DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

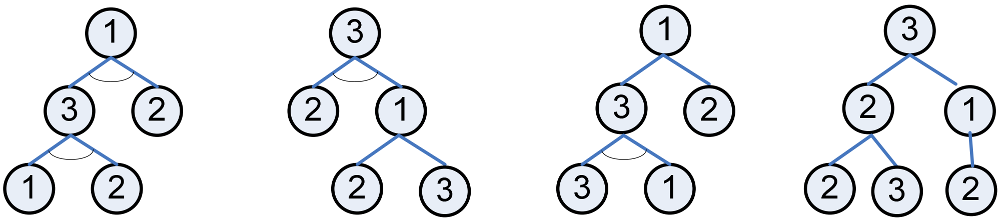
Patrones en árboles

© Fernando Berzal, berzal@acm.org

Patrones en árboles

- Tipos de árboles
- Tipos de subárboles / patrones en árboles
- POTMiner [Partially-Ordered-Tree Miner]
- Algoritmos
- Aplicaciones

Tipos de árboles



Árbol ordenado

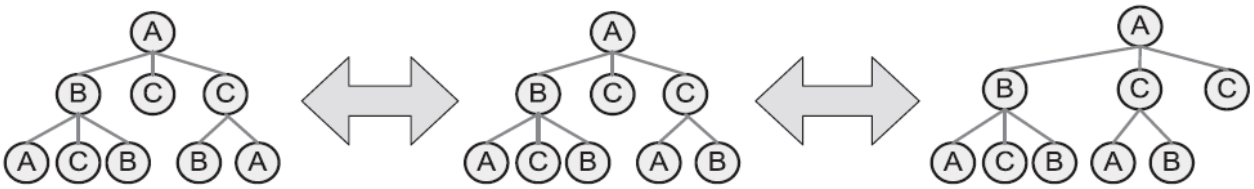
Árboles parcialmente ordenados

Árbol no ordenado

Tipos de árboles

Ejemplo

3 representaciones alternativas del mismo árbol [no ordenado]

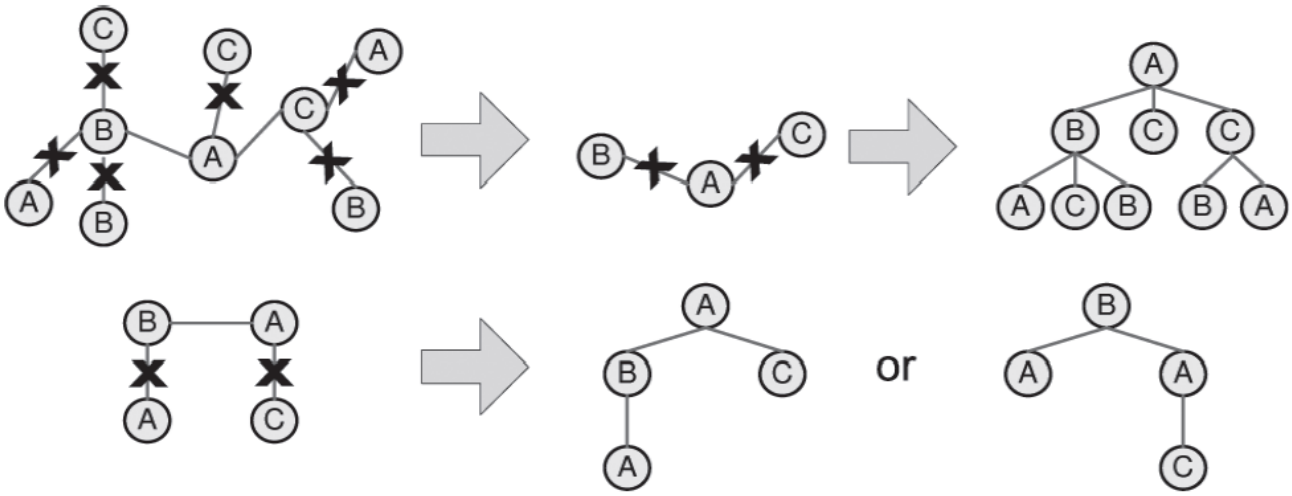


Codificación del árbol

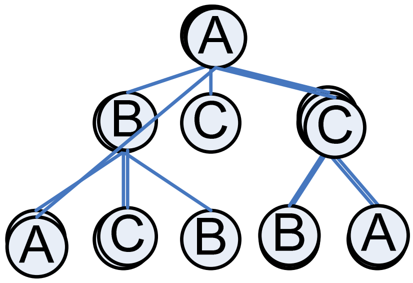
- DFS [Depth-first]: A C B ↑ A ↑ ↑ B
- BFS [Breadth-first]: A $ C B $ B A
- Depth sequence: (0,A) (1,C) (2,B) (2,A) (1,B)

Tipos de árboles

“Free trees” (sin raíz asignada)

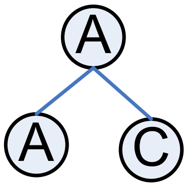


Tipos de subárboles



Original tree

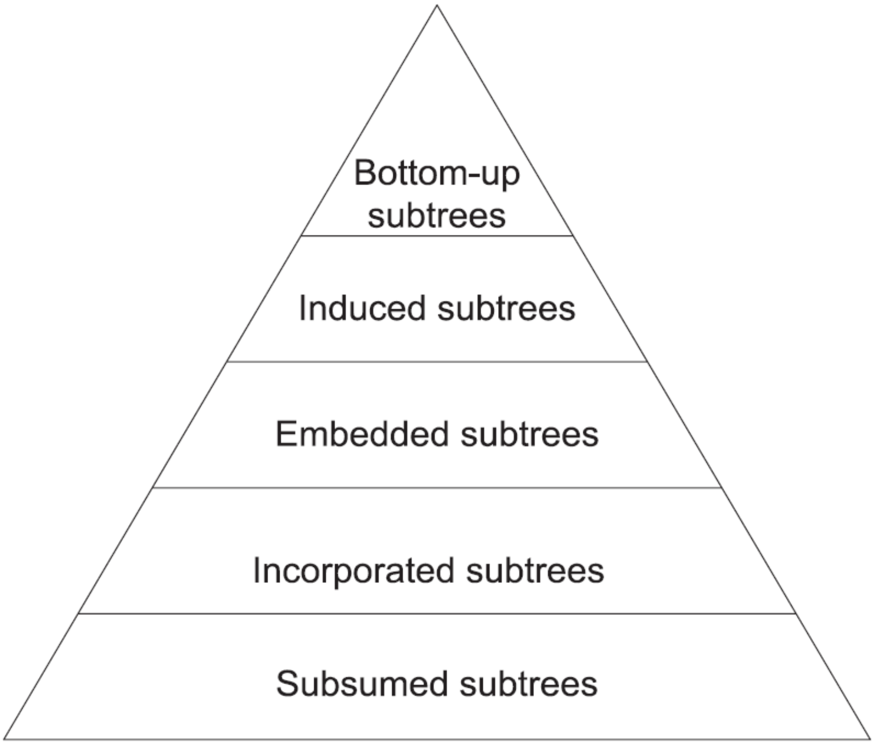
Bottom-up subtree



Induced subtree

Embedded subtree

Tipos de subárboles



Bottom-up subtrees

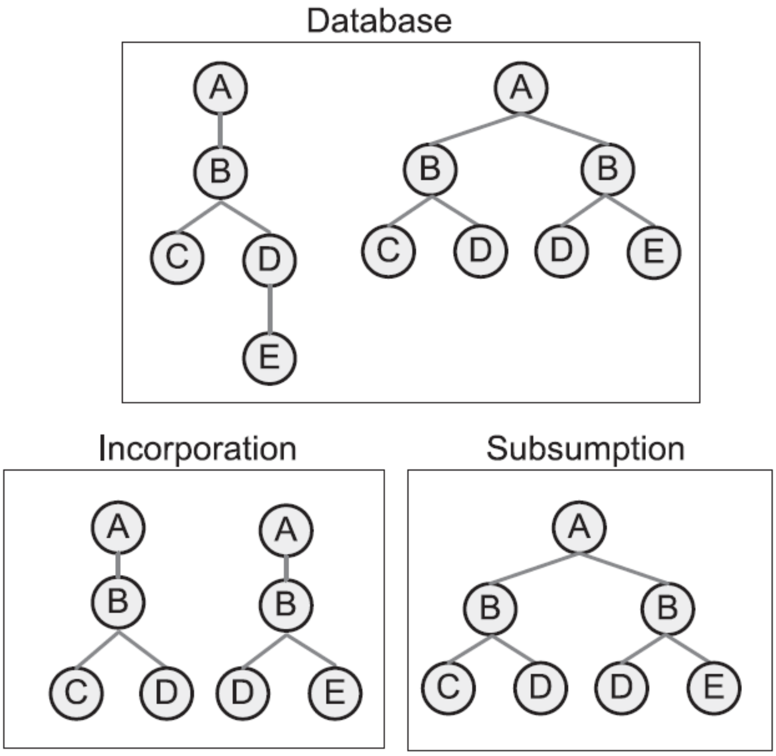
Induced subtrees

Embedded subtrees

Incorporated subtrees

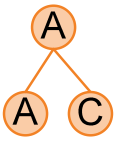
Subsumed subtrees

Tipos de subárboles

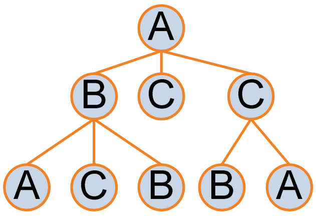


Tipos de subárboles

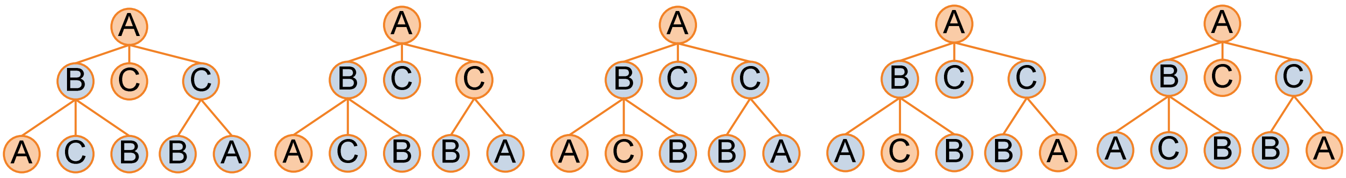
Embedded subtree



Original tree



Ocurrencias del patrón en el árbol original



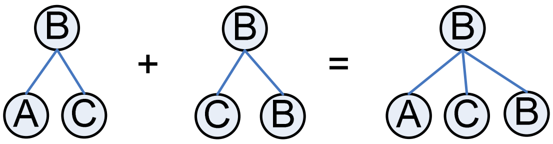
POTMiner

Partially-Ordered Tree Miner

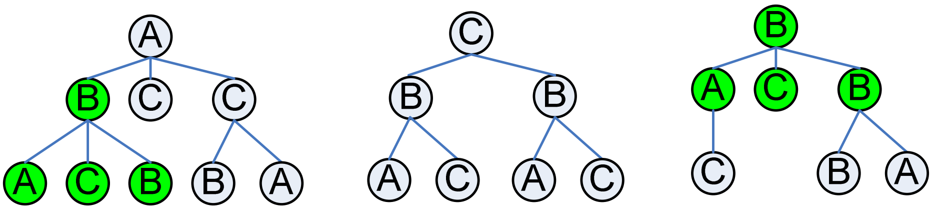
Jiménez, Berzal & Cubero (KAIS'2009)

Algoritmo tipo Apriori:

- Generación de candidatos



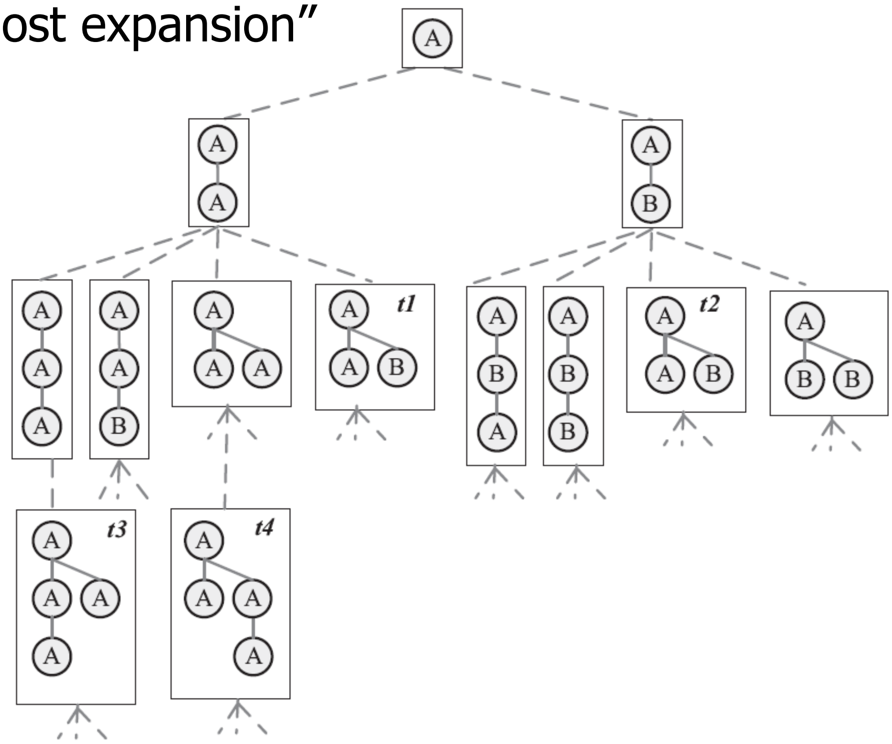
- Conteo del soporte



POTMiner

Generación de candidatos

"Rightmost expansion"



POTMiner

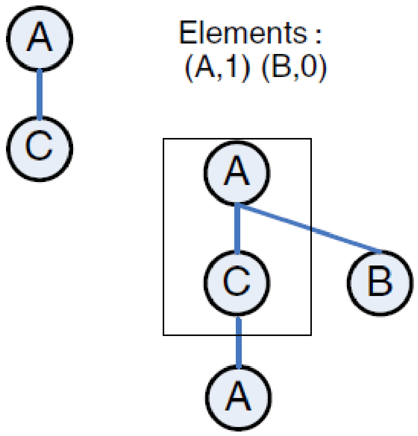
Generación de candidatos

Diferentes estrategias

- "Rightmost expansion" (FreqT)
  - TMG [Tree-Model-Guided] candidate enumeration
  - … con secuencias de profundidad (Unot, uFreqt, Gaston, TRIPS)
- Extensión basada en clases de equivalencia (TreeMiner, SLEUTH, POTMiner, RETRO, Phylominer)
- "Right-and-left" tree join (AMIOT)
- "Extension and join" (HybridTreeMiner)

POTMiner

Clases de equivalencia

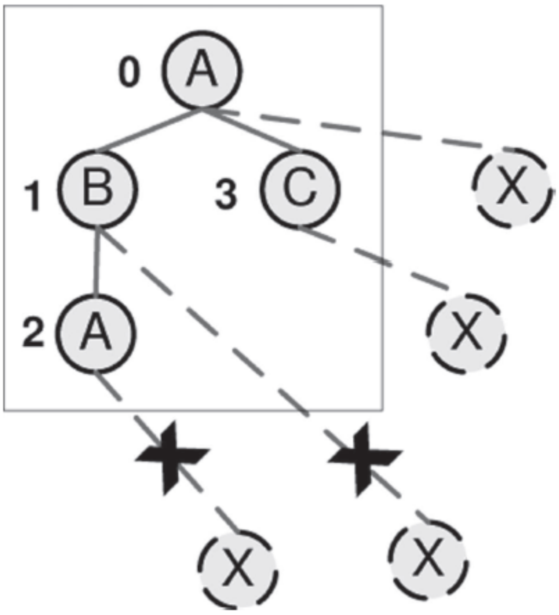


Clase de equivalencia
con dos elementos, ACA y AC↑B
que comparten el prefijo AC

POTMiner

Generación de candidatos

Clases de equivalencia



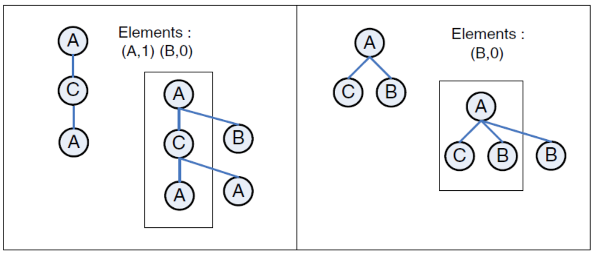
IDEA
Generar todos los patrones posibles, pero sin generar patrones por duplicado…

POTMiner

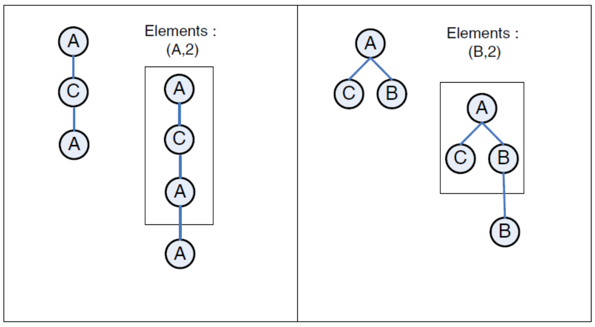
Generación de candidatos

Clases de equivalencia

- “Cousin extension” (en anchura)



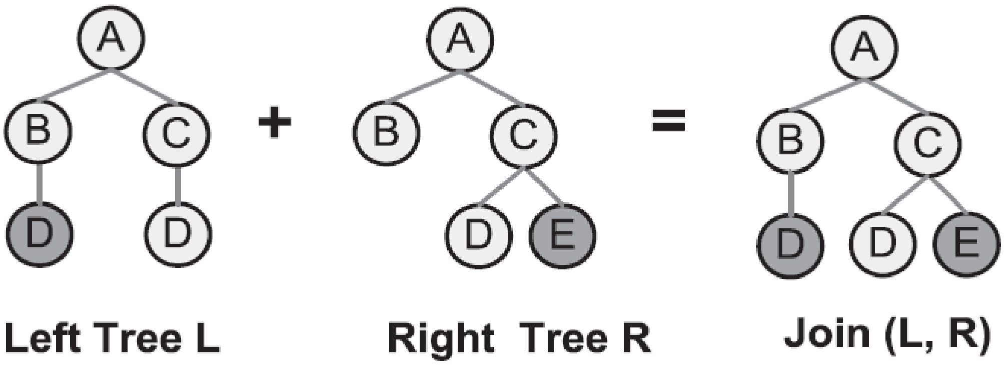
- “Child extension” (en profundidad)



POTMiner

Generación de candidatos

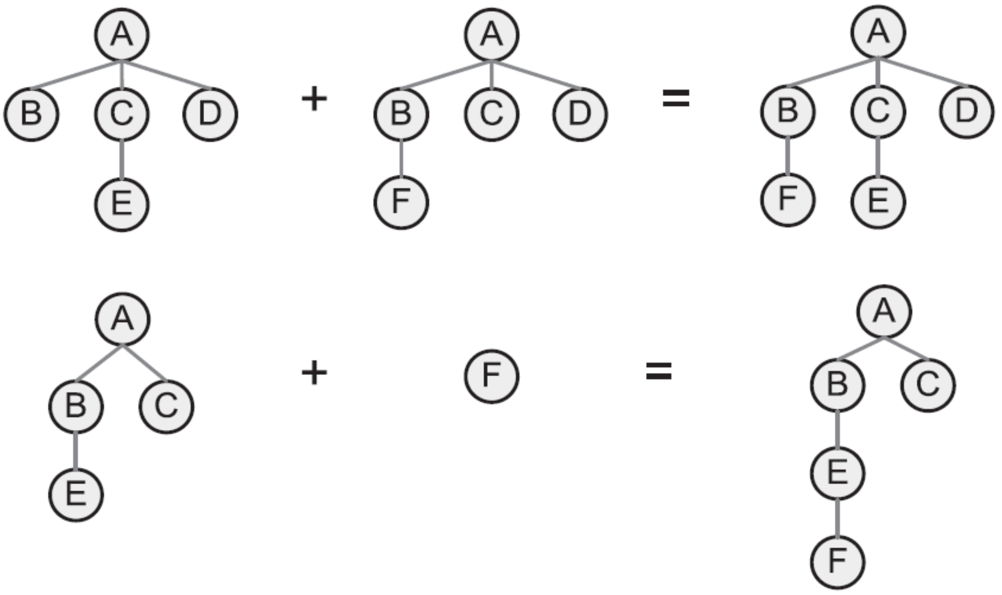
“RL Tree Join”



POTMiner

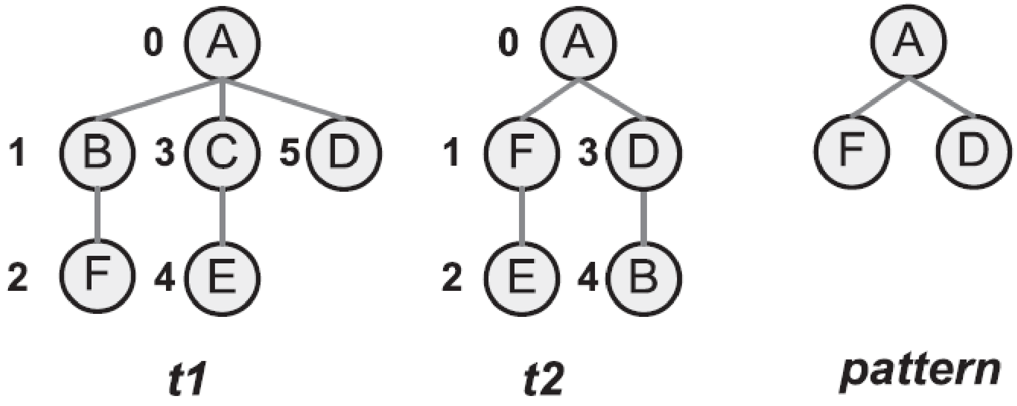
Generación de candidatos

"Extension and join"



POTMiner

Cálculo del soporte

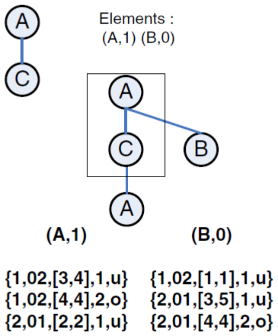


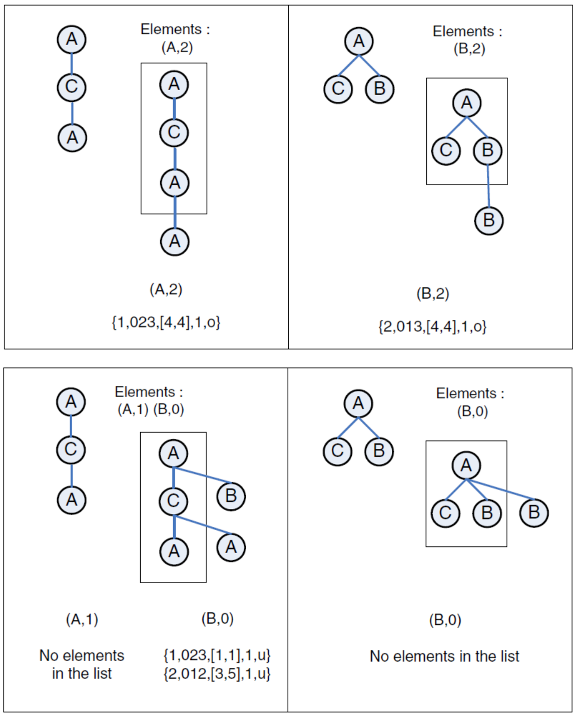
- Listas de ocurrencias **(tid, i_1, i_2… i_k)**
- Listas de ámbitos **(tid, m, s)**

POTMiner

Cálculo del soporte

Reunión de listas de ámbitos





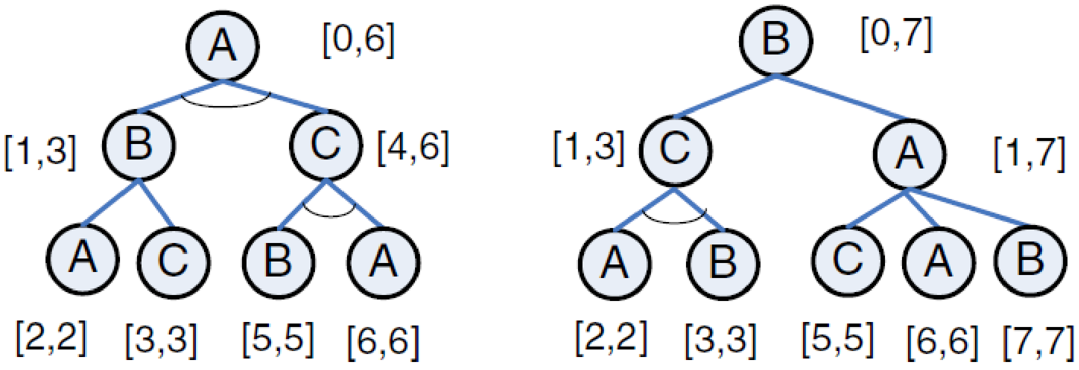
In-scope join
(child extension)

Out-scope join
(cousin extension)

POTMiner

Ejemplo

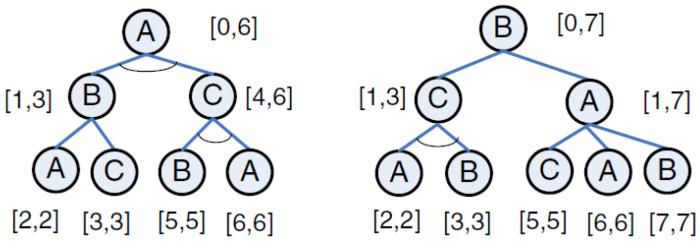
Conjunto de datos



POTMiner

Ejemplo

Patrones de tamaño 1 (“representación vertical”)



{1,_,[0,6],0,_}
{1,_,[2,2],2,u}
{1,_,[6,6],2,o}
{2,_,[2,2],2,o}
{2,_,[1,7],1,u}
{2,_,[6,6],2,u}

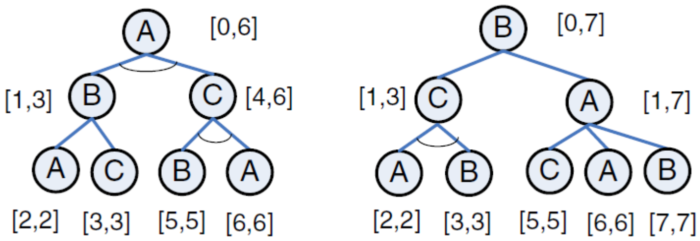
{1,_,[1,3],1,_}
{1,_,[5,5],2,o}
{2_,[0,7],0,_}
{2,_,[3,3],2,o}
{2,_,[7,7],2,u}

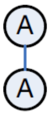
{1,_,[4,6],1,o}
{1,_,[3,3],2,u}
{2,_,[1,3],1,u}
{2,_,[5,5],2,u}

POTMiner

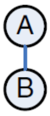
Ejemplo

Clases de equivalencia derivadas de los patrones de tamaño 1

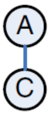




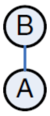
{1,0,[2,2],2,u}
{1,0,[6,6],2,o}
{2,4,[6,6],2,u}



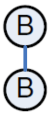
{1,0,[1,3],1,o}
{1,0,[5,5],2,o}
{2,4,[7,7],1,u}



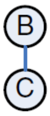
{1,0,[4,6],1,o}
{1,0,[3,3],2,u}
{2,4,[5,5],1,u}



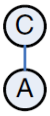
{1,1,[2,2],1,u}
{2,0,[4,7],1,u}
{2,0,[6,6],2,u}



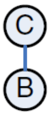
{2,0,[3,3],2,o}
{2,0,[7,7],2,u}



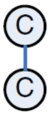
{1,1,[3,3],1,u}
{2,0,[1,3],1,u}
{2,0,[5,5],2,u}



{1,4,[6,6],1,o}
{2,1,[2,2],1,o}



{1,4,[5,5],1,o}
{2,1,[3,3],1,o}

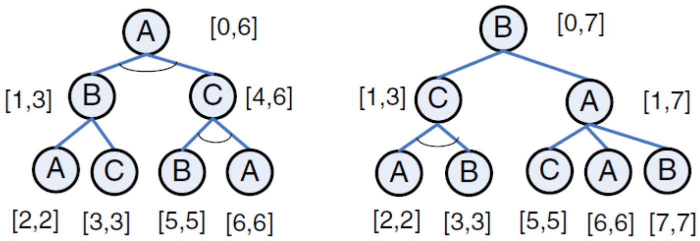


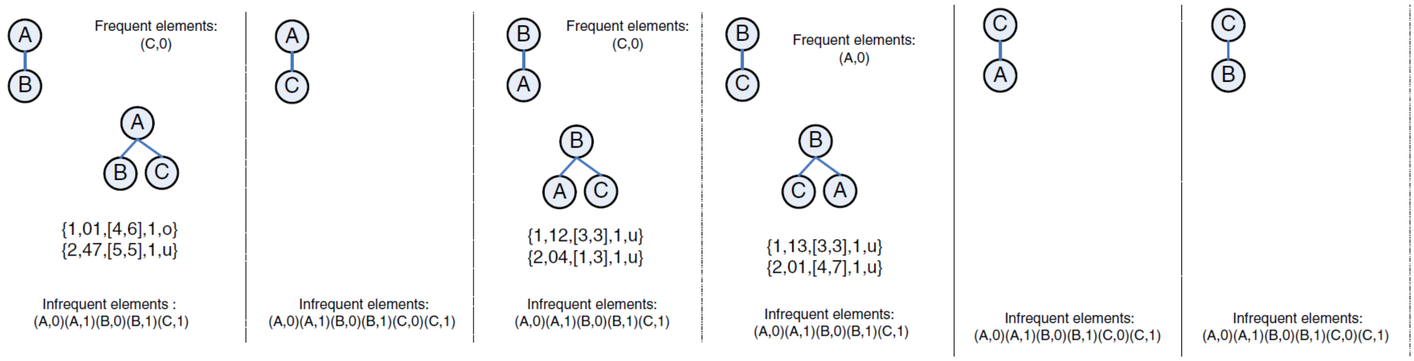
No elements in the scope list

POTMiner

Ejemplo

Clases de equivalencia derivadas de los patrones de tamaño 2





Frequent elements:
(C,0)

{1,01,[4,6],1,o}
{2,47,[5,5],1,u}

Infrequent elements :
(A,0)(A,1)(B,0)(B,1)(C,1)

Infrequent elements:
(A,0)(A,1)(B,0)(B,1)(C,0)(C,1)

Frequent elements:
(C,0)

{1,12,[3,3],1,u}
{2,04,[1,3],1,u}

Infrequent elements:
(A,0)(A,1)(B,0)(B,1)(C,1)

Frequent elements:
(A,0)

{1,13,[3,3],1,u}
{2,01,[4,7],1,u}

Infrequent elements:
(A,0)(A,1)(B,0)(B,1)(C,1)

Infrequent elements:
(A,0)(A,1)(B,0)(B,1)(C,0)(C,1)

Infrequent elements:
(A,0)(A,1)(B,0)(B,1)(C,0)(C,1)

POTMiner

Algoritmo basado en clases de equivalencia

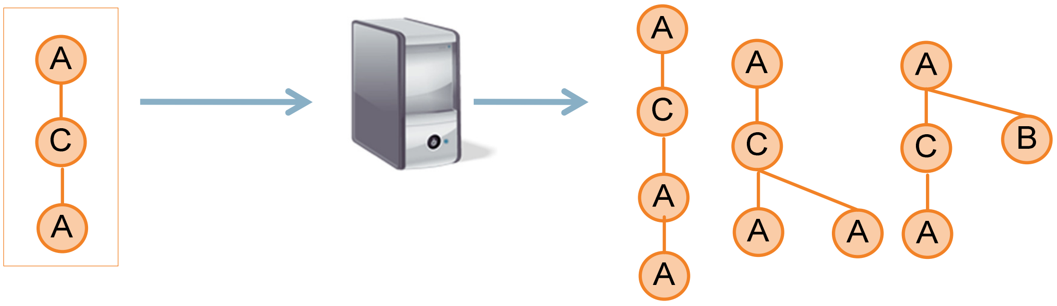
(como SPADE para secuencias, TreeMiner/Sleuth para árboles)

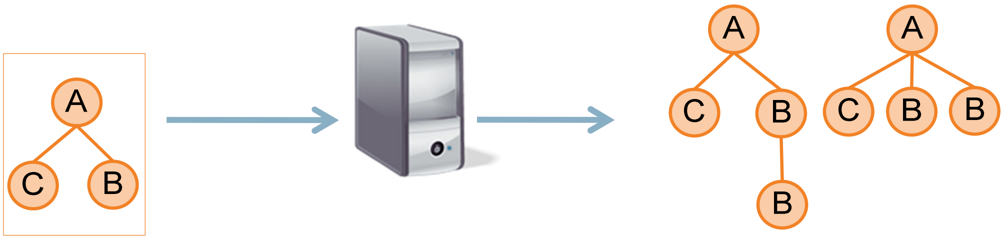
```
algorithm $POTMiner$

Obtain frequent nodes (frequent patterns of size 1)
Build candidate classes $C_1$ from the frequent nodes
for k=2 to MaxSize
    for each class $P \in C_{k-1}$
        for each element $p \in P$.
            Compute the frequency of $p$
            if $p$ is frequent
            then
                Create a new class $P'$ from $p$.
                Add $P'$ to $C_k$
```

POTMiner

Implementación paralela

Distribución de candidatos [CD: Candidate Distribution]





POTMiner

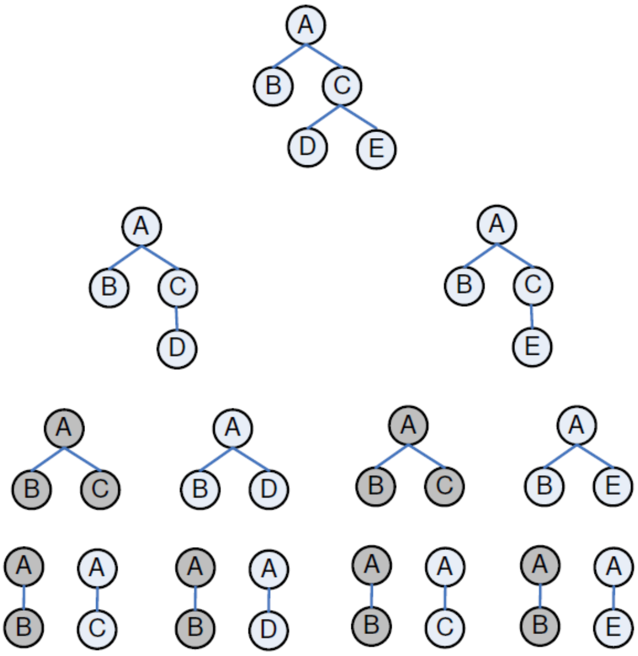
Implementación paralela

```
algorithm $ParallelPOTMiner$
    Obtain frequent nodes (frequent patterns of size 1)
    Build candidate classes $C_1$ from the frequent nodes
    for k=2 to MaxSize
        for each class $P \in C_{k-1}$
            Extend $P$ in parallel to obtain $P_{extended}$
        for each class $P \in C_{k-1}$
            $C_k = C_k \bigcup P_{extended}$
```

POTMiner

POTMiner "light"

(generación de listas de ámbitos bajo demanda)

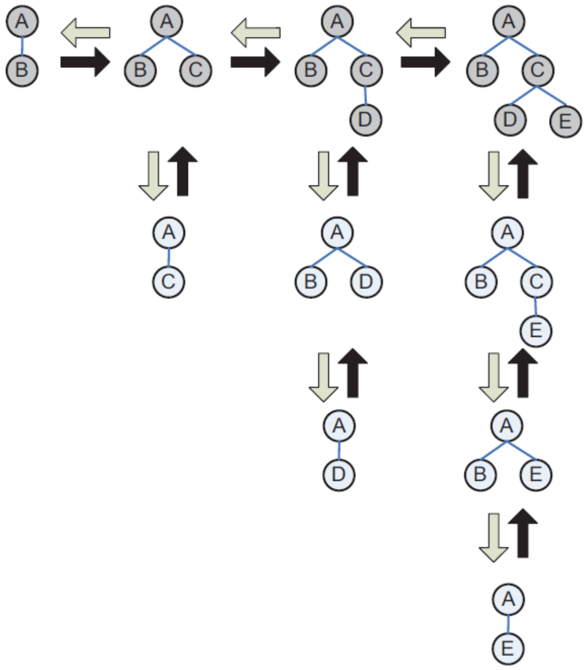


```
algorithm $scopeList$ (Tree $t$): $s$
    // $t : n_1, n_2..n_{k-1}, n_k$

    if $k = 1$
    then
        $s$ = scope list of node $n_k$
    else
        $t_1 = t - n_k$
        $t_2 = t - n_{k-1}$
        $s_1 = scopeList(t_1)$
        $s_2 = scopeList(t_2)$
        if $n_k$.parent $= n_{k-1}$
        then
            // $t$ was obtained by child extension
            s = $in\text{-}scope\text{-}join$ $(s_1, s_2)$
        else
            // $t$ was obtained by cousin extension
            s = $out\text{-}scope\text{-}join$ $(s_1, s_2)$
```

POTMiner

POTMiner DP ["dynamic programming"]

Tiempo de CPU vs. Uso de memoria



```
algorithm $scopeListDP$ (Tree $t$): $s$
// $t : n_1, n_2..n_{k-1}, n_k$

for i=1 to k
    $list[i]$ = scope list of node $n_i$
    for j=1 to i-1
        if j=1
        then // pattern of size 2
            list[i] = $in\text{-}scope\text{-}join$ $(list[j], list[i])$
        else
            $s$ = subtree[j+1][i] // $s : s_1, s_2..s_j, s_{j+1}$
            if $s_{j+1}$.parent $= s_j$
            then
                list[i] = $in\text{-}scope\text{-}join(list[j], list[i])$
            else
                list[i] = $out\text{-}scope\text{-}join$ $(list[j], list[i])$
return list[k]
```

POTMiner

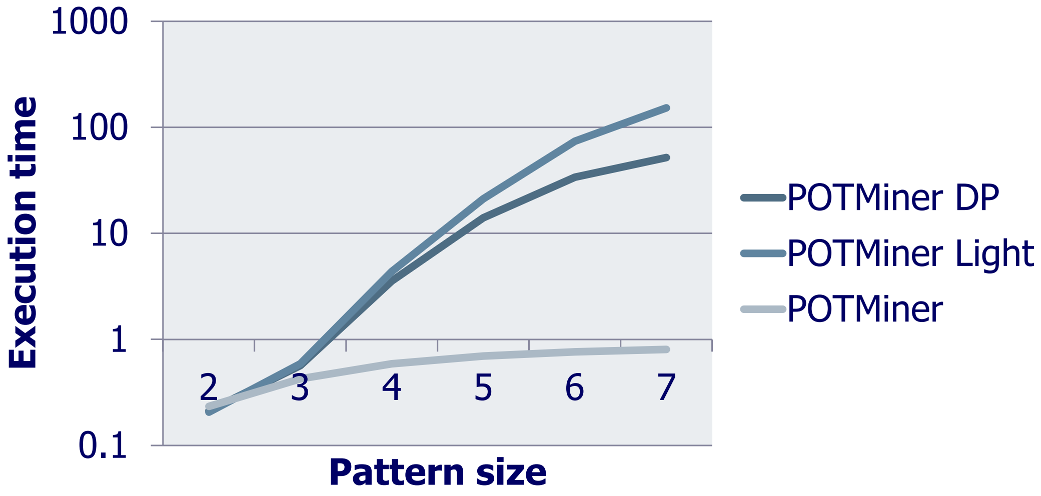
Eficiencia

Algorithm	CPU Time	Memory consumption (number of scope lists)
POTMiner	$O\left(\frac{t*(Ln^2)^{MaxSize}}{(MaxSize-1)!}\right)$	$O(L^{MaxSize-1}*(MaxSize-2)!)$
POTMiner Light	$O\left(\frac{t*(2Ln^2)^{MaxSize}}{(MaxSize-1)!}\right)$	O(L)
POTMiner DP	$O\left(\frac{t*MaxSize^2*(Ln^2)^{Maxsize}}{(Maxsize-1)!}\right)$	O$(L+k-1)$

POTMiner

Resultados experimentales

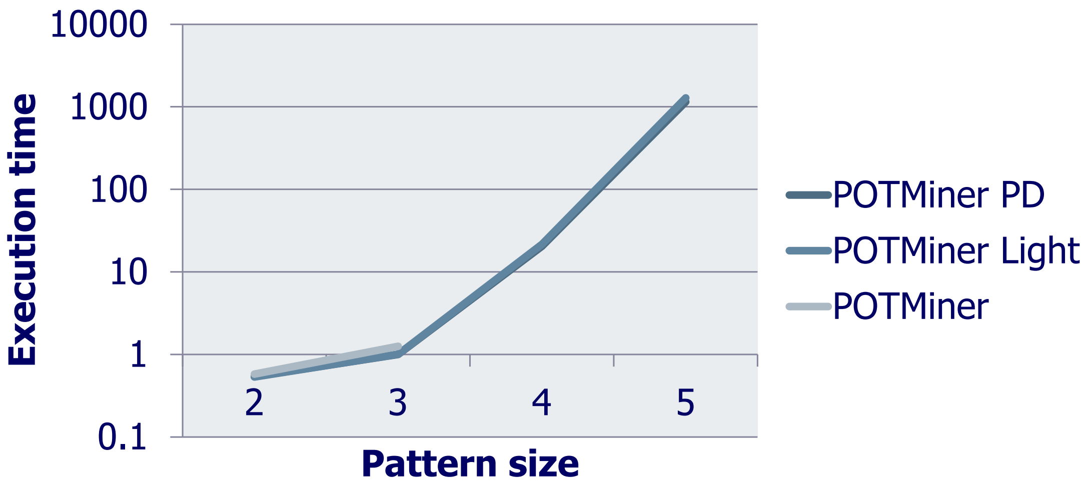
Datos sintéticos



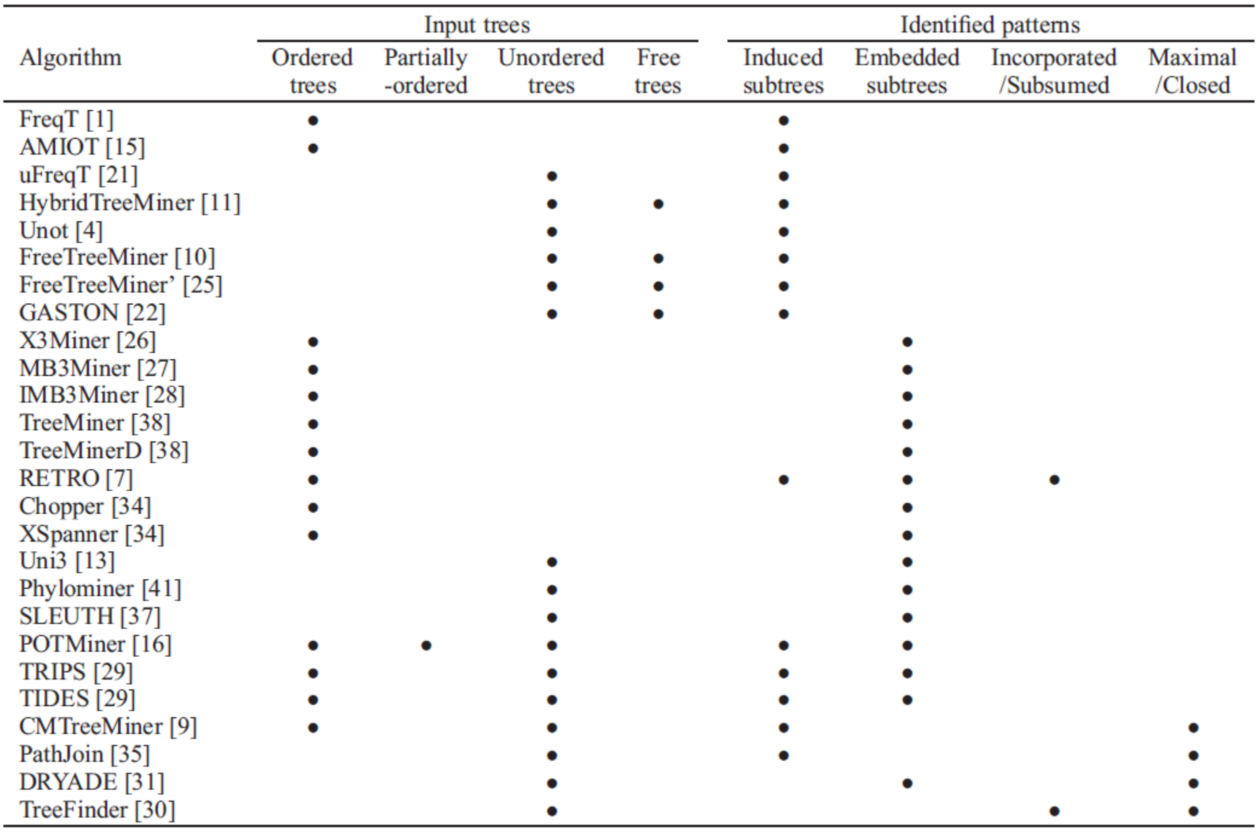
POTMiner

Resultados experimentales

Datos reales

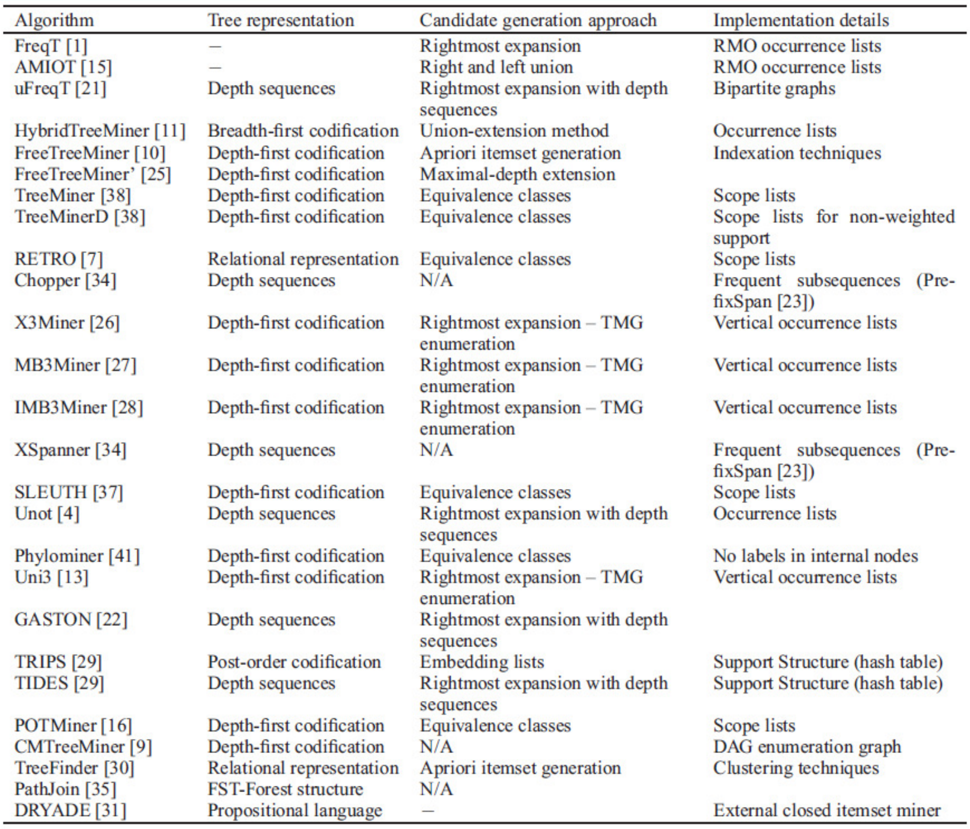


Algoritmos



Algorithm	Input trees				Identified patterns			
	Ordered trees	Partially -ordered	Unordered trees	Free trees	Induced subtrees	Embedded subtrees	Incorporated /Subsumed	Maximal /Closed
FreqT [1]	•				•			
AMIOT [15]	•				•			
uFreqT [21]			•		•			
HybridTreeMiner [11]			•	•	•			
Unot [4]			•		•			
FreeTreeMiner [10]			•	•	•			
FreeTreeMiner' [25]			•	•	•			
GASTON [22]			•	•	•			
X3Miner [26]	•					•		
MB3Miner [27]	•					•		
IMB3Miner [28]	•					•		
TreeMiner [38]	•					•		
TreeMinerD [38]	•					•		
RETRO [7]	•				•	•	•	
Chopper [34]	•					•		
XSpanner [34]	•					•		
Uni3 [13]			•			•		
Phylominer [41]			•			•		
SLEUTH [37]			•			•		
POTMiner [16]	•	•	•		•	•		
TRIPS [29]	•		•		•	•		
TIDES [29]	•		•		•	•		
CMTreeMiner [9]	•		•		•			•
PathJoin [35]			•		•			•
DRYADE [31]			•			•		•
TreeFinder [30]			•				•	•

Algoritmos

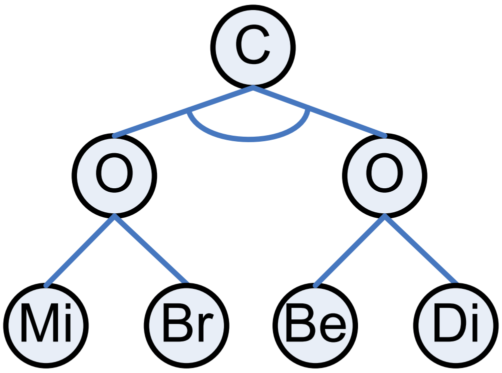


Algorithm	Tree representation	Candidate generation approach	Implementation details
FreqT [1]	–	Rightmost expansion	RMO occurrence lists
AMIOT [15]	–	Right and left union	RMO occurrence lists
uFreqT [21]	Depth sequences	Rightmost expansion with depth sequences	Bipartite graphs
HybridTreeMiner [11]	Breadth-first codification	Union-extension method	Occurrence lists
FreeTreeMiner [10]	Depth-first codification	Apriori itemset generation	Indexation techniques
FreeTreeMiner' [25]	Depth-first codification	Maximal-depth extension	
TreeMiner [38]	Depth-first codification	Equivalence classes	Scope lists
TreeMinerD [38]	Depth-first codification	Equivalence classes	Scope lists for non-weighted support
RETRO [7]	Relational representation	Equivalence classes	Scope lists
Chopper [34]	Depth sequences	N/A	Frequent subsequences (PrefixSpan [23])
X3Miner [26]	Depth-first codification	Rightmost expansion – TMG enumeration	Vertical occurrence lists
MB3Miner [27]	Depth-first codification	Rightmost expansion – TMG enumeration	Vertical occurrence lists
IMB3Miner [28]	Depth-first codification	Rightmost expansion – TMG enumeration	Vertical occurrence lists
XSpanner [34]	Depth sequences	N/A	Frequent subsequences (PrefixSpan [23])
SLEUTH [37]	Depth-first codification	Equivalence classes	Scope lists
Unot [4]	Depth sequences	Rightmost expansion with depth sequences	Occurrence lists
Phylominer [41]	Depth-first codification	Equivalence classes	No labels in internal nodes
Uni3 [13]	Depth-first codification	Rightmost expansion – TMG enumeration	Vertical occurrence lists
GASTON [22]	Depth sequences	Rightmost expansion with depth sequences	
TRIPS [29]	Post-order codification	Embedding lists	Support Structure (hash table)
TIDES [29]	Depth sequences	Rightmost expansion with depth sequences	Support Structure (hash table)
POTMiner [16]	Depth-first codification	Equivalence classes	Scope lists
CMTreeMiner [9]	Depth-first codification	N/A	DAG enumeration graph
TreeFinder [30]	Relational representation	Apriori itemset generation	Clustering techniques
PathJoin [35]	FST-Forest structure	N/A	
DRYADE [31]	Propositional language	–	External closed itemset miner

Aplicaciones

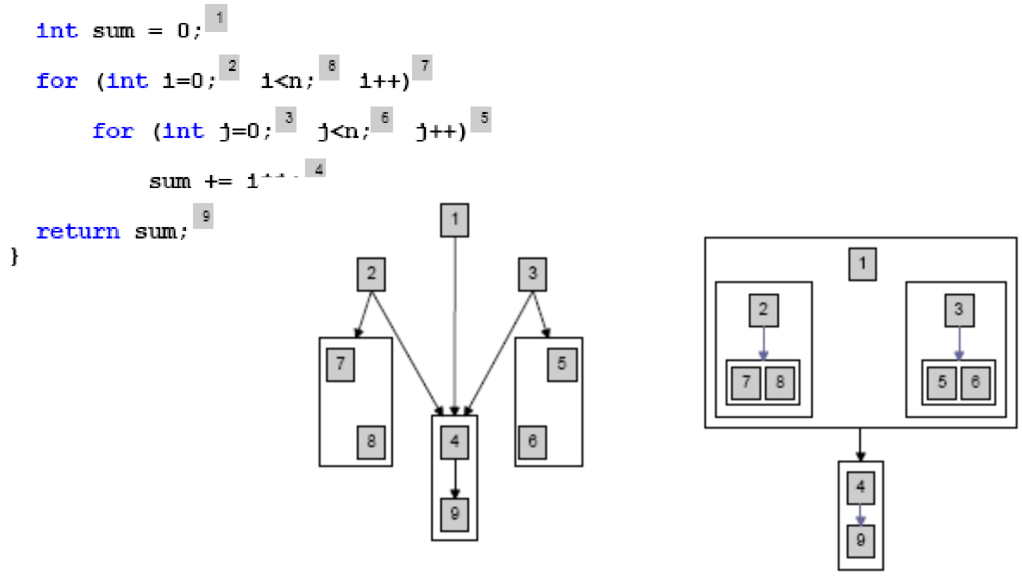
Documentos XML

```
<customer>
   <order>
      <item>milk</item>
      <item>bread</item>
   </order>
   <order>
      <item>beer</item>
      <item>diapers</item>
   </order>
</customer>
```



Aplicaciones

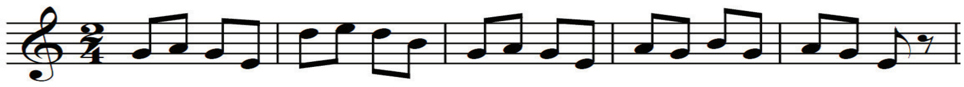
Análisis de software (grafos de dependencias)

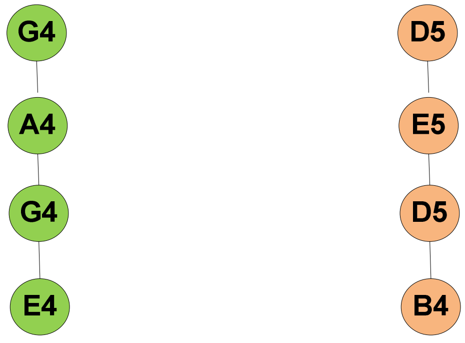


```
public int TestNestedFor (int n) {
  int sum = 0;
  for (int i=0;  i<n;  i++)
      for (int j=0;  j<n;  j++)
          sum += i
  return sum;
}
```

Aplicaciones

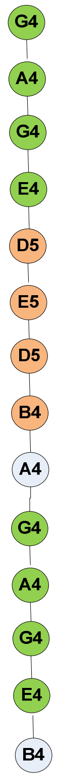
Patrones musicales





Patrón exacto

Patrón similarn



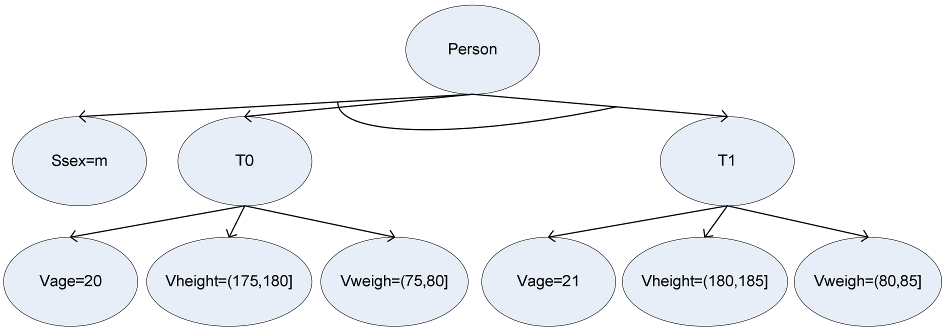
Aplicaciones

Estudios longitudinales

Representación basada en el tiempo

		Person					
id	**Ssex**	**V0age**	**V0height**	**V0weigh**	**V1age**	**V1height**	**V1weigh**
1	m	20	(175,180]cm	(75,80)kg	21	(180,185]cm	(80,85]kg
2	f	16	(155,160]cm	(50,55]kg	17	(160,165]cm	(60,65]kg

		Person		
id	**Ssex**	**Vage**	**Vheight**	**Vweigh**
1	m	20	(175,180]cm	(75,80)kg
1	m	21	(180,185]cm	(80,85]kg
2	f	16	(155,160]cm	(50,55]kg
2	f	17	(160,165]cm	(60,65]kg



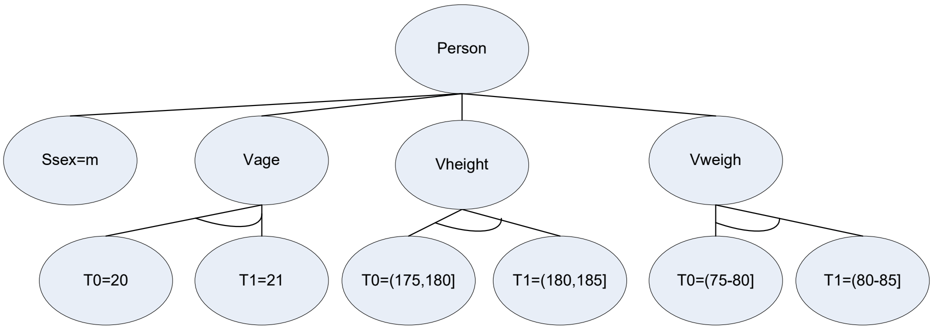
Aplicaciones

Estudios longitudinales

Representación basada en variables

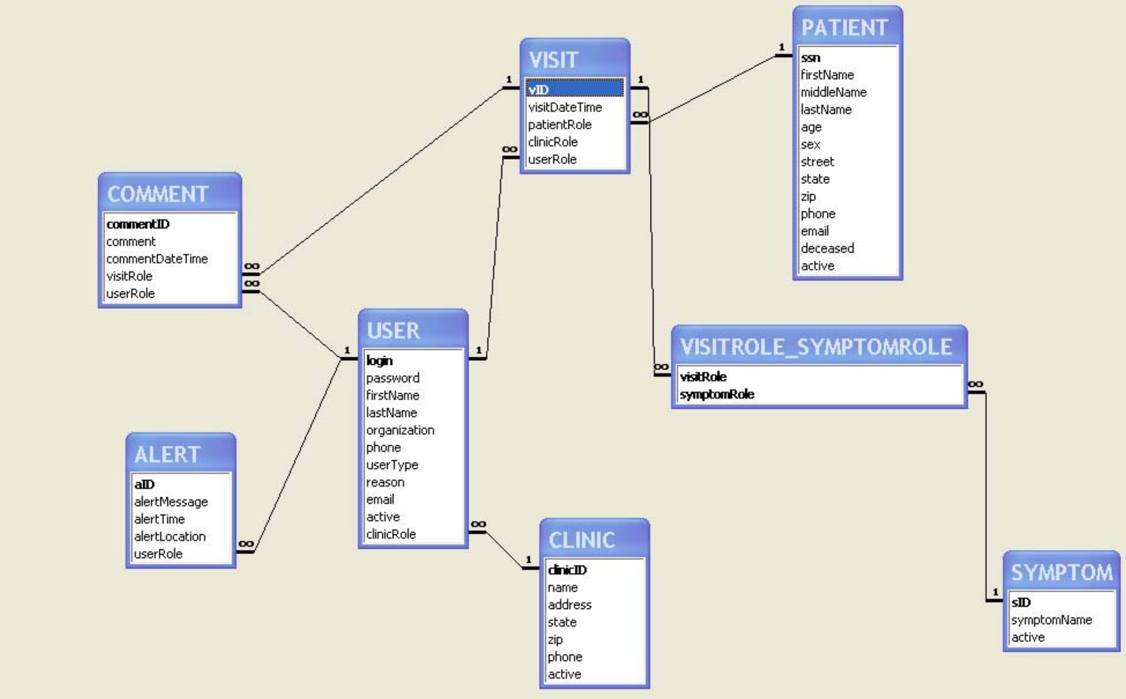
		Person					
id	**Ssex**	**V0age**	**V0height**	**V0weigh**	**V1age**	**V1height**	**V1weigh**
1	m	20	(175,180]cm	(75,80)kg	21	(180,185]cm	(80,85]kg
2	f	16	(155,160]cm	(50,55]kg	17	(160,165]cm	(60,65]kg

		Person		
id	**Ssex**	**Vage**	**Vheight**	**Vweigh**
1	m	20	(175,180]cm	(75,80)kg
1	m	21	(180,185]cm	(80,85]kg
2	f	16	(155,160]cm	(50,55]kg
2	f	17	(160,165]cm	(60,65]kg



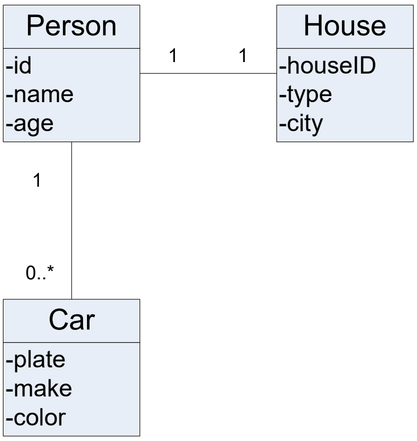
Aplicaciones

Bases de datos multirelacionales



Aplicaciones

Bases de datos multirelacionales



Person

id	name	age	houseID
1	Peter	young	5
…	…	…	

House

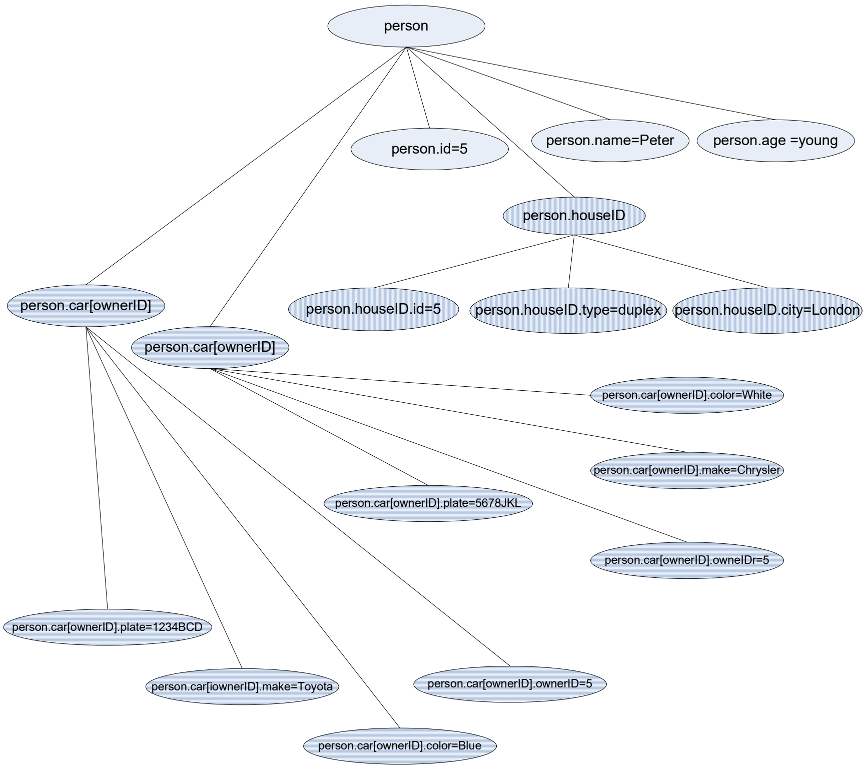
houseID	type	city
5	duplex	London
…	…	…

Car

plate	make	color	ownerID
1234BCD	Toyota	blue	1
5678JKL	Chyrsler	white	1
…	…	…	

Aplicaciones

Bases de datos multirelacionales


person
person.id=5
person.name=Peter
person.age =young
person.houseID
person.car[ownerID]
person.houseID.id=5
person.houseID.type=duplex
person.houseID.city=London
person.car[ownerID]
person.car[ownerID].color=White
person.car[ownerID].make=Chrysler
person.car[ownerID].plate=5678JKL
person.car[ownerID].owneIDr=5
person.car[ownerID].plate=1234BCD
person.car[iownerID].make=Toyota
person.car[ownerID].ownerID=5
person.car[ownerID].color=Blue

Aplicaciones

Bases de datos multirelacionales

Dos tipos de patrones

- E = Embedded subtrees
- I = Induced subtrees

Dos esquemas de representación

- K = "Key-based tree representation"
- O = "Object-based tree representation"

$$IK \subseteq EK \subset_{eq} IO \subseteq EO$$

Agradecimientos

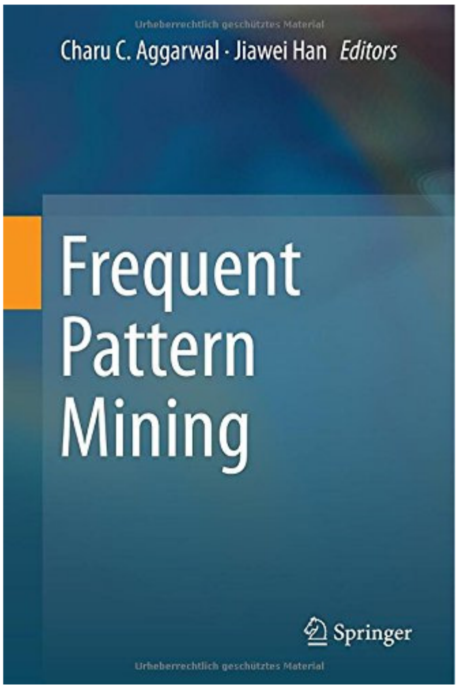
Aída Jiménez, Ph.D.

Tesis doctoral
Knowledge discovery in non-linear structures
Departamento de Ciencias de la Computación e I.A.
Marzo de 2011

"If you torture the data long enough,
it will confess" -- Ronald Coase

Bibliografía

- Charu C. Aggarwal & Jiawei Han(editors):
  Frequent Pattern Mining.
  Springer, 2014.
  ISBN 3319078208.



Urheberrechtlich geschütztes Material

Charu C. Aggarwal · Jiawei Han *Editors*

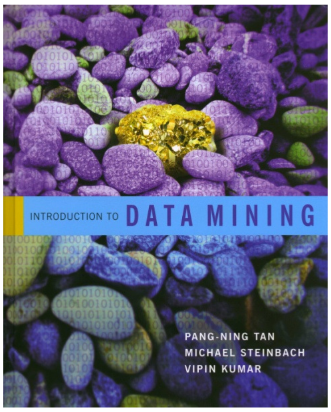
Frequent Pattern Mining

Springer

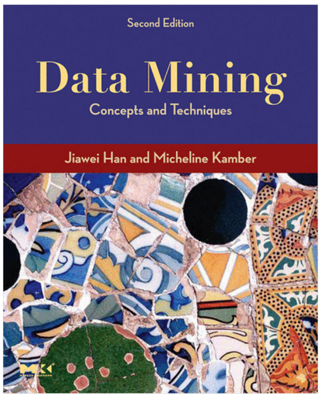
Urheberrechtlich geschütztes Material

Bibliografía

- Pang-Ning Tan,
  Michael Steinbach
  & Vipin Kumar:
  Introduction to Data Mining
  Addison-Wesley, 2006.
  ISBN 0321321367 [capítulos 6&7]



- Jiawei Han
  & Micheline Kamber:
  Data Mining:
  Concepts and Techniques
  Morgan Kaufmann, 2006.
  ISBN 1558609016 [capítulo 5]



Bibliografía – Algoritmos

- Aída Jiménez, Fernando Berzal & Juan Carlos Cubero:
  Frequent tree pattern mining: A survey,
  Intelligent Data Analysis 14(6):603-622, 2010.
  DOI 10.3233/IDA-2010-0443

- Aída Jiménez, Fernando Berzal & Juan Carlos Cubero:
  POTMiner: Mining ordered, unordered, and partially-ordered trees,
  Knowledge and Information System 23(2): 199-224, 2010.
  DOI 110.1007/s10115-009-0213-3

- Aída Jiménez, Fernando Berzal & Juan Carlos Cubero:
  Mining frequent patterns from XML data:
  Efficient algorithms and design trade-offs.
  Expert Systems with Applications 39(1):1134-1140, 2012.
  DOI 10.1016/j.eswa.2011.07.113

- Aída Jiménez, Fernando Berzal & Juan Carlos Cubero:
  Interestingness measures for association rules within groups.
  Intelligent Data Analysis 17(2):195-215, 2013.
  DOI 10.3233/IDA-130574

Bibliografía – Aplicaciones

- Fernando Berzal, Juan Carlos Cubero & Aída Jiménez:
Hierarchical program representation for program element matching.
Proceedings of the 8th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'07), pp. 467-476. DOI 10.1007/978-3-540-77226-2_48

- Aída Jiménez, Miguel Molina-Solana, Fernando Berzal & Waldo Fajardo:
Mining transposed motifs in music.
Journal of Intelligent Information Systems 36(1):99-115, 2011.
DOI 10.1007/s10844-010-0122-7

- Aída Jiménez, Fernando Berzal & Juan-Carlos Cubero:
Mining patterns from longitudinal studies.
Proceedings of the 7th international conference on Advanced Data Mining and Applications (ADMA'11), pp. 166-179. DOI 10.1007/978-3-642-25856-5_13

- Aída Jiménez, Fernando Berzal & Juan Carlos Cubero:
Using trees to mine multirelational databases.
Data Mining and Knowledge Discovery 24(1):1-39, 2012.
DOI 10.1007/s10618-011-0218-x