Comparación algoritmos

Sesión 16

Edgar Andrade, PhD

Mayo de 2019

Departmento de Matemáticas Aplicadas y Ciencias de la Computación

Presentación

En esta sesión estudiaremos:

1. Complejidad computacional de los algoritmos (tablas, tableaux, DPLL).
2. Tiempos de ejecución de tablas, tableaux, DPLL.
3. Complejidad computacional del problema SAT.

Contenido

1. Complejidad tablas, tableaux, DPLL
2. Tiempos de ejecución tablas, tableaux, DPLL
3. Complejidad problema SAT

Complejidad computacional

Definición

La complejidad computacional de un problema bien definido es el número de operaciones algorítmicas requeridas para resolverlo.

Complejidad computacional

Definición

La complejidad computacional de un problema bien definido es el número de operaciones algorítmicas requeridas para resolverlo.

Precisamos el problema

1. Encontrar todos los modelos de una fórmula dada.

Precisamos el problema

1. Encontrar todos los modelos de una fórmula dada.
2. Determinar si una fórmula dada es satisfacible.

Precisamos el problema

1. Encontrar todos los modelos de una fórmula dada.
2. Determinar si una fórmula dada es satisfacible.
3. Verificar si una interpretación es un modelo de una fórmula dada.

Problema 1

```
def tablaVerdad(A, letrasProposicionales):

    interps = crear_interpretaciones(letrasProposicionales)

    lst = []
    for i in interps:
        if V_I(A, i) == 1:
            lst.append(i)

    return lst
```

Problema 1

```
def tablaVerdad(A, letrasProposicionales):

    interps = crear_interpretaciones(letrasProposicionales)

    lst = []
    for i in interps:
        if V_I(A, i) == 1:
            lst.append(i)

    return lst
```

crear_interpretaciones

```
def crear_interpretaciones(letrasProposicionales):
    interps = []
    aux = {}
    for a in letrasProposicionales:
        aux[a] = 1
        interps.append(aux)
    for a in letrasProposicionales:
        interps_aux = [i for i in interps]
        for i in interps_aux:
            aux1 = {}
            for b in letrasProposicionales:
                if a == b:
                    aux1[b] = 1 - i[b]
                else:
                    aux1[b] = i[b]
            interps.append(aux1)
    return interps
```

crear_interpretaciones

```
def crear_interpretaciones(letrasProposicionales):
    interps = []
    aux = {}
    for a in letrasProposicionales:
        aux[a] = 1
    interps.append(aux)
    for a in letrasProposicionales:
        interps_aux = [i for i in interps]
        for i in interps_aux:
            aux1 = {}
            for b in letrasProposicionales:
                if a == b:
                    aux1[b] = 1 - i[b]
                else:
                    aux1[b] = i[b]
            interps.append(aux1)
    return interps
```

crear_interpretaciones

```
def crear_interpretaciones(letrasProposicionales):
    interps = []
    aux = {}
    for a in letrasProposicionales:
        aux[a] = 1                                      2N
        interps.append(aux)
    for a in letrasProposicionales:
        interps_aux = [i for i in interps]
        for i in interps_aux:
            aux1 = {}
            for b in letrasProposicionales:
                if a == b:
                    aux1[b] = 1 - i[b]
                else:
                    aux1[b] = i[b]
                interps.append(aux1)
    return interps
```

crear_interpretaciones

```
def crear_interpretaciones(letrasProposicionales):
    interps = []
    aux = {}
    for a in letrasProposicionales:
        aux[a] = 1                                    2N
        interps.append(aux)
    for a in letrasProposicionales:
        interps_aux = [i for i in interps]
        for i in interps_aux:
            aux1 = {}
            for b in letrasProposicionales:
                if a == b:
                    aux1[b] = 1 - i[b]
                else:
                    aux1[b] = i[b]
            interps.append(aux1)
    return interps
```

crear_interpretaciones

```
def crear_interpretaciones(letrasProposicionales):
    interps = []
    aux = {}
    for a in letrasProposicionales:
        aux[a] = 1                                  2N
        interps.append(aux)
    for a in letrasProposicionales:                 N*
        interps_aux = [i for i in interps]
        for i in interps_aux:
            aux1 = {}
            for b in letrasProposicionales:
                if a == b:
                  aux1[b] = 1 - i[b]
                else:
                  aux1[b] = i[b]
                interps.append(aux1)
    return interps
```

crear_interpretaciones

```
def crear_interpretaciones(letrasProposicionales):
    interps = []
    aux = {}
    for a in letrasProposicionales:
        aux[a] = 1                                          2N
        interps.append(aux)
    for a in letrasProposicionales:                         N*
        interps_aux = [i for i in interps]
        for i in interps_aux:
            aux1 = {}
            for b in letrasProposicionales:
                if a == b:
                  aux1[b] = 1 - i[b]
                else:                                       N(1 + 1 + 1)
                  aux1[b] = i[b]
                interps.append(aux1)
    return interps
```

crear_interpretaciones

```
def crear_interpretaciones(letrasProposicionales):
    interps = []
    aux = {}
    for a in letrasProposicionales:
        aux[a] = 1                                  2N
        interps.append(aux)
    for a in letrasProposicionales:                 N*
        interps_aux = [i for i in interps]
        for i in interps_aux:
            aux1 = {}
            for b in letrasProposicionales:
                if a == b:
                    aux1[b] = 1 - i[b]
                else:                               3N
                    aux1[b] = i[b]
                interps.append(aux1)
    return interps
```

crear_interpretaciones

```
def crear_interpretaciones(letrasProposicionales):
    interps = []
    aux = {}
    for a in letrasProposicionales:
        aux[a] = 1                                   2N
        interps.append(aux)
    for a in letrasProposicionales:                  N*
        interps_aux = [i for i in interps]
        for i in interps_aux:
            aux1 = {}
            for b in letrasProposicionales:
                if a == b:
                    aux1[b] = 1 - i[b]
                else:                                3N
                    aux1[b] = i[b]
                interps.append(aux1)
    return interps
```

crear_interpretaciones

```
def crear_interpretaciones(letrasProposicionales):
    interps = []
    aux = {}
    for a in letrasProposicionales:
        aux[a] = 1                                  2N
        interps.append(aux)
    for a in letrasProposicionales:                 N*
        interps_aux = [i for i in interps]
        for i in interps_aux:                       2^N*
            aux1 = {}
            for b in letrasProposicionales:
                if a == b:
                    aux1[b] = 1 - i[b]
                else:                               3N
                    aux1[b] = i[b]
                interps.append(aux1)
    return interps
```

crear_interpretaciones

```
def crear_interpretaciones(letrasProposicionales):
    interps = []
    aux = {}
    for a in letrasProposicionales:
        aux[a] = 1
        interps.append(aux)
    for a in letrasProposicionales:
        interps_aux = [i for i in interps]
        for i in interps_aux:
            aux1 = {}
            for b in letrasProposicionales:
                if a == b:
                    aux1[b] = 1 - i[b]
                else:
                    aux1[b] = i[b]
                interps.append(aux1)
    return interps
```

$$2N + N * (2^N * 3N)$$

crear_interpretaciones

```
def crear_interpretaciones(letrasProposicionales):
    interps = []
    aux = {}
    for a in letrasProposicionales:
        aux[a] = 1
        interps.append(aux)
    for a in letrasProposicionales:
        interps_aux = [i for i in interps]
        for i in interps_aux:
            aux1 = {}
            for b in letrasProposicionales:
                if a == b:
                    aux1[b] = 1 - i[b]
                else:
                    aux1[b] = i[b]
                interps.append(aux1)
    return interps
```

$$2N + 2^N(2N^2)$$

Problema 1

```
def tablaVerdad(A, letrasProposicionales):

    interps = crear_interpretaciones(letrasProposicionales)

    lst = []
    for i in interps:
        if V_I(A, i) == 1:
            lst.append(i)

    return lst
```

Problema 1

```
def tablaVerdad(A, letrasProposicionales):

    interps = crear_interpretaciones(letrasProposicionales)
```

$2N + 2^N(2N^2)$

```
    lst = []
    for i in interps:
        if V_I(A, i) == 1:
            lst.append(i)

    return lst
```

Problema 1

```
def tablaVerdad(A, letrasProposicionales):

    interps = crear_interpretaciones(letrasProposicionales)
```

$2N + 2^N(2N^2)$

```
    lst = []
    for i in interps:
        if V_I(A, i) == 1:
            lst.append(i)

    return lst
```

Problema 1

def tablaVerdad(A, letrasProposicionales):

    interps = crear_interpretaciones(letrasProposicionales) $2N + 2^N(2N^2)$

    lst = []
    for i in interps: $2(2^N)$
        if V_I(A, i) == 1:
            lst.append(i)

    return lst

Tablas de verdad — Problema 2

```
def satisfacible(A, letrasProposicionales):

    interps = crear_interpretaciones(letrasProposicionales)

    for i in interps:
        if V_I(A, i) == 1:
            return "Satisfacible"
```

Tablas de verdad — Problema 2

```
def satisfacible(A, letrasProposicionales):

    interps = crear_interpretaciones(letrasProposicionales)

    for i in interps:
        if V_I(A, i) == 1:
            return "Satisfacible"
```

☞ Para tablas de verdad, siempre estamos en el peor escenario. El problema 2 es igual de difícil que el problema 1.

Problema 2

Para el DPLL:

- Si se considera el peor escenario, el problema 2 es igual de difícil que el problema 1.

Problema 2

Para el DPLL:

- Si se considera el peor escenario, el problema 2 es igual de difícil que el problema 1.
- En muchos escenarios, el problema 2 es igual de fácil al problema 3.

Problema 3

```
def modelo(A, i):

    if V_I(A, i) == 1:
      return True
```

Problema 3

```
def modelo(A, i):

    if V_I(A, i) == 1:
        return True
```

El problema 3 es "fácil"

Para cualquier A, es posible resolver el problema 3 en un tiempo polinomial respecto al número de conectivos de A.

Contenido

1 Complejidad tablas, tableaux, DPLL

2 Tiempos de ejecución tablas, tableaux, DPLL

3 Complejidad problema SAT

Calculando empíricamente el tiempo de ejecución

Condición A: $p \wedge q$

Condición B:

Condición C:

Condición D:

Calculando empíricamente el tiempo de ejecución

Condición A: $p \wedge q \quad (p \wedge q) \wedge r$

Condición B:

Condición C:

Condición D:

Calculando empíricamente el tiempo de ejecución

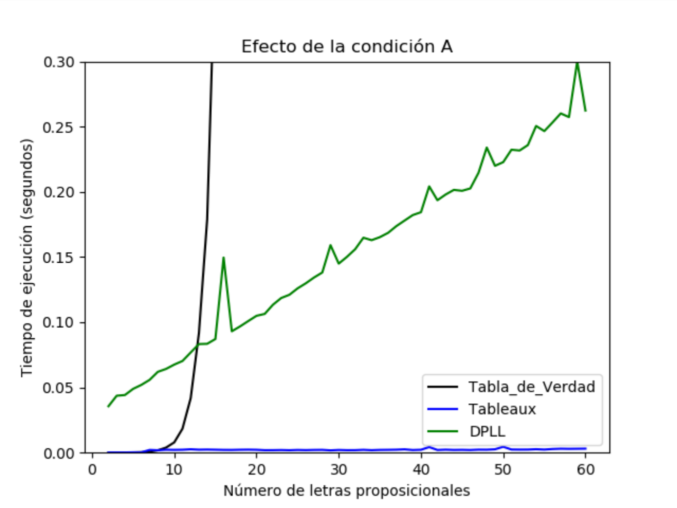
Condición A: $p \wedge q \quad (p \wedge q) \wedge r \quad ((p \wedge q) \wedge r) \wedge s \quad \ldots$

Condición B:

Condición C:

Condición D:

Tiempos de ejecución: Condición A



Calculando empíricamente el tiempo de ejecución

Condición A: $p \wedge q \quad (p \wedge q) \wedge r \quad ((p \wedge q) \wedge r) \wedge s \quad \ldots$

Condición B:

Condición C: $p \vee q$

Condición D:

Calculando empíricamente el tiempo de ejecución

Condición A: $p \wedge q \quad (p \wedge q) \wedge r \quad ((p \wedge q) \wedge r) \wedge s \quad \ldots$

Condición B:

Condición C: $p \vee q \quad (p \vee q) \wedge (\neg p \vee \neg q)$

Condición D:

Calculando empíricamente el tiempo de ejecución

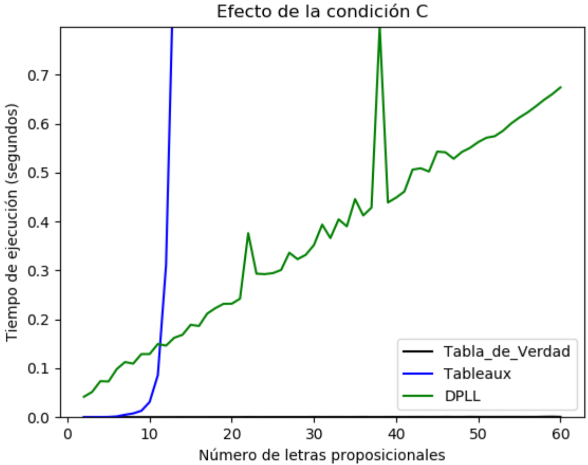
Condición A: $p \wedge q \quad (p \wedge q) \wedge r \quad ((p \wedge q) \wedge r) \wedge s \quad \ldots$

Condición B:

Condición C: $p \vee q \quad (p \vee q) \wedge (\neg p \vee \neg q) \quad ((p \vee q) \wedge (\neg p \vee \neg q)) \wedge (\neg p \vee \neg q) \ldots$

Condición D:

Tiempos de ejecución: Caso C



Calculando empíricamente el tiempo de ejecución

Condición A: $p \wedge q \quad (p \wedge q) \wedge r \quad ((p \wedge q) \wedge r) \wedge s \quad \ldots$

Condición B: $p \vee q$

Condición C: $p \vee q \quad (p \vee q) \wedge (\neg p \vee \neg q) \quad ((p \vee q) \wedge (\neg p \vee \neg q)) \wedge (\neg p \vee \neg q) \ldots$

Condición D:

Calculando empíricamente el tiempo de ejecución

Condición A: $p \wedge q \quad (p \wedge q) \wedge r \quad ((p \wedge q) \wedge r) \wedge s \quad \ldots$

Condición B: $p \vee q \quad (p \vee q) \wedge (p \vee r)$

Condición C: $p \vee q \quad (p \vee q) \wedge (\neg p \vee \neg q) \quad ((p \vee q) \wedge (\neg p \vee \neg q)) \wedge (\neg p \vee \neg q) \ldots$

Condición D:

Calculando empíricamente el tiempo de ejecución

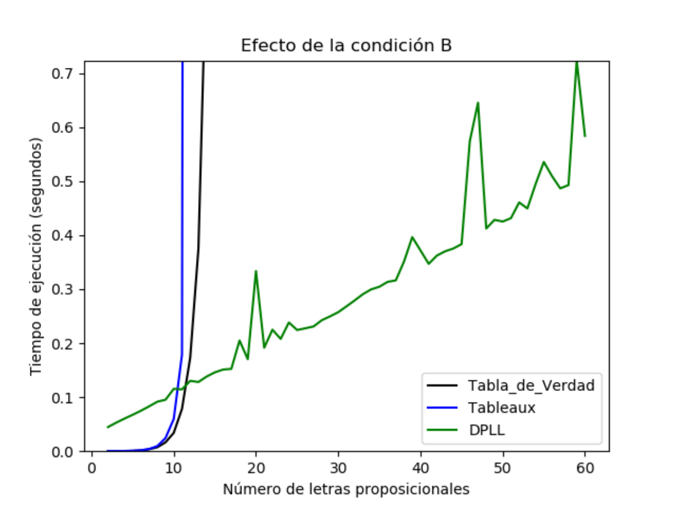
Condición A: $p \wedge q \quad (p \wedge q) \wedge r \quad ((p \wedge q) \wedge r) \wedge s \quad \ldots$

Condición B: $p \vee q \quad (p \vee q) \wedge (p \vee r) \quad ((p \vee q) \wedge (p \vee r)) \wedge (p \vee s) \ldots$

Condición C: $p \vee q \quad (p \vee q) \wedge (\neg p \vee \neg q) \quad ((p \vee q) \wedge (\neg p \vee \neg q)) \wedge (\neg p \vee \neg q) \ldots$

Condición D:

Tiempos de ejecución: Caso B



Calculando empíricamente el tiempo de ejecución

Condición A: $p \wedge q \quad (p \wedge q) \wedge r \quad ((p \wedge q) \wedge r) \wedge s \quad \ldots$

Condición B: $p \vee q \quad (p \vee q) \wedge (p \vee r) \quad ((p \vee q) \wedge (p \vee r)) \wedge (p \vee s) \ldots$

Condición C: $p \vee q \quad (p \vee q) \wedge (\neg p \vee \neg q) \quad ((p \vee q) \wedge (\neg p \vee \neg q)) \wedge (\neg p \vee \neg q) \ldots$

Condición D: $p \wedge q$

Calculando empíricamente el tiempo de ejecución

Condición A: $p \wedge q \quad (p \wedge q) \wedge r \quad ((p \wedge q) \wedge r) \wedge s \quad \ldots$

Condición B: $p \vee q \quad (p \vee q) \wedge (p \vee r) \quad ((p \vee q) \wedge (p \vee r)) \wedge (p \vee s) \ldots$

Condición C: $p \vee q \quad (p \vee q) \wedge (\neg p \vee \neg q) \quad ((p \vee q) \wedge (\neg p \vee \neg q)) \wedge (\neg p \vee \neg q) \ldots$

Condición D: $p \wedge q \quad (p \wedge q) \vee (p \wedge r)$

Calculando empíricamente el tiempo de ejecución

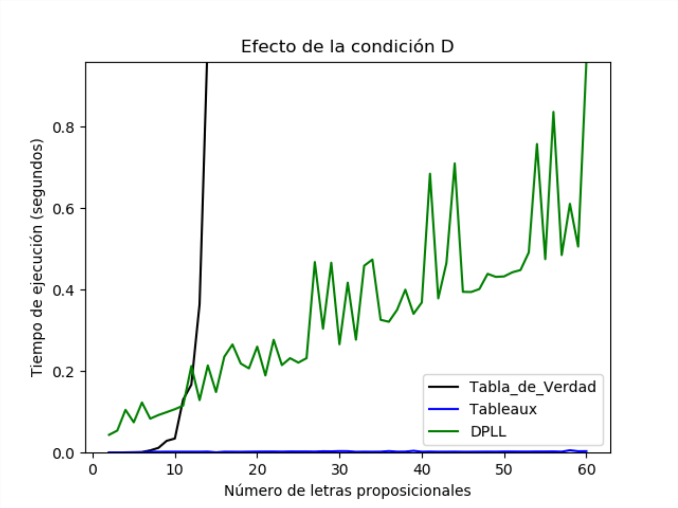
Condición A: $p \wedge q \quad (p \wedge q) \wedge r \quad ((p \wedge q) \wedge r) \wedge s \quad \dots$

Condición B: $p \vee q \quad (p \vee q) \wedge (p \vee r) \quad ((p \vee q) \wedge (p \vee r)) \wedge (p \vee s) \dots$

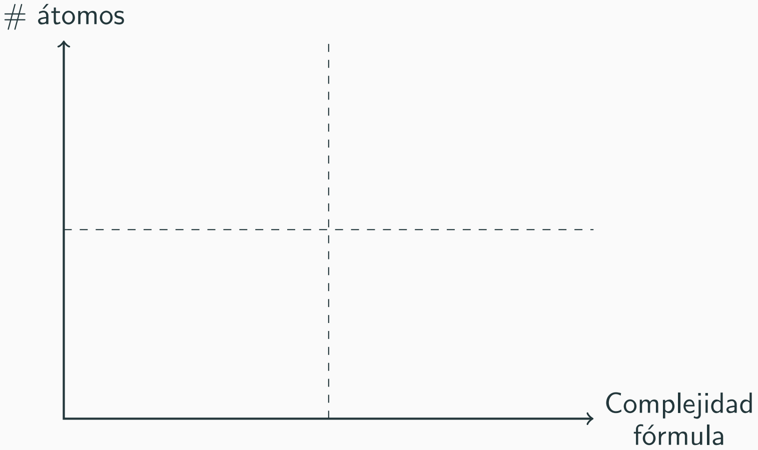
Condición C: $p \vee q \quad (p \vee q) \wedge (\neg p \vee \neg q) \quad ((p \vee q) \wedge (\neg p \vee \neg q)) \wedge (\neg p \vee \neg q) \dots$

Condición D: $p \wedge q \quad (p \wedge q) \vee (p \wedge r) \quad ((p \wedge q) \vee (p \wedge r)) \vee (p \wedge s) \dots$

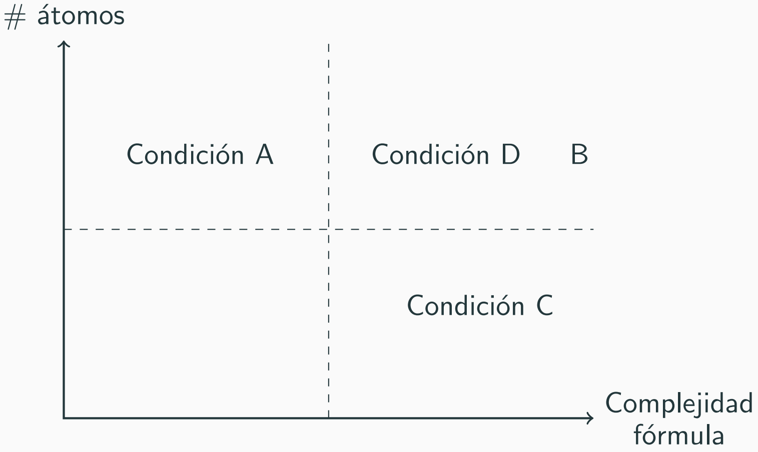
Tiempos de ejecución: Caso D



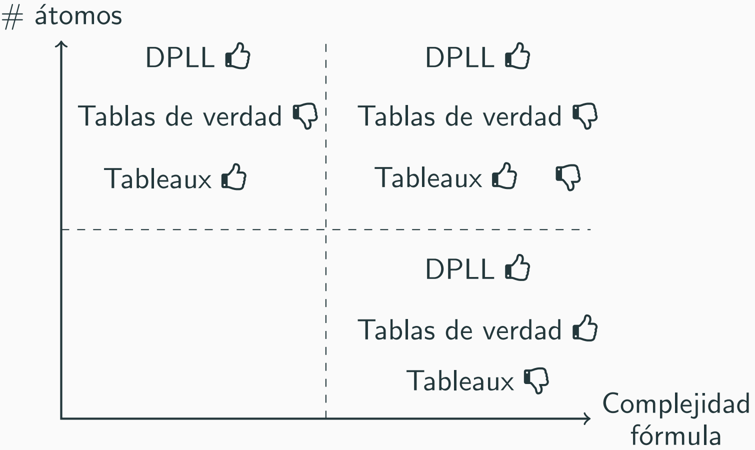
Efecto de los parámetros



Efecto de los parámetros



Efecto de los parámetros



Contenido

1 Complejidad tablas, tableaux, DPLL

2 Tiempos de ejecución tablas, tableaux, DPLL

3 Complejidad problema SAT

El problema sat

Dada una fórmula en FNC, determinar si ella es satisfacible o no.

El problema sat

Dada una fórmula en FNC, determinar si ella es satisfacible o no.

¿sat es P?

P es el conjunto de los problemas de decisión que pueden resolverse mediante computaciones deterministas en tiempo polinomial.

☞ Todos los algoritmos conocidos hasta el momento toman un tiempo exponencial en ser resueltos en el peor escenario.

El problema sat

Dada una fórmula en FNC, determinar si ella es satisfacible o no.

¿sat es P?

P es el conjunto de los problemas de decisión que pueden resolverse mediante computaciones deterministas en tiempo polinomial.

☞ Todos los algoritmos conocidos hasta el momento toman un tiempo exponencial en ser resueltos en el peor escenario.

El problema sat

Dada una fórmula en FNC, determinar si ella es satisfacible o no.

¿sat es P?

P es el conjunto de los problemas de decisión que pueden resolverse mediante computaciones deterministas en tiempo polinomial.

Todos los algoritmos conocidos hasta el momento toman un tiempo exponencial en ser resueltos en el peor escenario.

sat sí es NP

NP es el conjunto de los problemas de decisión para el cual las instancias afirmativas tienen pruebas que pueden hacerse mediante computaciones deterministas en tiempo polinomial.

Fin de la sesión 16

En esta sesión usted ha aprendido a:

1. Representar de manera abstracta la complejidad computacional de tablas, tableaux y DPLL.
2. Comparar los tiempos de ejecución de tablas, tableaux y DPLL.
3. Comprender los conceptos básicos de la complejidad computacional del problema SAT.