Resolución Eficiente de Horn-SAT mediante Algoritmos Bottom-Up y Top-Down

Jacson Ferney Cardenas Torres, M.Sc. (En curso), Ingeniero Mecatrónico, Especialización Automatización Industrial, Universidad Santo Tomás, Bucaramanga, Colombia, jacson.cardenas@ustabuca.edu.co

Resumen - La satisfacibilidad Horn proposicional (Horn-SAT) es fundamental en inteligencia artificial, representando un pilar en campos principales: (i) consulta/respuesta de programas lógicos, (ii) propagación unitaria en solucionadores SAT modernos, y (iii) modelado mediante hipergrafos dirigidos (And/Or-grafos). Horn-SAT constituye el problema tratable más prominente caracterizado por fórmulas con cláusulas Horn que contienen como máximo un literal positivo, reduciendo la complejidad de exponencial a lineal. Aunque W.F. Dowling y J.H. Gallier (1984) desarrollaron un algoritmo Bottom-Up Lineal (LBU) pionero, éste presenta la limitación fundamental de inferir casi todas las consecuencias lógicas independientemente de la consulta específica, siendo impracticable para lógicas expresivas donde genera modelos potencialmente infinitos. Los esfuerzos previos de algoritmos top-down lineales han resultado incompletos o técnicamente insuficientes, dejando una brecha crítica en la literatura.

El presente trabajo formula e implementa una versión básica del algoritmo Top-Down (LTD) para Horn-SAT sin contemplar ciclos, resolviendo las deficiencias fundamentales de enfoques anteriores mediante estrategias de marcaje dirigidas por consultas. La contribución principal consiste en ofrecer una implementación correctamente especificada del enfoque top-down básico junto con un análisis comparativo exhaustivo que demuestra cuándo cada estrategia es superior. La evaluación experimental utiliza fórmulas Horn parametrizadas como benchmarks (matriz, incremental y completo) que exponen desafíos inherentes a la clase Horn, demostrando que la eficiencia relativa depende fundamentalmente de las características topológicas: LTD sobresale frente a LBU en estructuras incrementales por su capacidad de limitar el espacio de búsqueda dirigido por consultas, mientras que LBU mantiene ventajas en configuraciones donde las constantes de recursión afectan el rendimiento del LTD.

Palabras claves: Horn-SAT, Satisfacibilidad, complejidad, algoritmo Bottom-Up, algoritmo Top-Down, benchmarks

Abstract - Abstract - Propositional Horn satisfiability (Horn-SAT) is fundamental in artificial intelligence, representing a pillar in main fields: (i) query/answering propositional logic programs, (ii) unit propagation in modern SAT solvers, and (iii) modeling through directed hypergraphs (And/Or-graphs). Horn-SAT constitutes the most prominent tractable problem characterized by formulas with Horn clauses containing at most one positive literal, reducing complexity from exponential to linear. Although W.F. Dowling and J.H. Gallier (1984) developed a pioneering Linear Bottom-Up (LBU) algorithm, it presents the fundamental limitation of inferring almost all logical consequences regardless of the specific query, being impractical for expressive logics where it generates potentially infinite models. Previous efforts on linear top-down algorithms have resulted incomplete or technically insufficient, leaving a critical gap in the literature.

This work formulates and implements a basic version of the Top-Down (LTD) algorithm for Horn-SAT without considering cycles, solving fundamental deficiencies of previous approaches through query-driven marking strategies. The main contribution consists in offering a correctly specified implementation of the basic top-down approach along with an exhaustive comparative analysis that demonstrates when each strategy is superior. Experimental evaluation uses parametrized Horn formulas as benchmarks (complete, incremental, and matrix) that expose challenges inherent to the Horn class, demonstrating that relative efficiency depends fundamentally on topological characteristics: LTD outperforms LBU in incremental structures by effectively restricting the query-driven search space, while LBU retains advantages in configurations where recursion overhead impacts LTD's practical performance.

Keywords: Horn-SAT, Satisfiability, complexity, Bottom-Up algorithm, Top-Down algorithm, benchmarks

I. INTRODUCCIÓN

El problema de Satisfacibilidad (SAT) constituye uno de los pilares fundamentales en la teoría de la computación y la inteligencia artificial, utilizado para modelar y resolver problemas que se pueden expresar en términos de variables booleanas. Estas variables se organizan en fórmulas lógicas representadas en forma clausal, donde una cláusula es una disyunción de literales, y una fórmula en forma clausal es una conjunción de estas cláusulas. El problema SAT general busca determinar si existe una asignación de verdad para las variables que satisfaga todas las cláusulas, siendo este un problema NP-completo con relevancia en áreas como verificación de sistemas, optimización y razonamiento automatizado.

Dentro de este contexto, Horn-SAT emerge como un subproblema eficiente de SAT donde las fórmulas están restringidas a cláusulas Horn que contienen como máximo un

literal positivo, siguiendo patrones ($\neg p_1 \vee ... \vee \neg p_k$) o ($p \vee \neg p_1 \vee ... \vee \neg p_k$). Esta restricción estructural permite resolver Horn-SAT en tiempo lineal, contrastando significativamente con la complejidad NP-completa del SAT general, y habilita aplicaciones eficientes en bases de datos deductivas, programación lógica declarativa y razonamiento automatizado.

Para la resolución de Horn-SAT, dos estrategias de búsqueda son convencionales: bottom-up que comienza con los hechos e intenta deducir la consulta con inferencias tipo modus-ponens, y top-down que va desde la consulta hacia los hechos con inferencias tipo modus-tollens. El algoritmo Bottom-Up Lineal (LBU), desarrollado por W.F. Dowling y J.H. Gallier en 1984 [1], construye sistemáticamente el modelo mínimo completo utilizando contadores para detectar cláusulas unitarias. Sin embargo, presenta la limitación fundamental de deducir casi todas las consecuencias lógicas independientemente de la consulta específica, siendo inviable para lógicas expresivas donde puede generar modelos infinitos.

Por el contrario, la búsqueda Top-Down constituye la base de la programación lógica moderna (SLD-resolución) y sus dialectos, adoptando una estrategia dirigida por consultas específicas que explora solo las asignaciones de verdad relevantes. No obstante, los intentos previos de desarrollar algoritmos top-down lineales han resultado incompletos o técnicamente insuficientes, dejando una brecha crítica en la literatura.

La relevancia de comparar ambos enfoques surge de su incompatibilidad estratégica fundamental que los hace óptimos para dominios de aplicación mutuamente excluyentes. Bottom-Up es ideal para compiladores que requieren análisis exhaustivo de flujo de datos, bases de datos deductivas que necesitan materialización completa de vistas, y sistemas de verificación exhaustiva. Conversamente, Top-Down es esencial para programación lógica de primer orden donde Bottom-Up generaría modelos infinitos, verificación formal dirigida que valida propiedades específicas, y consultas eficientes en bases de conocimiento industriales con millones de reglas. Esta incompatibilidad crea escenarios donde la selección incorrecta puede resultar en ineficiencia exponencial o inviabilidad computacional.

El presente trabajo aborda esta brecha mediante el desarrollo e implementación de una versión básica del algoritmo Top-Down (LTD) para Horn-SAT sin contemplar ciclos, proporcionando una implementación correctamente especificada junto con un análisis comparativo exhaustivo. La evaluación experimental utiliza tres benchmarks parametrizados: matriz (patrones estructurados en m filas y n columnas), incremental (estructura jerárquica acumulativa), y completo (grafo dirigido completo). Estas configuraciones permiten determinar bajo qué condiciones estructurales cada algoritmo es superior, estableciendo criterios fundamentados para la selección de estrategias según las características topológicas del dominio.

II. DESCRIPCIÓN DEL PROBLEMA

El problema Horn-SAT puede representarse como un conjunto de hechos, reglas y una consulta. Los “Hechos” son afirmaciones consideradas verdaderas en el sistema y sirven como base para inferir nuevos conocimientos. Las “Reglas” son expresiones lógicas que relacionan diferentes literales y permiten derivar nuevos hechos a partir de los existentes. Una “Consulta” busca verificar si, dada la base de hechos y las reglas, es posible deducir un literal específico planteado. En esencia, el problema consiste en determinar si, a partir de los hechos iniciales y las reglas, es posible deducir la consulta planteada sin redundancias.

Por ejemplo, las reglas:

$$\text{Regla 1: } P_1 \leftarrow P_2 \wedge P_3 \quad (1)$$
$$(\text{Si } P_2 \text{ y } P_3 \text{ son verdaderos, se deduce } P_1)$$

$$\text{Regla 2: } P_5 \leftarrow P_1 \wedge P_4 \quad (2)$$
$$(\text{Si } P_1 \text{ y } P_4 \text{ son verdaderos, se deduce } P_5)$$

Que ilustran la derivación de nuevos hechos a partir de hechos previos y reglas lógicas.

Una fórmula en forma normal conjuntiva CNF es un conjunto de cláusulas donde cada cláusula es un conjunto de literales. Una CNF es Horn si todas las cláusulas tienen como máximo un literal positivo. Por ejemplo:

$$\neg P_2 \vee \neg P_3 \vee P_1 \quad y \quad \neg P_1 \vee \neg P_4 \vee P_5 \quad (3)$$

Las cláusulas de Horn también pueden incluir literales solo negativos o positivos, como:

$$\neg P_1 \vee \neg P_2 \quad y \quad P_3 \vee P_4 \quad (4)$$

Este enfoque permite modelar lógicamente el problema y representar consultas como literales objetivos en las fórmulas Horn.

Un literal o consulta se considera consecuencia lógica de una base de hechos y reglas si resulta verdadero en todos los casos en que dichos hechos y reglas también lo son. Asimismo, si la fórmula Horn compuesta por los hechos, las reglas y la negación de la consulta es insatisfacible, se concluye que la consulta es una consecuencia lógica de esa base. De forma equivalente, si la consulta es una consecuencia lógica, su negación necesariamente conduce a una contradicción.

El problema Horn-SAT también puede representarse mediante grafos AND/OR; una estructura en forma de árbol que combina nodos y conectores para modelar las relaciones lógicas. Cada regla se representa por un conector AND que conecta el nodo consecuente de la regla con todos los nodos antecedentes. Un conector AND es verdadero si todos los nodos antecedentes son verdaderos. Cada proposición por su parte se representa como un nodo OR el cual es verdadero si al menos un conector AND cuyo consecuente es el nodo verdadero. Determinar si una pregunta Q es consecuencia de los hechos y reglas equivale a verificar si el nodo Q en el grafo AND/OR asociado es evaluado como verdadero.

Finalmente, la solución del problema Horn-SAT se traduce en construir un árbol AND/OR siendo un subgrafo acíclico que incluye la consulta como raíz y las hojas como hechos iniciales.

Este enfoque asegura que todas las deducciones necesarias para alcanzar la consulta estén respaldadas por los hechos iniciales y evita la formación de ciclos en el proceso de deducción.

III. ALGORITMO BOTTOM-UP (LBU)

La fase de inicialización del algoritmo Bottom-Up (LBU) está diseñada para preparar las estructuras necesarias en la fase deductiva (DBU). Esta etapa organiza los datos de entrada y asigna valores iniciales a las variables involucradas.

Algorithm 1 LBU - Initialization Phase of LBU

Input: F_Π, R_Π, Q, B_Π
Output: YES, NO
1: **for** $p = 1$ to $|B_\Pi|$ **do**
2: $firstOutput[p] \leftarrow$ **T**
3: $rulesInBody[p] \leftarrow \emptyset$
4: **end for**
5: $currentRule \leftarrow R_\Pi.first$
6: **for** $r = 1$ to $|R_\Pi|$ **do**
7: $head[r] \leftarrow currentRule.h$
8: $counter[r] \leftarrow |currentRule.B|$
9: **for** each $p \in B$ **do**
10: $rulesInBody[p] \leftarrow rulesInBody[p] \cup \{r\}$
11: **end for**
12: $currentRule \leftarrow currentRule.next$
13: **end for**
14: **return** DBU(F_Π)

La inicialización asociada a cada regla denominada *counter[r]* es una estructura crucial que indica cuántos literales del antecedente de la regla aún no han sido satisfechos. Inicialmente, cada regla recibe un contador igual al número de literales en su antecedente $|currentRule.B|$. Esto permite que las reglas sean evaluadas de manera eficiente durante la fase deductiva ya que cada decremento en el contador acerca la regla a ser activada.

La fase deductiva del algoritmo Bottom-Up (DBU) es una de las etapas más importantes del proceso ya que permite deducir nuevos literales y verificar si la consulta Q puede ser derivada. En esta etapa, los contadores asociados a cada regla desempeñan un papel fundamental en la optimización del rendimiento del algoritmo.

La implementación de los contadores permite reducir drásticamente la complejidad del algoritmo, pasando de un enfoque cuadrático a uno lineal. Este cambio se logra por qué; en lugar de evaluar constantemente todas las reglas para verificar si pueden ser activadas, los contadores actúan como indicadores directos que señalan cuándo una regla está lista para deducir su literal consecuente.

El funcionamiento se basa en reducir el valor del contador asociado a una regla cada vez que uno de sus literales en el antecedente es satisfecho. Cuando el contador alcanza el valor cero, significa que todos los literales en el antecedente de la regla han sido satisfechos y por lo tanto la regla puede activarse para deducir su único literal consecuente. Este enfoque incremental elimina la necesidad de iterar innecesariamente sobre reglas irrelevantes, lo que mejora significativamente la complejidad del algoritmo.

Algorithm 2 DBU - Deductive Phase of DBU

Input: F_Π
Output: YES, NO
1: $deducedPropositions \leftarrow F_\Pi$
2: **while** $deducedPropositions \neq \emptyset$ **do**
3: ExtractFirstProposition(*deducedPropositions*)
4: **if** $firstOutput[p] =$ **T then**
5: $firstOutput[p] \leftarrow$ **F**
6: **for** each $r \in rulesInBody[p]$ **do**
7: $counter[r] \leftarrow counter[r] - 1$
8: **if** $counter[r] = 0$ **then**
9: **if** $head[r] = Q$ **then**
10: **return** YES
11: **end if**
12: $deducedPropositions \leftarrow deducedPropositions \cup \{head[r]\}$
13: **end if**
14: **end for**
15: **end if**
16: **end while**
17: **return** NO

IV. ALGORITMO TOP-DOWN (LTD)

El algoritmo Top-Down (LTD) se enfoca en resolver el problema Horn-SAT iniciando desde la consulta Q e intentando deducir si esta puede ser derivada a partir de los hechos F_Π y las reglas R_Π. Este enfoque utiliza un mecanismo de marcaje de nodos y se organiza en tres funciones principales: (1) Inicialización, (2) OR y (3) AND. A continuación, se describe cada una de estas partes:

La fase de Inicialización en el algoritmo Top-Down (LTD) organiza las estructuras de datos necesarias para gestionar el marcaje de nodos durante la deducción. Cada literal p en la base de literales B_Π se marca inicialmente como UNEXPANDED, indicando que aún no ha sido explorado. Además, se asocia a cada literal una lista de reglas $rulesWithHead[p]$ que lo tienen como consecuente, facilitando la relación entre literales y sus reglas asociadas. Este proceso prepara el algoritmo para realizar búsquedas dirigidas desde la consulta Q.

Algorithm 3 LTD - Initializing data-structures

Input: F_Π, R_Π, Q, B_Π
Output: YES, NO
1: **for** $p = 1$ to $|B_\Pi|$ **do**
2: $propositionState[p] \leftarrow$ UNEXPANDED
3: $rulesWithHead[p] \leftarrow \emptyset$
4: **end for**
5: **for** each $p \in F_\Pi$ **do**
6: $propositionState[p] \leftarrow$ **T**

```
7: end for
8: currentRule ← R_Π.first
9: for r = 1 to |R_Π| do
10:     body[r] ← currentRule.B
11:     rulesWithHead[h] ← rulesWithHead[h] ∪ {r}
12:     currentRule ← currentRule.next
13: end for
14: if propositionState[Q] = UNEXPANDED then
15:     OR(Q)
16: end if
17: if propositionState[Q] = T then
18:     return YES
19: else
20:     return NO
21: end if
```

La función OR toma un literal p como entrada y lo marca como EXPANDING mientras evalúa si puede deducirse. Este proceso, consiste en verificar todas las reglas asociadas a p en la lista *rulesWithHead*[p]. Para cada regla se evalúa su antecedente mediante la función AND si alguna regla es satisfecha p se marca como T, y el algoritmo detiene la búsqueda para p. Si no existen reglas asociadas o ninguna es válida p se marca como F. Este paso garantiza que los nodos correspondientes a p, se evalúan completamente antes de avanzar.

Algorithm 4 OR

```
Input: p
Output: No value
1: state[p] ← EXPANDING
2: for each r ∈ rulesWithHead[p] do
3:     state[p] ← AND(r)
4:     if state[p] = T then
5:         return
6:     end if
7: end for
8: if rulesWithHead[p] = ∅ then
9:     state[p] ← F
10: end if
11: return
```

La función AND se encarga de evaluar el antecedente de una regla compuesto por un conjunto de literales. Para cada literal en el antecedente *body*[r], si aún no ha sido explorado como UNEXPANDED se llama recursivamente a la función OR**.** Si algún literal del cuerpo es falso F, la regla completa falla devolviendo F**.** Si todos los literales son verdaderos T, entonces la regla se considera satisfecha devolviendo T. Este paso asegura que se marcan como visitados todos los nodos necesarios para validar la regla.

Algorithm 5 AND

```
Input: r
Output: T, F
1: for each p ∈ body[r] do
2:     if state[p] = UNEXPANDED then
3:         OR(p)
4:     end if
5:     if state[p] = F then
6:         return F
7:     end if
8: end for
9: return T
```

V. BENCHMARKS

Los benchmarks son herramientas fundamentales para evaluar el rendimiento de algoritmos ya que proporcionan un marco estandarizado y reproducible para medir su eficiencia en diferentes escenarios. En el caso del problema Horn- SAT, los dos algoritmos considerados tienen una complejidad lineal, pero este dato puede ser insuficiente para comprender en profundidad sus eficiencias relativas. En este sentido, los benchmarks ayudan a obtener un mejor conocimiento de los algoritmos y permiten comparar objetivamente sus desempeños bajo diferentes configuraciones.

V-A. Benchmark Completo

El benchmark completo consta de n literales donde; n es el parámetro que define el tamaño del benchmark. Las reglas del benchmark forman un grafo completo de n nodos, es decir; cada par de literales está conectado por una regla. Esto genera un total de $n(n-1)$ reglas con un tamaño total de entrada de $2n(n-1)$.

Las reglas tienen la forma:

$$\{P_i \leftarrow P_j \mid i, j \in \{1, 2, \ldots, n\},\ i \neq j\} \tag{5}$$

donde i y j son índices diferentes entre 1 y n.

V-B. Benchmark Incremental

La fórmula compacta del benchmark incremental se construye acumulando literales en el cuerpo de cada regla de manera progresiva. La última regla P_n incluye todas las literales desde P_{n-1} hasta P_1. La penúltima regla P_{n-1} incluye todas las literales desde P_{n-2} hasta P_1 y así sucesivamente hasta llegar a la primera regla que conecta $P_2 \leftarrow P_1$. Este proceso refleja una estructura jerárquica acumulativa, donde cada regla se construye sobre las anteriores. El tamaño total del benchmark es de $\frac{n}{2}(n+1)$, y se generan n reglas en total.

$$P_n \leftarrow P_{n-1}, P_{n-2}, \ldots, P_1 \tag{6}$$

$$P_{n-1} \leftarrow P_{n-2}, \ldots, P_1 \tag{7}$$

$$P_2 \leftarrow P_1 \tag{8}$$

V-C. Benchmark Matriz

El Benchmark Matriz organiza las reglas en m filas cada una con n proposiciones conectadas de manera jerárquica de derecha a izquierda. Las proposiciones dentro de cada fila son independientes y no se repiten en otras filas, lo que permite que cada fila pueda evaluarse de forma aislada. Si se representa mediante su grafo AND/OR las reglas quedan dispuestas en forma de matriz. El tamaño total del benchmark es de $2mn$, y

el número total de reglas generadas es *mn*.

$$\text{Fila } m : P_{m,n} \leftarrow P_{m,n-1} \leftarrow \cdots \leftarrow P_{m,1} \quad (9)$$

$$\text{Fila } m-1 : P_{m-1,n} \leftarrow P_{m-1,n-1} \leftarrow \cdots \leftarrow P_{m-1,1} \quad (10)$$

$$\text{Fila } 1 : P_{1,n} \leftarrow P_{1,n-1} \leftarrow \cdots \leftarrow P_{1,1} \quad (11)$$

V-D. Tabla Comparativa de Benchmarks

La tabla a continuación presenta un análisis comparativo de los parámetros y la complejidad de los diferentes benchmarks, especificando sus características principales:

Tabla 1. *Parámetros y complejidad de los benchmarks*

Parámetros	Completo (n)	Incremental (n)	Matriz (m, n)
Entrada	2n(n-1)	$\frac{n}{2}(n+1)$	2mn
Reglas	n(n-1)	n	mn
Complejidad	$O(n^2)$	$O(n^2)$	$O(n^2)$

Cada fila de la tabla representa un parámetro específico de los benchmarks. Las columnas indican las configuraciones de los benchmarks (Completo, Incremental y Matriz), mientras que cada casilla detalla las dimensiones o características relevantes como: el número de proposiciones, reglas y el tamaño total de entrada. Esto permite una comparación clara y estructurada de sus propiedades.

VI. COMPLEJIDAD DE LOS ALGORITMOS

Las complejidades de los algoritmos Bottom-Up y Top-Down varían según el tipo de benchmark debido a las características estructurales de las reglas y proposiciones. En el caso del benchmark Completo, las reglas forman un grafo completo con *n(n-1)* reglas entre los nodos. En este caso solo el algoritmo Bottom-Up es aplicable debido a la presencia de ciclos en la estructura. Este algoritmo tiene una complejidad cuadrática $O(n^2)$ ya que debe recorrer y evaluar todas las reglas que conectan los literales en el grafo.

En el benchmark Incremental donde cada regla acumula los literales de las reglas anteriores; ambos algoritmos tienen una complejidad cuadrática $O(n^2)$ en el peor caso. Esto se debe a que; para deducir la última proposición cada algoritmo debe recorrer y procesar las dependencias jerárquicas entre las reglas acumulando los literales necesarios hasta alcanzar la consulta.

Finalmente, en el benchmark Matriz, las reglas se organizan en una estructura bidimensional de m filas y n columnas. El algoritmo Top-Down tiene una complejidad lineal O(m) ya que evalúa las filas independientemente. Por otra parte, el algoritmo Bottom- Up tiene una complejidad $O(n^2)$ ya que recorre todas las filas y columnas para procesar las reglas de manera acumulativa.

Tabla 2. *Complejidades de los algoritmos para los diferentes benchmarks*

Benchmark	Algoritmo	Complejidad
Completo	Bottom-Up	$O(n^2)$
Incremental	Top-Down	$O(n^2)$
	Bottom-Up	$O(n^2)$
Matriz	Top-Down	$O(m)$
	Bottom-Up	$O(n^2)$

En la tabla anterior cada fila representa un tipo de benchmark (Completo, Incremental o Matriz). Las columnas describen el algoritmo evaluado (Top-Down o Bottom-Up) y su complejidad asociada. Cada casilla de la última columna muestra la complejidad del algoritmo correspondiente bajo el benchmark especificado.

VII. RESULTADOS EXPERIMENTALES

En esta sección se presentan los resultados experimentales obtenidos a partir de los benchmarks diseñados para analizar el desempeño de los algoritmos Bottom-Up y Top-Down.

Las fórmulas booleanas utilizadas se expresaron en el formato estándar DIMACS, ampliamente adoptado en la comunidad científica para la representación de problemas SAT en forma normal conjuntiva (CNF). Este formato consta de una cabecera que indica el número de variables y cláusulas seguidas de las cláusulas expresadas como listas de literales, donde cada literal es un número entero positivo para el consecuente y negativos para los antecedentes, finalizando cada cláusula con un ”0” como el identificador final de linea. Por ejemplo: un benchmark incremental con 3 proposiciones; 2 reglas, un hecho y una pregunta *Q* puede representarse de la siguiente manera:

```
p cnf 3 2
2 -1 0
3 -2 -1 0
1 0
-3 0
```

En este ejemplo se define en la primera fila que hay 3 proposiciones; P_1, P_2 y P_3, 2 reglas en su forma CNF; $P_2 \leftarrow P_1$ y $P_3 \leftarrow P_2 \wedge P_1$, un hecho inicial; P_1 y una pregunta Q que verifica si P_3 es deducible. Para obtener los resultados se evaluaron los dos algoritmos para un tamaño máximo de *n* igual 3000 por cada configuración del benchmark y se calculó el tiempo promedio de ejecución para garantizar una evaluación precisa y confiable.

VII-A. Desempeño del algoritmo *Bottom-Up con el benchmark completo*

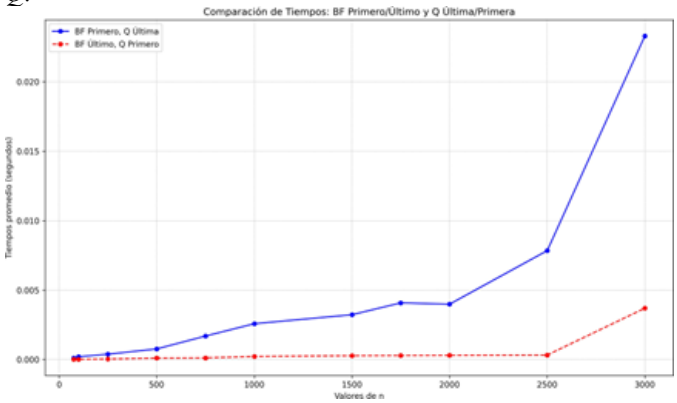
El benchmark completo no puede evaluarse con el algoritmo básico de Top-Down ya que este tipo de benchmark contempla ciclos debido a la estructura de grafo completo y por lo tanto este algoritmo básico no está diseñado para manejar ciclos y al hacerlo generaría un error.

En el caso del algoritmo Bottom-Up se evaluaron dos configuraciones distintas para medir su desempeño: En la primera configuración la pregunta Q se sitúa al inicio y el hecho BF al final. En este caso el algoritmo opera de manera casi constante para cualquier valor de *n*. Esto se debe a que el algoritmo selecciona primero el hecho final y en la primera

regla disponible tiene a Q como consecuente, por lo tanto, la respuesta se obtiene inmediatamente independientemente del tamaño de n.

En la segunda configuración se intercambia la pregunta Q para el final y el hecho BF se pasa al principio. En este caso el algoritmo debe deducir todas las reglas de forma ordenada aplicando una búsqueda en profundidad. Como la pregunta se encuentra en la última posición se requiere deducir secuencialmente todas las reglas hasta llegar a Q. Esto incrementa el tiempo de ejecución de manera cuadrática como se observa en la Figura. 1.

Figura 1. *Desempeño del algoritmo Bottom-Up en el benchmark completo bajo diferentes configuraciones de BF y Q.*

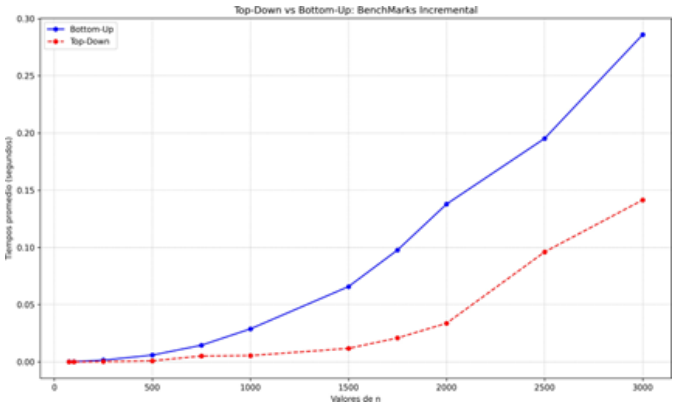


VII-B. Desempeño de los algoritmos con el benchmark incremental

En este benchmark las reglas acumulan los literales de las reglas anteriores para formar una estructura lineal jerárquica. Ahora bien, los algoritmos Bottom-Up y Top-Down presentan una complejidad cuadrática $O(n^2)$ ya que ambos deben procesar todas las dependencias jerárquicas para deducir la proposición final.

A pesar de compartir la misma complejidad, el algoritmo Top-Down es más eficiente en este caso debido a su capacidad para restringir el espacio de búsqueda compensando las constantes asociadas a sus estructuras más complejas. Por el contrario, el algoritmo Bottom-Up es más adecuado en escenarios que requieren un recorrido sistemático y completo de todas las reglas. Estas características se reflejan claramente en los tiempos promedios obtenidos como se ilustra en la Figura. 2.

Figura 2. *Comparación del desempeño de los algoritmos Top-Down y Bottom-Up en el benchmark tipo incremental.*

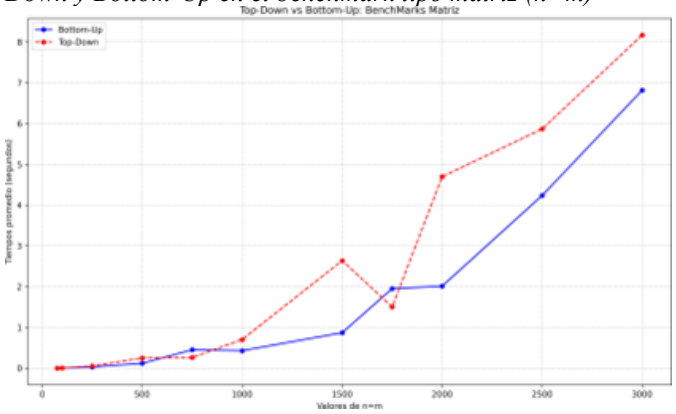


VII-C. Desempeño de los algoritmos con el benchmark matriz

En el benchmark matriz las reglas se organizan en m filas y n columnas. El algoritmo Top-Down presenta una complejidad lineal $O(m)$ como se representa en la tabla de complejidades. Sin embargo, los resultados obtenidos muestran un comportamiento ligeramente ruidoso lo que podría ser; indicativo de que se requieren más iteraciones o ajustes en el algoritmo para capturar con precisión una tendencia lineal.

En contraste el algoritmo Bottom-Up tiene una complejidad cuadrática $O(n^2)$ lo cual es consistente con los resultados observados en la Figura 3 que muestran una curva parabólica característica de este comportamiento.

Figura 3. *Comparación del desempeño de los algoritmos Top-Down y Bottom-Up en el benchmark tipo matriz (n=m)*



VIII. CONCLUSIONES

En este artículo se desarrolló un análisis detallado de los algoritmos Bottom-Up y Top-Down evaluados en los siguientes benchmarks bajo diferentes configuraciones: Completo, Incremental y Matriz. Se presentaron las complejidades teóricas de ambos algoritmos y se evaluó su desempeño experimental utilizando para ello estructuras representadas en formato DIMACS y explorando diversas combinaciones de preguntas y hechos iniciales.

Los resultados obtenidos permiten concluir que el rendimiento relativo de los algoritmos depende directamente de

la estructura del benchmark y de las reglas evaluadas. En el caso del benchmark tipo matriz el algoritmo del Bottom-Up demostró ser más eficiente que el algoritmo del Top-Down, debido a que las constantes asociadas al Top- Down son mayores que las del Bottom-Up. Esto se debe a que la recursión del Top-Down incrementa estas constantes, mientras que el Bottom-Up únicamente decrece un contador en cada iteración. Aunque el Top-Down tiene una complejidad lineal $O(m)$ y el Bottom-Up es cuadrático $O(n^2)$ las constantes del Top-Down afectan su desempeño práctico haciendo que en este caso el Bottom-Up sea más rápido. Sin embargo; esta mejora del Bottom-Up es mínima y cuando este algoritmo es peor su desempeño puede llegar a ser exponencialmente más deficiente en comparación con el Top-Down.

En el benchmark incremental el Top-Down resultó superior a pesar de que ambos algoritmos comparten una complejidad cuadrática $O(n^2)$. Esta mejora del Top-Down en el benchmark incremental se debe a que sobresale en situaciones donde su capacidad de restringir el espacio de búsqueda compensa las constantes asociadas a sus estructuras más complejas; mientras que el Bottom-Up es más efectivo en contextos donde se requiere un recorrido sistemático y completo de las reglas. Sin embargo, se reafirma que ninguno de los dos algoritmos es absolutamente mejor en todos los casos.

Reconocimiento

El autor agradece al CINVESTAV-IPN por el curso impartido de Inteligencia Artificial en el marco de la Maestría en Ciencias en Computación y al Dr. Gonzalo Escalada Imaz por su dedicación, guía y enseñanza en el diseño de estos algoritmos, cuyos conocimientos fueron esenciales para la elaboración de este artículo.

REFERENCIAS

[1] W. Dowling and J. Gallier, “Linear-time algorithms for testing the satisfiability of propositional Horn formulae,” Journal of Logic Programming, vol. unknown, no. 3, pp. 267–284, 1984.

[2] M. Ghallab and G. Escalada-Imaz, “A linear control algorithm for a class of rule-based systems,” Journal of Logic Programming, vol. unknown, no. 11, pp. 117–132, 1991.

[3] G. E. Imaz, “Solving horn-sat top-down and linearly,” 2024, unpublished Draft (ongoing work).