a los estados del MDP.
- Aristas: Si la salida de un servicio web A puede ser usada como entrada de otro servicio B, significa que es posible llamar al servicio B. Esta relación se representa con una arista de A a B cuyo peso dependerá de los parámetros de calidad de servicio de B. En este trabajo se asume que la salida del servicio A puede ser enviada tal cual al servicio B sin alguna clase de preprocesamiento.

VI. EXPERIMENTACIÓN

Programación dinámica y algoritmo de Dijkstra. Se realizaron dos experimentos, uno en un escenario real y el otro artificial, en una computadora con procesador Intel Core i5 2.5 GHz, sistema operativo Windows 8.1 de 64 bits, y 6 Gb de RAM. Se usó el lenguaje de programación Java para implementar tanto los algoritmos de programación dinámica como el de Dijkstra. En cada caso a cada algoritmo se le proporcionaba una matriz de adyacencias, que indica que vértices salen aristas hacia otros vértices en el grafo. Con la matriz de adyacencias es posible hacer una iteración del tamaño de una de sus dimensiones, por lo que el algoritmo de Dijkstra implementado no requiere del uso de un conjunto Q para almacenar los vértices. Además, se hicieron modificaciones al algoritmo de Dijkstra para que encuentre el camino más largo o con mayor peso, esto para que funcione de manera similar a los algoritmos de programación dinámica. El algoritmo 5 muestra el algoritmo modificado; en la línea 6 se puede apreciar que la condición original del algoritmo se ha cambiado.

Algoritmo 5: Algoritmo de Dijkstra modificado para los experimentos.

Data: G: a weighted directed graph, w: all edge weights, s: source or initial state.
Result: The single-source largest path.

```
1  for each vertex v ∈ V [G]:
2      d[v] ← 0
3      π[v] ← NIL
4  for each vertex u ∈ V [G]:
5  for each vertex v ∈ Adj[u]:
6      if d[u] < d[u] + w(u,v):
7          d[u] ← d[u] + w(u,v)
8          π[u] ← u
```

En el escenario real, el problema de composición presentado en el primer escenario consiste en dos clases de servicios web. La primera clase es de servicios del estado del tiempo en la que se puede obtener la temperatura de una ciudad. La otra clase es de servicios web que pueden ser usados para convertir temperaturas de una unidad a otra, por ejemplo, de Fahrenheit a Celsius. Los servicios utilizados son los siguientes:

1. Servicio web de la National Oceanic and Atmospheric Administration, disponible en http://graphical.weather.gov/xml/SOAP_server/ndfdXMLserver.php.

2. Servicio web GlobalWeather, disponible en http://www.webservicex.net/globalweather.asmx.
3. Servicio web del Weather channel, disponible en http://api.wunderground.com/.

En la clase de servicios web de conversión de temperaturas se tienen 4 servicios web.

1. Un servicio web de calculadora simple como el que está disponible en http://www.dneonline.com/ calculator.asm. Dado que

$$C = \frac{5*(F-32)}{9} \quad (9)$$

, podemos usar las operaciones de resta, multiplicación y división para la conversión de temperaturas.

2. Servicio web ConvertTemperature, disponible en http://www.webservicex.net/ConvertTemperature.asmx.
3. Servicio web TemperatureConversions, disponible en http://webservices.daehosting.com/services/TemperatureConversions.wso
4. Servicio web TempConver, disponible en http://www.w3schools.com/webservices/tempconvert.asmx

Los valores de los parámetros de calidad de servicio se obtuvieron mediante un programa en lenguaje Java diseñado para obtener los valores usando las siguientes formulas:

$$Availability = \frac{C_S}{C_T} \quad (10)$$

donde C_S es el número de llamadas exitosas al servicio web y C_T es el total de llamadas,

$$Executiontime = \frac{T}{C_T} \quad (11)$$

donde T es el tiempo total de ejecución para todas las C_T llamadas, y

$$Throughput = \frac{C_S}{T} \quad (12)$$

con C_T= 50.

Los valores de los parámetros de calidad de servicio fueron obtenidos durante varios días en diferentes momentos del día, esto para obtener valores representativos. Se obtienen los valores de cada parámetro y se calculan los valores promedio.

Los resultados obtenidos con estos servicios se muestran en la Fig. 1. El algoritmo de Dijkstra muestra un mejor tiempo que los tres algoritmos de programación dinámica, obteniendo un tiempo máximo de 8.21 × 10^{-6} en la primera ejecución. El factor de descuento γ no afecta al algoritmo de Dijkstra por lo que únicamente se realizaron 10 ejecuciones sucesivas.

Para el escenario artificial se crearon 18000 servicios web individuales. Dado que cada clase contiene 18 servicios web, al resolver un problema de composición de servicios web con 1000 clases hipotéticas o nodos, realmente se está resolviendo un problema de 18 × 1000 = 18000 servicios web. Los resultados obtenidos se muestran en la Fig. 2. De nuevo el algoritmo de Dijkstra muestra un mejor tiempo que los tres algoritmos de programación dinámica.

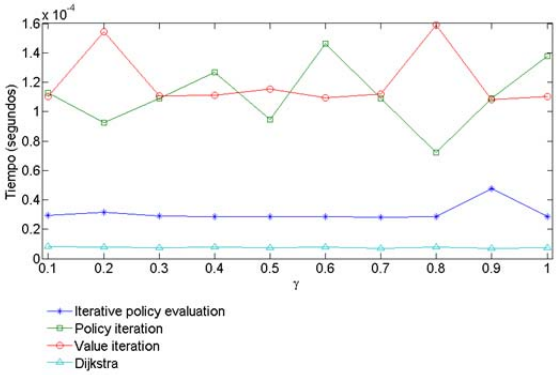

Figura 1. Tiempos en el escenario real.

Algoritmo 6: Algoritmo de Dijkstra bidireccional.

Data: G: a weighted directed graph, w: all edge weights, s: source or initial state.
Result: The single-source largest path.

for *each vertex* $v \in V[G]$**:**
  $d_1[v] \leftarrow 0$
  $d_2[v] \leftarrow 0$
  $\pi[v] \leftarrow NIL$
  $\pi_2[v] \leftarrow NIL$
for *each vertex* $u,v \in V[G]$**:**
//From start to end
  for *each vertex* $x \in Adj[u]$**:**
    if $d_1[u] < d_1[u] + w(u,x)$**:**
      $d_1[u] \leftarrow d_1[u] + w(u,x)$
      $\pi_1[u] \leftarrow u$
//From end to start
  for *each vertex* $x \in PrAdj[v]$**:**
    if $d_2[x] < d_2[v] + w(x,v)$**:**
      $d_2[x] \leftarrow d_2[u] + w(x,v)$
      $\pi_2[x] \leftarrow u$
End when both paths find each other.

Algoritmo de Dijkstra bidireccional. La idea de la búsqueda bidireccional es ejecutar dos búsquedas simultáneas, una hacia adelante desde el estado inicial y otra hacia atrás desde el estado final, esperando que se encuentren en medio. En el caso del algoritmo de Dijkstra bidireccional (algoritmo 6), se llevan a cabo dos procesos: el algoritmo de Dijkstra clásico que empieza en el nodo S y va hacia adelante y una versión hacia atrás del mismo algoritmo que empieza desde el nodo G.

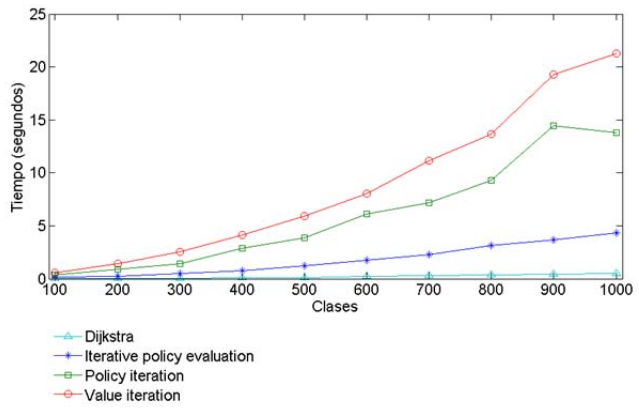

Figura 2. Tiempos en el escenario artificial.

El algoritmo termina cuando ambos recorridos se encuentran, aproximadamente a la mitad del grafo. Las líneas 7 a 10 representan la parte del algoritmo que va del nodo inicial al nodo final, mientras que las líneas 11 a 14 muestran la pare que va del nodo final al inicial. Ambos recorridos pueden llevarse a cabo en forma paralela, lo que permitió probar implementaciones tanto en Java como en C, usando threads y OpenMP respectivamente. Para comparar el algoritmo de Dijkstra bidireccional con los algoritmos de programación dinámica y con el algoritmo clásico se llevaron a cabo dos experimentos tomando los escenarios real y artificial anteriores en la misma máquina. Mientras que la implementación en lenguaje Java tomo´ mucho más tiempo que el algoritmo clásico, la implementación en lenguaje C sí obtuvo una mejora en el tiempo.

En el escenario real, aunque las implementaciones de Dijkstra obtuvieron mejor tiempo que los algoritmos de programación dinámica, el algoritmo de Dijkstra clásico obtuvo mejor tiempo que el bidireccional (Fig. 3). De igual manera, en el escenario artificial las implementaciones de Dijkstra mostraron un mejor tiempo que los algoritmos de programación dinámica. La Fig. 4 muestra los tiempos obtenidos para este segundo escenario. Comparando las implementaciones del algoritmo de Dijkstra se puede ver que con 100 clases, el tiempo del algoritmo clásico es ligeramente mejor que el tiempo del algoritmo bidireccional. La Fig. 5 muestra que se logra una pequeña mejora en el tiempo de 200 a 400 clases y una mejora significativa a partir de 500.

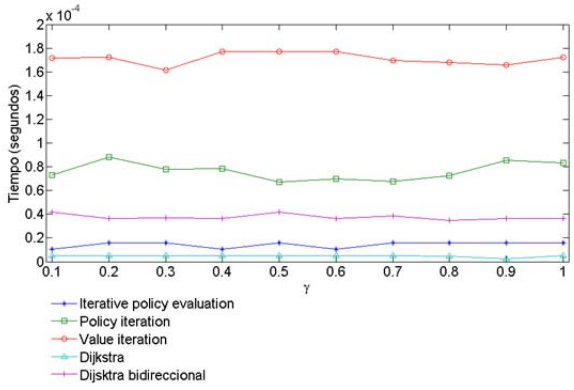

Figura 3. Tiempos del algoritmo de Dijkstra bidireccional vs algoritmo clásico vs algoritmos de programación dinámica en el escenario real.

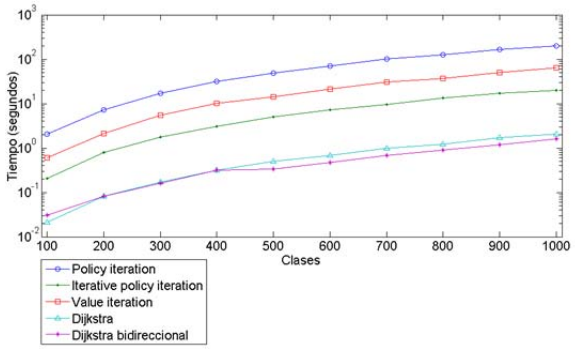

Figura 4. Tiempos del algoritmo de Dijkstra bidireccional vs algoritmo clásico vs algoritmos de programación dinámica en el escenario artificial.

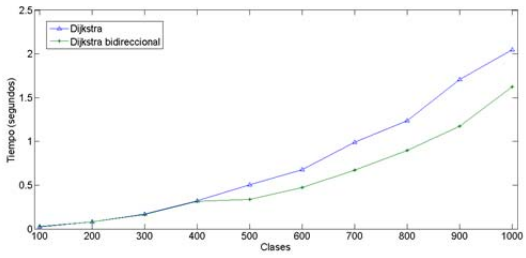

Figura 5. Tiempos del algoritmo de Dijkstra bidireccional vs algoritmo clásico en el escenario artificial.

VII. CONCLUSIONES

En este trabajo se propone el algoritmo de Dijkstra bidireccional como método para resolver el problema de composición de servicio web. Los experimentos fueron realizados con datos creados artificialmente y con un conjunto de datos reales que involucran siete servicios web disponibles públicamente. Los resultados experimentales muestran que el algoritmo de Dijkstra bidireccional es efectivo cuando se manejan una gran cantidad de clases de servicios web, mientras que el algoritmo de Dijkstra clásico es mejor cuando el número de clases es pequeño. Como trabajo futuro se propone probar los algoritmos con diferentes patrones de composición. Además, se está trabajando en integrar la composición de servicios web con un agente que calcule los parámetros de calidad de servicio y otro que haga un filtrado previo de ellos.

AGRADECIMIENTOS

Agradecemos a la Universidad Autónoma de Yucatán (UADY) y al Consejo Nacional de Ciencia y Tecnología de México (CONACYT) por el apoyo prestado a esta investigación.

REFERENCIAS

[1] A. Ryman, S. Weerawarana, J.-J. Moreau, and R. Chinnici, "Web services description language (WSDL) version 2.0 part 1: Core language," W3C recommendation, W3C, June 2007.
http://www.w3.org/TR/2007/REC-wsdl2020070626.
[2] D. Booth, H. Haas, F. McCabe, C. Ferris, M. Champion, D. Orchard, and E. Newcomer, "Web services architecture," W3C note, W3C, Feb. 2004. http://www.w3.org/TR/2004/NOTEws-arch-20040211/.

[3] A. Mani and A. Nagarajan, "Understanding quality of service for web services."
https://www.ibm.com/developerworks/java/library/wsquality/.
[4] F. Qiqing, P. Xiaoming, L. Qinghua, and H. Yahui, "A global qos optimizing web services selection algorithm based on moaco for dynamic web service composition," in Information Technology and Applications, 2009. IFITA '09. International Forum on, vol. 1, pp. 37–42, 2009.
[5] H. Shen, Z. Ding, and H. Chen, "Reliable web services selection using a heuristic algorithm," in Grid and Cooperative Computing (GCC), 2010 9th International Conference on, pp. 290–295, 2010.
[6] B. Zhang, "A heuristic bidirectional search algorithm for automatic web service composition," in Advanced Intelligence and Awarenss Internet (AIAI 2010), 2010 International Conference on, pp. 407–411, Oct 2010.
[7] N. Ukey, R. Niyogi, K. Singh, A. Milani, and V. Poggioni, "A bidirectional heuristic search for web service composition with costs," Int. J. Web Grid Serv., vol. 6, pp. 160–175, June 2010.
[8] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, Introduction to Algorithms. McGraw-Hill Higher Education, 2nd ed., 2001.
[9] H. Yan, W. Zhijian, and L. Guiming, "A novel semantic web service composition algorithm based on qos ontology," in Computer and Communication Technologies in Agriculture Engineering (CCTAE), 2010 International Conference On, vol. 2, pp. 166– 168, June 2010.
[10] H. Wang, X. Zhou, X. Zhou, W. Liu, and W. Li, "Adaptive and dynamic service composition using q-learning," in Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on, vol. 1, pp. 145–152, Oct 2010.
[11] L. Yu, W. Zhili, M. Lingli, W. Jiang, L. Meng, and Q. Xue-song, "Adaptive web services composition using q-learning in cloud," in Services (SERVICES), 2013 IEEE Ninth World Congress on, pp. 393–396, June 2013.
[12] V. Todica, M.-f. Vaida, and M. Cremene, "Using Machine Learning in Web Service Composition," in SERVICE COMPUTATION 2012, The Fourth International Conferences on Advanced Service Computing, pp. 122–126, 2012.
[13] V. Uc-Cetina, F. Moo-Mena, and R. Hernandez-Ucan, "Composition of Web Services Using Markov Decision Processes and Dynamic Programming," The Scientific World Journal, vol. 2015, p. 9, 2015.
[14] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, MA, USA: MIT Press, 1st ed., 1998.
[15] Y. Yan, M. Chen, and Y. Yang, "Anytime QoS Optimization over the PlanGraph for Web Service Composition," in Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12, (New York, NY, USA), pp. 1968–1975, ACM, 2012.
[16] Y. Guo, S. Chen, and Z. Feng, "Composition oriented web service semantic relations research," in Service Sciences (IJCSS), 2011 International Joint Conference on, pp. 69–73, May 2011.

Francisco Moo Mena is a Professor in Computer Sciences at Universidad Autónoma de Yucatán, in Mérida, Mexico. From the Institute National Polytéchnique de Toulouse, in France, he received a Master Degree in Computer Science and a PhD, in 2003 and 2007, respectively. He also received another Master Degree in Distributed Systems from the Instituto Tecnológico y de Estudios Superiores de Monterrey, Mexico, in 1997. He received a BS in Computer Systems Engineering from the Instituto Tecnológico de Mérida, Mexico, in 1995. His research interests include self-healing systems, Web service architectures, and semantic Web services.

Rafael Hernández Ucán received his master degree in Computer Science at Universidad Autónoma de Yucatán in 2016. He received a BS in Computer Science in the same university in 2011. His research interest include Web services, artificial intelligence and machine learning.

Víctor Uc Cetina received the PhD degree (Dr. rer. nat.) in Computer Science from Humboldt-Universität zu Berlin, Germany, in 2009; his M. Sc in Intelligent Systems from ITESM in 2002 and his BS in Computer Systems Engineering from ITM in 1999. He is currently working as an assistant professor in the Facultad de Matemáticas at Universidad Autónoma de Yucatán, México. His research interests include artificial intelligence, machine learning. bayesian reasoning and vision learning.

Francisco Madera Ramírez received his B. Sc. degree from the Universidad Autónoma de Yucatán, México; his PhD from the University of East Anglia, UK. He is the Computer Science postgraduate Chairman at the University of Yucatán. Dr. Madera teaches subjects related to computer graphics and videogames development; and his research is focused on collision detection algorithms, parallel and distributed systems and GPU programming.