Revista Cubana de Ciencias Informáticas
Vol. 10, No. 1, Enero-Marzo, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-12

Tipo de artículo: Artículo original
Temática: Inteligencia Artificial
Recibido: 14/07/2015 | Aceptado: 02/11/2015

KSAS. Algoritmo para la búsqueda por palabras clave en documentos XML

KSAS. Algorithm for keyword search in XML documents

Beatriz M. Méndez-Hernández [1*], Rosendo Moreno-Rodríguez [1], Yailen Martínez-Jiménez [1]

[1] Universidad Central "Marta Abreu" de Las Villas, Carretera a Camajuaní km. 5 ½, Villa Clara. Teléfono: 42-281515

[*] Autor para correspondencia: bmendez@uclv.edu.cu

Resumen

La búsqueda por palabras clave en documentos XML es muy útil y ampliamente usada como forma de recuperación de información. Existen diferentes algoritmos que permiten realizar esta búsqueda y otros que permiten identificar nodos significativos en documentos XML. En este artículo se explican los algoritmos más eficientes para realizar la búsqueda por palabras clave, se plantea un modelo matemático que soluciona problemas de este tipo. También se presenta un estudio de la metaheurística Optimización basada en Colonias de Hormigas y en especial del algoritmo Sistema de Hormigas. A continuación, se propone un algoritmo basado en el algoritmo Sistema de Hormigas y para finalizar se realizan pruebas estadísticas para tratar de demostrar la superioridad del algoritmo propuesto en cuanto a tiempo de ejecución sin que por ello se degrade la calidad de la solución.

Palabras clave: búsqueda por palabras clave, XML, Sistema de Hormigas, KSAS

Abstract

The keyword search in XML documents is very useful and widely used as a method of information retrieval. There are different algorithms to perform this search and others to identify significant nodes in XML documents. This article explains the most efficient algorithms for keyword searching; a mathematical model is proposed to solve such problems. It also presents a study of the Ant Colony Optimization metaheuristic, especially the Ant System algorithm. A heuristic based on the Ant System algorithm is proposed and finally statistical tests are performed in order to demonstrate the superiority of the algorithm proposed in terms of computational time without degrading the quality of the solutions.

Keywords: *keyword search, XML, Ant System, KSAS*

Introducción

Los documentos XML han llegado a convertirse en el estándar para la representación de datos y el intercambio de información en Internet. Debido a su flexibilidad estructural y su heterogeneidad resulta muy difícil para los usuarios hacer una consulta estructurada que exprese lo que el usuario necesita. Actualmente, la búsqueda por palabras clave en documentos XML se ha convertido en un popular paradigma para la recuperación de información en documentos XML (Hristidis V., 2003; Li et al., 2007; Liu and Chen, 2007; Sun et al., 2007; Bao et al., 2009). La búsqueda por palabras clave en documentos XML resulta significativa por su simplicidad, ya que los usuarios no necesitan aprender un complejo lenguaje de consulta (XPath) o conocer la estructura de los datos subyacentes. Sin embargo, este tipo de consulta puede ser imprecisa o retornar un gran número de nodos que no son de interés para el usuario, algunos son simplemente nodos que enlazan las palabras clave que el usuario necesita.

Por esto, la efectividad en cuanto a la relevancia del resultado es la parte más crucial de la búsqueda por palabras clave en documentos XML y puede resumirse en tres problemas esenciales. 1) Se debe poder identificar eficazmente el tipo de nodo o nodos que se intentan encontrar con esta consulta. 2) Se debe poder inferir eficazmente los tipos de nodos que se quieren encontrar con esta consulta. 3) Se debe evaluar cada resultado de la consulta en consideración con los dos problemas anteriores (Bao et al., 2009).

La mayoría de los algoritmos que se han desarrollado para realizar este tipo de búsqueda la realizan de forma exhaustiva en el XML y se basan en diferentes semánticas como Ancestro Común más Cercano, Menor Ancestro Común más Cercano, Exclusivo Ancestro Común más Cercano y Ancestro Común más Cercano Compacto (LCA, SLCA, ELCA y CLCA respectivamente, por sus siglas en inglés), siendo la segunda semántica la que mejores resultados ha generado (Xu Yu, 2010).

El primer objetivo de este trabajo es plantear la búsqueda por palabras clave en documentos XML como un modelo matemático. El segundo objetivo es proponer un algoritmo que realice la búsqueda por palabras clave basado en la metaheurística Optimización por Colonias de Hormigas (OCH), específicamente en el algoritmo Sistema de Hormigas (SH), evitándose así una búsqueda exhaustiva en el árbol. Y, por último, evaluar el nuevo algoritmo comparándolo con uno de los algoritmos exhaustivos en cuanto al tiempo en que realiza la búsqueda y calidad de la solución.

Algoritmos basados en SLCA

La mayoría de los algoritmos que realizan la búsqueda por palabras clave son exhaustivos y están basados en las diferentes semánticas de árboles que existen. De todos estos algoritmos según la literatura, los que actúan de manera más eficiente son los basados en la semántica SLCA.

El algoritmo *Stack* (Guo et al., 2003) no es más que una variante del algoritmo de mezcla ordenada que utiliza una pila para procesar todos los SLCAs. La idea general del mismo es el uso de una pila para simular el recorrido postorden de un árbol XML virtual formado por la unión de los caminos de la raíz a cada nodo $S_1, \ldots, S_l$, mientras que los nodos son leídos en preorden. Cuando una entrada está en el tope de la pila, lo cual significa que todos sus descendientes en $S_1, \ldots, S_l$ han sido visitados, se sabe si la palabra clave aparece o no en el subárbol. Este algoritmo hace una lista de todas las palabras clave y procesa el prefijo común más largo del nodo que mejor ID tenga.

El algoritmo comienza creando una pila vacía y se tiene una lista de ID de los nodos que no han sido visitados, se lee el próximo nodo con el ID más pequeño y se realizan las operaciones necesarias. En esencia, el orden con que son leídos estos nodos es equivalente a un preorden de un árbol XML ignorando los nodos irrelevantes.

El algoritmo *Stack* trata todos los conjuntos de entrada $S_1, \ldots, S_l$ como si fueran iguales pero algunas veces la cantidad de elementos de estas listas varía drásticamente. Por esto se crea el algoritmo *Indexed Lookup Eager* (ILE) (Xu and Papakonstantinou, 2005), que trata el caso en que $|S_1|$ es muy diferente del $|S|$. La complejidad temporal de este algoritmo es $O\ (|S_l| \sum_{i=2}^{l} d * \log |S_i| + |S_l| d * \log |S_l|)$ ó $O\ (|S_l| ld * \log |S|)$.

El algoritmo *Scan Eager* (SE) modifica al algoritmo ILE porque utiliza un escaneo lineal para encontrar $l_m\ ()$ y $r_m\ ()$. Es decir, se aprovecha de que los accesos a cualquier lista de palabras clave son estrictamente en orden creciente en al algoritmo ILE.

SE mantiene un puntero para cada lista de palabras clave y el cursor avanza de S_2 hasta encontrar el nodo que está más cerca de *v* ya sea por la derecha o por la izquierda. Para asegurar el buen funcionamiento de este algoritmo nótese que *p* tiene que ser no más pequeño que $|S_1|$. La complejidad temporal de SE es un $O\ (ld |S_1| + d \sum_{i=2}^{l} |S_i|)$ ó $O\ (ld |S|)$ (Xu and Papakonstantinou, 2005). Como puede verse la complejidad temporal de SE es la misma que la del algoritmo ILE.

Para concluir con algoritmos que realizan la búsqueda basados en la semántica SLCA se describe el algoritmo *Multiway-SLCA* (Sun et al., 2007) que mejora la actuación del algoritmo ILE pero mantiene la misma complejidad temporal para el peor caso (Xu and Papakonstantinou, 2005). De este algoritmo existen dos variantes, *Basic Multiway-SLCA* (BMS) e *Incremental Multiway-SLCA* (IMS) propuestas por (Sun et al., 2007), el pseudocódigo del segundo se muestra en la figura 1. El algoritmo IMS introduce además una optimización con el objetivo de reducir el

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 1, Enero-Marzo, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-12

número de computaciones de *lca* que realiza el algoritmo BMS. BMS necesita recuperar los nodos en orden desde un conjunto no ordenado y esto trae consigo un gasto de tiempo extra. Por lo que de estas dos variantes analizaremos la variante optimizada, IMS. El algoritmo tiene como entrada una lista de palabras clave *(S_1, …, S_l)* y la salida son todos los SLCAs.

```
1: let v_m = last({first(S_i) | i ∈ [1,k]}), where v_m ∈ S_m
2: initialize n = 1;   α_1 = d_root
3: while (v_m ≠ null) do
4:    if (m ≠ 1) then
5:       v_1 = closest(v_m, S_1)
6:       if (v_m ≺_p v_1) then
7:          v_m = v_1
8:       end if
9:    end if
10:   P = {pred(v_m, S_i) | i ∈ [1,k], i ≠ m} ∪ {v_m}
11:   N = {next(v_m, S_i) | i ∈ [1,k], next(v_m, S_i) ≠ null}
12:   initialize r_max = last(N);    r = v_m
13:   repeat
14:      remove ℓ from P, where ℓ = first(P)
15:      α = lca(ℓ, r)
16:      r = last(r, v) where v ∈ N s.t. v = next(v_m, S_j), ℓ ∈ S_j
17:   until (r = null) or (α ⋠_a r) or (r = r_max)
18:   if (r = null) or (α ⋠_a r) then
19:      if (α_n ⪯_a α) then
20:         α_n = α
21:      else if (α ⋠_a α_n) then
22:         n = n + 1;   α_n = α
23:      end if
24:      v_m = last(r, out(α_n, S_1), ··· , out(α_n, S_k))
25:   else
26:      v_m = r
27:   end if
28: end while
29: if (α_1 = d_root) then return ∅ else return {α_1, ··· , α_n}
```

Figura 1. Pseudocódigo del algoritmo IMS.

Este algoritmo computa SLCA iterativamente, en cada iteración se selecciona un nodo ¨ancla¨ v_m que se almacena al igual que su índice *m*. IMS se optimiza básicamente en las líneas 7-8. Nótese que cada llamada a la función *closest* requiere dos computaciones LCA. IMS reduce el número de computaciones LCA a un máximo de L, es decir como máximo una por cada lista de palabras clave. BMS e IMS dan como salida todos los nodos SLCA en un tiempo $O(|S_1| ld \log |S|)$.

Optimización basada en Colonias de Hormigas

Los algoritmos de Optimización basada en Colonias de Hormigas (OCH) (Dorigo and Di Caro, 1999, Dorigo et al., 1999, Dorigo and Stützle, 2003, Dorigo and Birattari, 2010) se inspiran directamente en el comportamiento de las colonias reales de hormigas para solucionar problemas de optimización combinatoria. Se basan en una colonia de hormigas artificiales; es decir, agentes computacionales simples que trabajan de manera cooperativa y se comunican mediante rastros artificiales de feromona. Los algoritmos de OCH son esencialmente métodos constructivos: en cada iteración del algoritmo, cada hormiga construye una solución al problema recorriendo un grafo de construcción *G1*. La información heurística, mide la preferencia heurística de moverse desde el nodo *r* hasta el nodo *s*; es decir, la propensión a recorrer la arista a_{rs} y se denota por η_{rs}. Las hormigas no modifican esta información durante la ejecución del algoritmo.

La información de los rastros artificiales de feromona, mide la "deseabilidad aprendida" del movimiento de *r* a *s*. Imita de forma numérica a la feromona real que depositan las hormigas naturales. Esta información se modifica durante la ejecución del algoritmo dependiendo de las soluciones encontradas por las hormigas y se denota por τ_{rs}.

El algoritmo Sistema de Hormigas (Dorigo M., 1996, Heinonen and Pettersson, 2007), fue el primer algoritmo de OCH. Su versión actual (*Ant Cycle*) apareció conjuntamente con otras variantes de éste. El SH se caracteriza por el hecho de que, la actualización de feromona se realiza una vez que todas las hormigas han completado sus soluciones, y se lleva a cabo como sigue: primero, todos los rastros de feromona se reducen en un factor constante, implementándose la evaporación de feromona según la siguiente ecuación:

$$t_{ij} \leftarrow (1-\rho)\cdot t_{ij},\ \rho \in (0,1]$$

donde ρ se conoce como constante de evaporación y es la encargada de reducir los rastros de feromona para evitar el estancamiento de las soluciones y t_{ij} es la cantidad de feromona asociada al arco a_{ij}.

A continuación, cada hormiga de la colonia deposita una cantidad de feromona en función de la calidad de su solución, según la ecuación:

$$t_{ij} \leftarrow t_{ij} + \Delta t^k \quad \forall a_{ij} \in S^k$$

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 1, Enero-Marzo, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-12

donde $\Delta t^k = f(C(S^k))$, representa la cantidad de feromona a depositar por la hormiga *k* en cada arco a_{ij} de su solución encontrada (S^k). Este valor depende de la calidad de dicha solución ($C(S^k)$).

Las soluciones en el SH se construyen como sigue: en cada paso de construcción, una hormiga *k* escoge ir al siguiente nodo con una probabilidad que se calcula como:

$$p_{ij}^k = \frac{(\tau_{ij})^\alpha \cdot (\eta_{ij})^\beta}{\sum_{j \in N_i^k} (\tau_{ij})^\alpha \cdot (\eta_{ij})^\beta} \qquad si \quad j \in N_i^k$$

donde N_i^k es el vecindario alcanzable por la hormiga *k* cuando se encuentra en el nodo *i*. Los parámetros alfa *(α)* y beta *(β)* controlan el proceso de búsqueda. Para α=0 se tiene una búsqueda heurística estocástica clásica, mientras que para *β=0* sólo el valor de la feromona tiene efecto. Un valor de *α<1* lleva a una rápida situación de convergencia (*stagnation*) (Dorigo et al., 1999). El vector P_{ij}^k contiene las probabilidades de movimiento calculadas para los nodos de la vecindad (N_i^k) de la hormiga *k*. El valor τ_{ij} representa el elemento *(i, j)* en la matriz de feromona y η_{ij} se denomina función de visibilidad o función heurística y mide la calidad de un estado *j* a partir del *i*.

Modelo Matemático

La literatura recomienda que la recuperación de información estructurada, como es la información que nos brindan los XML, se debe regir por el principio de recuperación de información estructurada: "Un sistema de recuperación de información deberá siempre recuperar la parte más específica del documento que dé respuesta a la consulta" (Manning et al., 2008).

Este principio se considera muy difícil de implementar algorítmicamente, ya que puede presentar cierta ambigüedad debido a que dos nodos pueden tener la misma etiqueta y representar dos cosas totalmente diferentes. Por ejemplo, un creador y el producto que este creó pueden tener el mismo nombre y ambas serían buenas respuestas para el usuario.

Para dar solución a esto para documentos XML existe una fórmula que permite calcular el peso de cada subárbol de acuerdo a las palabras clave de entrada y así evaluar todos los posibles resultados. Esta fórmula es conocida como función *context resemblance* (C_R) (Manning et al., 2008):

$$C_R = \begin{cases} \frac{1+|C_q|}{1+|C_d|}, si\_C_q\_coincide\_con\_C_d \\ 0, \_en\_otros\_casos \end{cases}$$

donde $|C_q|$ representa el número de nodos que están en la consulta y $|C_d|$ el número de nodos del documento. El valor de C_R es 1 si y solo si la cantidad de palabras clave coincide con el número de nodos del documento XML, la cual se consideraría la respuesta perfecta.

A continuación, se plantea un problema usando como base el principio de recuperación de información estructurada y la función vista anteriormente.

Dado un documento XML *V*, con *n* elementos (nodos), donde V_i son los fragmentos (subárboles) del documento que contienen todas las palabras clave o palabras de entrada, $|C_q|$ es la cantidad de palabras de entrada y $|C_{di}|$ la cantidad de nodos del fragmento (subárbol) *i*, el problema nos quedaría de la siguiente manera:

$$\forall q \in C_q, \exists d \in C_{di} : q = d$$

Después de que esto se garantice, el problema a resolver no sería más que:

$$\max \frac{1+|C_q|}{1+|C_{di}|}$$
$$sujeto\_a : |C_{di}| \geq |C_q|$$

De esta forma se garantiza que se escoja el subárbol que contenga todas las palabras clave y al mismo tiempo el que menor cantidad de nodos tenga.

KSAS

La esencia del algoritmo KSAS es la búsqueda de palabras clave en documentos XML usando el algoritmo SH de la metaheurística OCH.

El algoritmo SH realiza una búsqueda en un determinado ambiente, generalmente sobre grafos no dirigidos y completos, estos grafos son su espacio de búsqueda que van explotando hasta encontrar la mejor solución o al menos una bastante buena.

Para la aplicación de este algoritmo a nuestro problema se hizo necesario realizar una serie de modificaciones debido a que el espacio de búsqueda donde se ejecuta la explotación es la información del documento XML, la cual se representa en forma de árbol como fue descrito en el planteamiento del problema. A continuación, se muestra en la figura 2 un documento XML y el árbol que lo representa.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 1, Enero-Marzo, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-12

```
<?xml version="1.0" encoding=UTF-8"?>
<doc>
   <customer id='123'>
      <name>...</name>
      <address>...</address>
       ...
      <order>
         <amount>...</amount>
            <date>...</date>
         <item quant='12'>
            <name>...</name>
         </item>
         <item quant='4'>...</item>
          ...
      </order>
      <order>...</order>
       ...
      <payment>
         <number>...</number>
         <date>...</date>
      </payment>
      <payment>>...</payment>
       ...
   </customer>
   <customer id='124'>...</customer>
</doc>
```

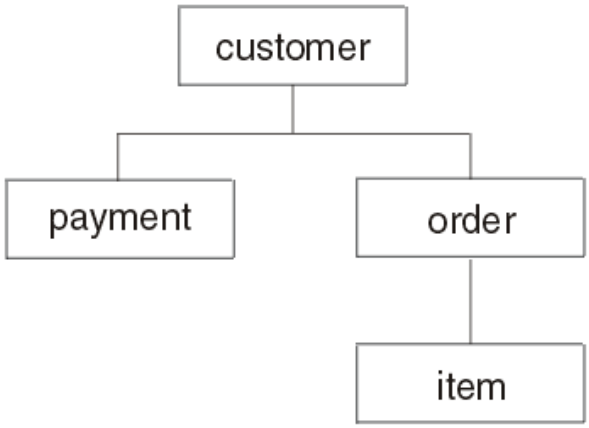

Figura 2. Ejemplo de XML y su árbol correspondiente.

El algoritmo KSAS fue implementado en Java y para trabajar con los documentos XML se utilizó la biblioteca JDOM, la cual facilita el manejo de la información del documento.

El proceso comienza posicionando cada una de las hormigas de la colonia en un elemento del documento XML escogido aleatoriamente, cada hormiga empieza a moverse por el árbol según las funciones de proporcionalidad utilizadas usualmente por el algoritmo SH, para resolver este problema la función heurística es la función objetivo del problema de optimización planteado anteriormente, la vecindad del nodo *i* son todos los hijos de este nodo y su padre, ya que estos son los únicos elementos relacionados con el nodo *i*, por tanto es adonde único puede moverse la hormiga. Luego de ser elegido el nodo al cual va a moverse la hormiga se introduce en el recorrido de la misma, si es que este nodo no ha sido visitado, en caso contrario, se inserta este nodo en una variable temporal, luego se procede a incrementar los valores de feromona con una actualización en línea, este proceso lo desarrollan todas las hormigas de la colonia. Cada hormiga termina cuando tiene en su recorrido todas las palabras clave que se buscaban o visitó todos los estados (nodos) sin encontrar todas las palabras clave. Siempre que se obtengan soluciones estas son evaluadas para tener almacenada la mejor solución, es decir, la menor cantidad de nodos que contengan todas las palabras clave (mejor solución global). Cuando cada hormiga de la colonia encontró su solución, se pasa a la evaporación de los

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 1, Enero-Marzo, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-12

rastros de feromona pertenecientes a la mejor solución encontrada hasta el momento, además se realiza una actualización de los rastros de la feromona asociados a esta solución.

Algorithm 1 Algoritmo KSAS

Inicializar parmetros (nc, $cantidad\_Hormigas$, $L_{global} = documento$, $t(i) = t_0$, palabras_clave, documento)
while $nc > 0$ **do**
  Para cada hormiga eligir elemento inicial
  while $cantidad\_palabras\_hormiga < cantidad\_palabras\_claves$ **do**
    foreach ($hormiga_i$) **do**
    Elegir próximo nodo a ser visitado
    End foreach
    Mover la hormiga al próximo nodo
    Insertar nodo seleccionado si no ha sido visitado e insertarlo en arreglo temporal
    Foreach ($hormiga_i$) **do**
    Evaporar los rastros de feromona
    End foreach
  end while
  for int i=0; $i < k$; i++ **do**
    Calcular costo de la solución (L_k)
  end for
  if $L_k > L_{global}$ **then**
    $L_k = L_{global}$
    Incrementar los valores de feromona asociados a L_k
  end if
end while
Imprimir el menor subárbol L_{global}

Figura 3. Pseudocódigo del algoritmo KSAS.

Resultados experimentales

Para validar el algoritmo se utilizaron 60 XML de la colección de INEX. El algoritmo se ejecutó en una computadora con procesador i5 a 3.30 GHz, con 4 GB de memoria RAM y sistema operativo Linux.

Para ver si el algoritmo KSAS funciona mejor que el algoritmo IMS se tratará de demostrar que el algoritmo KSAS mejora de forma significativa el tiempo de ejecución del algoritmo IMS pero que al mismo tiempo la calidad de la solución no experimente cambios significativos.

Para esto se les aplicó a los algoritmos la prueba de Wilcoxon para comparación de dos poblaciones, los resultados obtenidos se muestran a continuación. Estas pruebas se hicieron para tres dimensiones de arreglos de palabras clave a buscar (2, 3 y 10).

Tabla 1. Prueba de Friedman aplicada a los algoritmos IMS y KSAS

Variables	R^+	R^-	Significación
Calidad de la solución para 2 palabras clave (IMS-KSAS)	25	35	0.532
Calidad de la solución para 3 palabras clave (IMS-KSAS)	21	39	0.112
Calidad de la solución para 10 palabras clave (IMS-KSAS)	26	34	0.530
Costo computacional para 2 palabras clave (IMS-KSAS)	60	0	0.00
Costo computacional para 3 palabras clave (IMS-KSAS)	60	0	0.00
Costo computacional para 10 palabras clave (IMS-KSAS)	60	0	0.00

Como puede verse en cuanto a la calidad de solución no se encontraron diferencias significativas, ya que en todos los casos el algoritmo propuesto fue capaz de encontrar la solución óptima, por lo tanto, se puede decir que el comportamiento de los dos algoritmos en cuanto a este punto es similar. En cuanto al costo computacional, si se encontraron diferencias significativas, mejorando el algoritmo propuesto al IMS en todos los casos analizados pues como puede observarse en la tabla en todos los casos el tiempo computacional del algoritmo IMS es mayor que el tiempo del algoritmo KSAS.

Conclusiones

Se diseñó un modelo matemático para resolver la búsqueda de palabras clave en documentos XML, a partir de este modelo se implementó el algoritmo KSAS utilizando el comportamiento del algoritmo SH que realiza esta búsqueda teniendo como función heurística la función objetivo del modelo matemático diseñado; y por último se demostró que el algoritmo KSAS soluciona este problema de forma óptima ya que no muestra diferencias significativas en la calidad de las soluciones, lo cual quiere decir que es capaz de encontrar en la mayoría de los casos el subárbol óptimo, y si existen diferencias significativas en el costo computacional, donde para el total de los casos el algoritmo propuesto muestra un menor costo que el algoritmo IMS.

Referencias

BAO, Z., LING, T. W., CHEN, B. & LU, J. 2009. Effective XML Keyword Search with Relevance Oriented Ranking. *Proceedings of the 2009 IEEE International Conference on Data Engineering.* IEEE Computer Society.

DORIGO, M. 1992. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*.

DORIGO, M. & BIRATTARI, M. 2010. Ant colony optimization. *Encyclopedia of Machine Learning.* Springer.

DORIGO, M. & DI CARO, G. Ant colony optimization: a new meta-heuristic. Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, 1999. IEEE.

DORIGO, M., DI CARO, G. & GAMBARDELLA, L. M. 1999. Ant algorithms for discrete optimization. *Artificial life,* 5**,** 137-172.

DORIGO, M. & STÜTZLE, T. 2003. The ant colony optimization metaheuristic: Algorithms, applications, and advances. *Handbook of metaheuristics.* Springer.

DORIGO M., M. V. A. C. A. 1996. Ant system: optimization by a colony of cooperating agents *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on,* 26**,** 29-41.

GUO, L., SHAO, F., BOTEV, C. & SHANMUGASUNDARAM, J. 2003. XRANK: ranked keyword search over XML documents. *Proceedings of the 2003 ACM SIGMOD international conference on Management of data.* San Diego, California: ACM.

HEINONEN, J. & PETTERSSON, F. 2007. Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. *Applied Mathematics and Computation,* 187**,** 989-998.

HRISTIDIS V., P. Y. A. B., A. 2003. Keyword proximity search on XML graphs *Proceedings. 19th International Conference on Data Engineering.* IEEE.

LI, Y., CHAUDHURI, I., YANG, H., SINGH, S. & JAGADISH, H. DaNaLIX: a domain-adaptive natural language interface for querying XML. Proceedings of the 2007 ACM SIGMOD international conference on Management of data, 2007. ACM, 1165-1168.

LIU, Z. & CHEN, Y. Identifying meaningful return information for XML keyword search. Proceedings of the 2007 ACM SIGMOD international conference on Management of data, 2007. ACM, 329-340.

MANNING, C. D., RAGHAVAN, P. & SCHTZE, H. 2008. *Introduction to Information Retrieval*, Cambridge University Press.

SUN, C., CHAN, C.-Y. & GOENKA, A. K. 2007. Multiway SLCA-based keyword search in XML data. *Proceedings of the 16th international conference on World Wide Web*. Banff, Alberta, Canada: ACM.

XU, Y. & PAPAKONSTANTINOU, Y. 2005. Efficient keyword search for smallest LCAs in XML databases. *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. Baltimore, Maryland: ACM.

XU YU, J., QIN L. AND CHANG L. 2010. *Keyword Search in Databases*, Morgan & Claypool.