# Un ambiente Integrado de Ejecución para determinar la Satisfacibilidad de Fórmulas en Forma Normal Conjuntiva

Dar·o Funez, Patricia Roggero, Guillermo Leguizamon

LIDIC
Departamento de Informatica
Universidad Nacional de San Luis
Ejercito de los Andes 950
D5700HHW - San Luis - Argentina
Tel: 02652-420823 Fax: 02652-430224
e-mail: {dgfunez,proggero,legui}@unsl.edu.ar

**Abstract**

The satisfiability problem (SAT) is a classical $\mathcal{NP}$-complete problem of useful application in plenty of computer science areas, electric engineering, and mathematics. In this work we present an integration of all the components of the DPLL system enhanced with Neural Networks (DPLL_NN) in order to create a user friendly execution environment. One of the main components of DPLL_NN is an algorithm that is capable of solving some instances of the 3SAT problem, a well-known $\mathcal{NP}$-complete problem derived from SAT. DPLL_NN combines the *Davis-Putnam* rules with neural networks which are capable enough to learn complex problems. The environment proposed allows the user determine the satisfiability of a particular formula either randomly or manually generated. In addition, the environment provides a graphical tree representation of the corresponding assignation values that satisfy that formula (if it can be satisfied).

**Resumen**

El problema de satisfacibilidad (SAT), el primero en demostrarse que es $\mathcal{NP}$-completo, es fundamental en muchos campos de las ciencias de la computación, la ingeniería eléctrica y matemática. En el presente trabajo se integran todas las componentes del sistema DPLL con Redes Neuronales (DPLL_RN), para crear un ambiente de ejecución amigable al usuario. Una de las componentes fundamentales del sistema DPLL_RN consta de un algoritmo que permite resolver algunas instancias del problema 3SAT, un conocido problema $\mathcal{NP}$-completo derivado de SAT. El sistema DPLL_RN combina las reglas de *Davis-Putnam* con redes neuronales (DPRN), las cuales tienen una gran capacidad para aprender problemas complejos. El ambiente propuesto permite al usuario determinar la satisfacibilidad o no de una fórmula en lógica proposicional, permitiéndole ingresar una fórmula cualquiera o generar una aleatoriamente y luego se refleja en un árbol el camino seguido para obtener las asignaciones que hacen verdadera a dicha fórmula si tales asignaciones existen.

# 1. INTRODUCCIÓN

Los problemas $\mathcal{NP}$-completos son problemas para los cuales no se conoce un algoritmo de tiempo polinomial determin·stico que los pueda resolver. El ejemplo canonico de un problema $\mathcal{NP}$-completo es el problema de satisfacibilidad booleana (SAT) [4, 9, 2, 12].

SAT es un problema fundamental en logica matematica, teor·a de computacion, inferencia, razonamiento automatizado, ingenier·a VLSI, entre otros. Dicho problema se re ere a la tarea de encontrar asignaciones de verdad que hagan a una expresion booleana verdadera. Su importancia reside basicamente en el hecho de que es el primer problema, y uno de los mas simples, para el cual se demostro que es $\mathcal{NP}$-completo. Ademas muchos problemas practicos, tales como demostracion automatica de teoremas, veri cacion y deteccion de fallas de hardware, deteccion de isomor smo de grafos, el problema del viajante de comercio, etc [1], son problemas $\mathcal{NP}$-completos y se pueden transformar de manera e ciente a SAT. Por lo tanto, buenos algoritmos para SAT son indudablemente de gran utilidad. Dado que es poco probable que se pueda encontrar un algoritmo de tiempo polinomial para resolver SAT, buenas soluciones parecen importantes.

Por lo expuesto anteriormente es que se desarrollo el sistema DPLL_RN [11] cuya nalidad fue la de tratar de hallar un algoritmo que resuelva instancias del problema 3SAT, combinando uno de los algorimos practicos mas utilizados en la actualidad, el algoritmo *Davis-Putnam-Logemann-Loveland* (DPLL) [3, 13], cuya e ciencia depende de la eleccion de una regla de rami cacion basada en *Redes Neuronales* (RN) las que permiten realizar en forma heur·stica la seleccion de las variables y su correspondiente asignacion de valores de verdad. Una de las principales caracter·sticas de las RN en general esta dada por su gran capacidad para aprender a resolver problemas complejos y por poseer la habilidad de generalizacion, dos propiedades importantes a la hora de tratar de resolver instancias del problema de satisfacibilidad. Por otro lado, cabe destacar que una de las principales motivaciones para el desarrollo del sistema DPLL_RN es que el mismo permite trabajar con distintos tipos de instancias del problema de satisfacibilidad lo que puede ser de gran interes para entender la complejidad del mismo. Sin embargo, el problema fundamental de DPLL_RN es la falta de un ambiente amigable que permitiera su utilizacion, sin necesidad de conocer las caracter·sticas propias de la implementacion de este sistema, dado que el mismo requiere de una serie de pasos que pueden resultar tediosos cuando se pretende utilizarlo solo para nes practicos, i.e., obtener asignaciones de verdad para una formula. Es por ello que surge la necesidad de desarrollar un ambiente que facilite el uso del sistema DPLL_RN que oculte al usuario los detalles de implementacion del mismo y poder utilizarlo sin tener conocimiento sobre redes neuronales y el algoritmo DPLL, permitiendo de esta manera su utilizacion por parte de un amplio espectro de usuarios interesados en aprovechar los bene cios de DPLL_RN.

Este art·culo esta organizado como sigue. La seccion 2 describe brevemente el problema SAT. En la seccion 3 se muestra el algorimo DPLL. La seccion 4 presenta las caracter·sticas mas relevantes del sistema DPLL_RN. La seccion 5 describe el ambiente integrado de ejecucion y por ultimo, en la seccion 6 se presentan las conclusiones.

# 2. EL PROBLEMA DE SATISFACIBILIDAD (SAT)

El problema SAT es fundamental en muchos campos de las ciencias de la computacion, la ingenier·a electrica y matematica. Dicho problema fue el primero en demostrarse que es $\mathcal{NP}$-completo. El problema de satisfacibilidad consiste en la asignacion de valores de verdad a formulas en logica proposicional. El objetivo del problema de satisfacibilidad es determinar si existe una asignacion de valores de verdad para las variables booleanas que hagan verdadera a una formula.

Una variable booleana es una variable que toma alguno de los valores $0$ (falso) o $1$ (verdadero).

Las variables booleana se consideran proposiciones, que son los objetos elementales de la logica proposicional. El valor de la variable especi ca la verdad o falsedad de la proposicion. La proposicion $x$ es verdadera cuando a la variable booleana se le asigna el valor $1$, cuando el valor asignado es $0$ la proposicion es falsa. Una *asignación de verdad* [1] es una funcion que asigna $0$ o $1$ a toda variable booleana.

Las conectivas logicas $\wedge$ (conjuncion), $\vee$ (disyuncion) y $\neg$ (negacion) se usan para construir proposiciones llamadas formulas bien formadas a partir de un conjunto de variables booleana.

**Definición 1** *Sea $V$ un conjunto de variables booleanas:*

1. *Si $x \in V$, entonces $x$ es una fórmula bien formada.*
2. *Si $x$, $y \in V$, entonces $(\neg x)$, $(x \wedge y)$ y $(x \vee y)$ son fórmulas bien formadas.*
3. *Una expresión es una fórmula bien formada sobre $V$ solamente si puede ser obtenida a partir de las variables booleanas en el conjunto $V$ por un número finito de aplicaciones de las operaciones dadas en 2.*

Se dice que una formula $u$ satisface una asignacion de verdad si los valores de las variables hacen que $u$ tenga el valor $1$. Una *cláusula* es una formula bien formada que consiste de una disyuncion de variables o la negacion de variables. Una variable no negada se llama literal positivo y una variable negada literal negativo. Usando esta terminolog·a, una clausula es una disyuncion de literales, las formulas $x \vee \neg y$, $\neg x \vee z \vee \neg y$ y $x \vee z \vee \neg x$ son clausulas sobre el conjunto de variables $V = \{x, y, z\}$.

Una formula esta en *forma normal conjuntiva* si tiene la forma:

$$u = u_1 \wedge u_2 \wedge ..... \wedge u_m$$

donde cada $u_i$ es una clausula. Un teorema clasico de logica proposicional asegura que toda formula bien formada se puede transformar en una formula equivalente en forma normal conjuntiva. Entonces, de manera rigurosa *el problema de satisfacibilidad* es el problema de decidir si una formula en forma normal conjuntiva se satisface por alguna asignacion de verdad.

*Ejemplo:* Sea $V = \{x, y, z\}$ y las formulas $u$, $v$ y $w$:

$$u = (x \vee y) \wedge (\neg y \vee \neg z)$$
$$v = (x \vee \neg y \vee \neg z) \wedge (x \vee z) \wedge (\neg x \vee \neg y)$$
$$w = \neg x \wedge (x \vee y) \wedge (\neg y \vee x)$$

las formulas $u$ y $v$ se satisfacen por la asignacion de verdad $t(x) = T$, $t(y) = F$ y $t(z) = T$; donde $t : V \to \{T, F\}$ es la funcion de asignacion de valores de verdad de nida sobre el conjunto $V$. De esta manera, la primera clausula en $u$ es satisfecha por $x$ y la segunda por $\neg y$. La formula $w$ no es satisfecha por $t$, es mas, no es dif·cil ver que $w$ no es satisfecha por ninguna asignacion de verdad.

**Definición 2** *Una fórmula se dice que está en 3-forma normal conjuntiva, si cada cláusula contiene exactamente tres literales.*

[1] También se llama valuación.

Un caso particular derivado del problema SAT es el 3-SAT donde el objetivo del problema 3-SAT es determinar si una formula en forma normal conjuntiva (con tres literales por clausula) es satisfacible. Un aspecto muy importante en este contexto es que cualquier formula en forma normal conjuntiva puede ser transformada a una formula en forma normal conjuntiva con tres literales por clausula. Esto da la fundamentacion teorica necesaria para trabajar en DPLL_RN con formulas con tres literales por clausula y dado que el problema 3-SAT presenta una formulacion mas sencilla que SAT, es un problema usado comunmente en la bibliograf·a de teor·a de complejidad computacional para mostrar la existencia de otros problemas $\mathcal{NP}$-completos [5]. Ademas, se puede observar, que formulas expresadas de manera que todas las clausulas tengan la misma cantidad de variables son, de alguna manera, mas claras.

## 3. ALGORITMO DPLL

Un metodo simple para veri car satisfacibilidad de una formula, es generar todos los posibles resolventes, y entonces veri car si se genera la clausula vac·a. Davis y Putnam [4] introdujeron un metodo en el cual las variables se eliminan una a una de la formula. En cada paso se generan todos los posibles resolventes (basados en la eleccion de una variable) y entonces se eliminan todas las clausulas que mencionan a aquella variable. Cada paso genera un subproblema con unas pocas variables menos pero, posiblemente, con mas clausulas.

Por ejemplo, dadas dos clausulas $C_1 = (v \vee x_1 \vee ... \vee x_p)$ y $C_2 = (\neg v \vee y_1 \vee ... \vee y_r)$, donde $x_i \neq y_j$, para $i = 1, .., p$ y $j = 1, ..r$, el x *resolvente* de $C_1$ y $C_2$ es la clausula $(x_1 \vee ... \vee x_p \vee y_1 \vee ... \vee y_r)$, esto es, la disyuncion de $C_1$ y $C_2$ sin $v$ y $\neg v$. El resolvente es una consecuencia logica de la conjuncion del par de clausulas. La resolucion es el proceso de generar repetidamente resolventes desde las clausulas originales y los resolventes previamente generados, hasta que la clausula nula sea generada o no se puedan crear mas resolventes. En el primer caso la formula es no satisfacible y en el ultimo caso es satisfacible.

El algoritmo DPLL [3, 13, 12] es una variacion al algoritmo original propuesto por Davis y Putnam (DP), dicha variacion es mas e ciente que el metodo original para propositos practicos. DPLL usa una *regla de splitting*, la cual permite reemplazar al problema original por dos subproblemas mas pequenos, y ademas, permite determinar cual es la variable a seleccionar. Esencialmente, el algoritmo es una busqueda *depth-first con backtracking* a traves de las asignaciones de verdad (parciales). Este algoritmo tiene un fuerte crecimiento exponencial. En el peor caso, para una formula no satisfacible, DPLL buscara a traves de todas las asignaciones de verdad que puedan ser hechas.

Dos heur·sticas son aplicadas durante este proceso (ver Algoritmo 1) con la idea de podar el espacio de busqueda. Estas, son *la regla del literal puro* y *la cláusula unitaria*.

- La regla del literal puro se aplica cuando una variable aparece con un unico valor de verdad en todas las clausulas aun no satisfechas. En este caso, se asigna un valor de verdad a dicha variable de manera que todas las clausulas involucradas sean satisfechas. Este es un procedimiento altamente efectivo en la busqueda.

- Si alguna variable ocurre en la formula corriente en una clausula de longitud 1, entonces la regla de la clausula unitaria se aplica. El literal se selecciona y se le asigna un valor de verdad de manera que la clausula respectiva sea satisfecha. La aplicacion repetida de la regla unitaria comunmente se conoce como *Unit Propagation* o *Boolean Constraint Propagation (BCP)*.

El algoritmo DPLL trabaja por prueba y error, gra camente se puede observar la secuencia de pasos, utilizando un arbol de busqueda, que se forma de la siguiente manera:

**Algorithm 1** Algoritmo DPLL (con heur·sticas) en pseudoc◌digo

```
Procedure DPLL (CNF formula:φ)
 if (φ es vacía) return si
 else if (hay una clausula vacía en φ) return no
 else if (hay un literal puro v en φ) return DPLL(φ(v))
 else if (hay una clausula unitaria {v} en φ) return DPLL(φ(v))
 else seleccionar una variable v mencionada en φ
  if DPLL(φ(v) = si) then return sí
   else return DPLL(φ(¬v))
end
```

1. Un nodo corresponde a la eleccion de una variable. Dependiendo de este nodo, DPLL toma una de dos acciones posibles y esto se re eja con un descendiente (arco) en el arbol.

2. A partir de un arco, se extraen todas las implicaciones logicas que se derivan de la ultima eleccion tomada.

3. Se vuelve a 1, a menos que encuentre una solucion o llegue a una contradiccion. En este ultimo caso, se hace *backtraking* hasta el nodo incompleto mas cercano (con un unico arco descendiente), invierte el valor de la variable y vuelve a 2; si todos los nodos tienen sus dos arcos descendientes, se determina la satisfacibilidad de la formula.

En la tabla 1 se muestra la ejecucion del algoritmo para un ejemplo particular. Una de las columnas incluye la representacion gra ca del arbol de busqueda.

**Paso 0:** la instancia consiste de 5 clausulas que involucran 4 variables, a las cuales se les puede asignar el valor verdadero (T = True) o falso (F = False). Dado que es el paso inicial, el arbol de busqueda es vac·o.

**Paso 1:** en este paso DPLL selecciona aleatoriamente una variable, entre las clausulas mas cortas y le asigna un valor para que satisfaga la clausula a la cual pertenece, por ejemplo $w = T$. Entonces se agrega un nodo y un arco, los cuales representan la variable seleccionada ($w$) y su valor de verdad (en este caso $T$) respectivamente.

**Paso 2:** se extraen, a partir de la ultima eleccion, las implicaciones logicas: las clausulas que contienen la variable $w$ se satisfacen y son eliminadas, las clausulas que incluyen $\neg w$ son simpli cadas, y el resto de las clausulas se mantiene sin modi caciones. Si ninguna clausula unitaria (es decir con una sola variable) existe, entonces se debe seleccionar una nueva variable.

**Paso 3:** en este punto toma lugar la regla de division (*splitting*), por lo tanto otro nodo y otro arco se agregan al arbol.

**Paso 4:** como en el paso 2, pero en este punto existen clausulas unitarias, entonces las variables que ellas contienen deben ser jadas de manera adecuada (esto es lo que se denomina propagacion).

**Paso 5:** la propagacion de la clausula unitaria resulta en una contradiccion, por lo tanto, el camino corriente en el arbol de busqueda se marca con una $C$.

**Paso 6:** el algoritmo DPLL retrocede hasta la ultima variable en la que se produjo una division (*split*) ($x$), invierte su valor, es decir le da el valor $F$ y crea un nuevo arco.

**Paso 7:** se realiza lo mismo que en el paso 4.

**Paso 8:** La propagacion de la clausula unitaria elimina todas las clausulas, esto signi ca que se encontro una solucion $S$.

En este ejemplo se muestra como DPLL encuentra una solucion para una instancia satisfacible. Para una instancia no satisfacible, la no satisfacibilidad se prueba cuando al retroceder (paso 6) no es

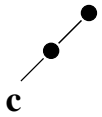
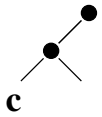
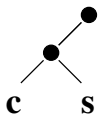
| Paso | Cláusulas | Árbol de búsqueda |
|---|---|---|
| **0** | $w \vee \neg x \vee y$<br>$\neg w \vee x \vee z$<br>$\neg w \vee \neg x \vee \neg y$<br>$\neg w \vee \neg x \vee y$<br>$x \vee y \vee \neg z$ | |
| **1** | split: $w = T$ |  |
| **2** | $x \vee z$<br>$\neg x \vee \neg y$<br>$\neg x \vee y$<br>$x \vee y \vee \neg z$ | |
| **3** | split: $x = T$ |  |
| **4** | $\neg y$ $y$ | |
| **5** | propagacion: $y = F$ $y = T$<br>contradiccion |  |
| **6** | volver al estado 3: $x = F$ |  |
| **7** | $z$<br>$y \vee \neg z$ | |
| **8** | propagacion: $z = T$ $y = T$<br>solucion: $w = T, x = F, y = T, z = T$ |  |

Tabla 1: Ejemplo de la resolucion de una instancia de 3-SAT utilizando el algoritmo DPLL.

posible ninguna accion, es decir, todas las variables tomaron los valores $T$ y $F$. En este caso todos los nodos en el arbol de busqueda nal tendr·an sus dos arcos descendientes y las hojas, rotuladas con $C$ (una contradiccion).

## 4. DPLL Y REDES NEURONALES (DPLL_RN)

La idea central de DPLL_RN es que a partir de un conjunto de formulas satisfacibles generadas aleatoriamente, se genera un conjunto de patrones que corresponden a las asignaciones optimas de valores a variables que satisfacen dichas formulas. Luego, se entrena una red neuronal con dichos patrones y nalmente se ejecuta el algoritmo DPRN para un conjunto de formulas satisfacibles y no satisfacibles generadas aleatoriamente. En la gura 1 se puede observar un esquema gra co con las distintas componentes del sistema.

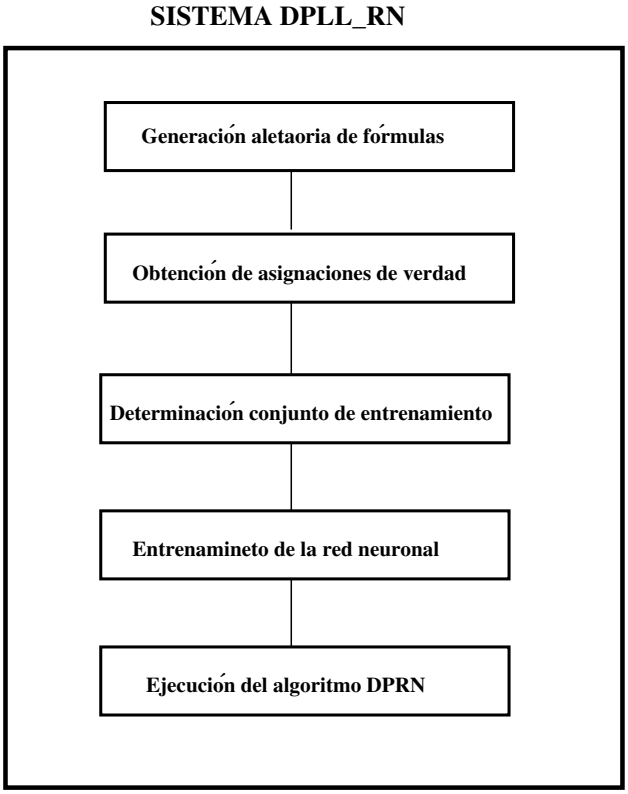



Figura 1: Componentes del sistema DPLL_RN

El algoritmo DPRN, que es una parte central del sistema, puede ser considerado un algoritmo h·brido ya que se basa en el algoritmo DPLL aumentado con el poder de las redes neuronales [8, 11]. DPRN tiene caracter·sticas propias que lo hacen interesante, dado que el mismo esta basado en el algoritmo DPLL pero haciendo uso de una red neuronal para que decida cual es la variable que debe ser seleccionada en cada caso y con que valor de verdad. De esta manera, DPRN permite determinar la satisfacibilidad y no satisfacibilidad de formulas que no se encuentran en el conjunto de entrenamiento debido al poder de generalizacion de las redes neuronales.

Las etapas involucradas en la aplicacion del sistema DPLL_RN son:

- Generacion aleatoria de formulas en forma normal conjuntiva (CNF): esta etapa consiste en la generacion aleatoria de las formulas utilizando el programa `mkcnf` [6].

- Obtencion de las asignaciones que hacen verdaderas las formulas: el conjunto de asignaciones de verdad que satisfacen las formulas se selecciona usando un algoritmo de busqueda. La seleccion esta basada en las reglas que usa el algoritmo DPLL, de manera que es un algoritmo exhaustivo que usa la regla del literal puro y la clausula unitaria. Este algoritmo permite obtener todas las asignaciones posibles de valores para todas las variables y en todos los ordenes posibles, y a partir de all· se seleccionan los caminos de longitud m·nima que satisfacen las formulas, esto es, los caminos de longitud m·nima en el arbol de busqueda generado por este algoritmo.

- Determinacion del conjunto de patrones de entrenamiento: el conjunto de entrenamiento se genera en base a las asignaciones de verdad obtenidas con el algoritmo mencionado en el item anterior, en donde por cada paso, se genera un conjunto de patrones analizando las decisiones que se deben tomar en cada punto.

- Entrenamiento de la red neuronal: una vez obtenidos los patrones de entrenamiento, se entrena con el simulador *Stuttgart Neural Networks Simulator* (SNNS) [10], una red neuronal de multiples capas con conexiones hacia adelante, utilizando el algoritmo *backpropagation* con momento, con los parametros estandar para este tipo de algoritmo. El numero de unidades de entrada y de salida es el doble del numero de variables.

- Ejecucion del algoritmo DPRN: en la ultima etapa se ejecuta el algoritmo DPRN, utilizando como entrada un conjunto de formulas satisfacibles y no satisfacibles no utilizado para el entrenamiento, dicho algoritmo determina la satisfacibilidad o no de las formulas dadas.

# 5. AMBIENTE DE EJECUCIÓN

Un ambiente de ejecucion se puede utilizar para describir el contexto en el cual se ejecutan ciertas tareas. Este trabajo en particular permite de nir el contexto en el cual es posible utilizar el sistema DPLL_RN, de una manera amigable y haciendo uso de los bene cios provistos por el lenguaje de programacion Java (e.g. posibilidades de desarrollo de aplicaciones web, etc.). El ambiente permite al usuario la posibilidad de elegir entre dos opciones para el ingreso de la formula, una de ellas es que el mismo genere una formula de forma aleatoria y la otra opcion es que el usuario ingrese la formula que el desee en forma manual. El ambiente entonces va a mostrar las asignaciones de verdad que hacen verdadera la formula, utilizando un arbol para tal n. La forma gra ca del mismo permite que el usuario conozca las asignaciones de verdad . En el caso que la formula no sea satisfacible el ambiente muestra un arbol vac·o.

## 5.1. Módulos que conforman el Ambiente

Obtencion del conjunto de redes neuronales

Aqu· se obtienen las redes neuronales necesarias para ejecutar el algoritmo DPRN del sistema DPLL_RN. Este conjunto de redes neuronales consta de una red para 3 variables, para ser utilizada cuando se requiere determinar la satisfacibilidad de una formula de 3 variables, una para 4 variables, y as· siguiendo, cabe destacar que se podr·an tener mas de una RN en cada caso particular, y seleccionar cualquiera de ellas. Es decir, es necesario contar con un pool de redes neuronales entrenadas segun lo descripto en la seccion 4, para ser utilizadas cuando el algoritmo DPRN lo requiera.

Sub-sistema DPRN (del sistema DPLL_RN)

El programa DPRN que forma parte del sistema DPLL_RN utiliza un archivo con la formula a analizar y una RN de las obtenidas segun se explica en el punto anterior. Cuando un usuario ingresa una formula para ser analizada, con el numero de variables de la misma determina cual red va a utilizar y abre el archivo de la red que corresponda, de aqu· se extrae toda la informacion de la red que necesita el algoritmo.

Generador de formulas aleatorias

El ambiente tiene una opcion de generar formulas de forma aleatoria y este informa sobre la satisfacibidad o no de la misma. Por tal motivo se necesita de un modulo que realice la tarea de generar una formula de forma aleatoria en CNF, que tambien es parte de este ambiente.

Manejo de las formulas

Una de las opciones del ambiente es que la formula sea generada de forma aleatoria. En este caso se solicita que se ingresen la cantidad de variables, la cantidad de clausulas que debe contener la formula y la semilla que debe ser utilizada por el generador aleatorio. La otra opcion es que el usuario ingrese la formula que el desee, en este caso se solicita la cantidad de variables, la cantidad de clausulas y la formula cuyas variables se denotan con letras mayusculas, comenzando desde la A. El formato de la formula puede contener los operadores logicos ($\vee$ y $\wedge$) y los parentesis que necesite. Sin embargo, estos ultimos son optativos dado que el sistema trabaja con formulas en forma normal conjuntiva.

Interface propiamente dicha

Cuando el ambiente se inicia muestra las dos opciones de ingreso de formulas. Una es el ingreso manual y la otra el aleatorio.

## 5.2. Un ejemplo utilizando el ambiente de ejecución

Cuando se inicia la ejecucion del ambiente el usuario va a ver una pantalla con las dos opciones para el ingreso de formulas. Una es el ingreso manual y la otra el aleatorio. A continuacion se muestra un ejemplo completo del funcionamiento del ambiente.

En la gura 2 se puede ver la primer pantalla que muestra el ambiente.

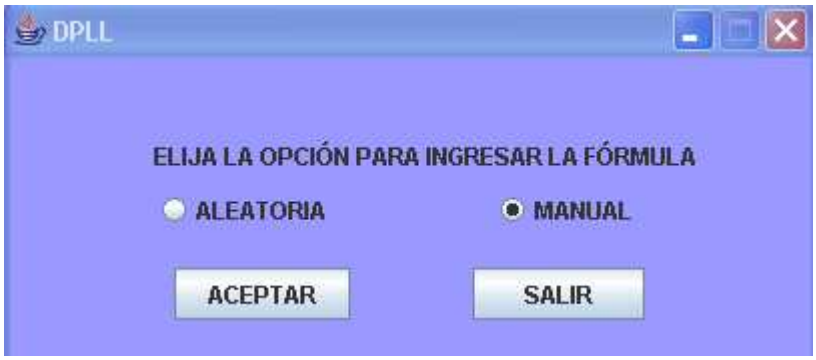



Figura 2: Eleccion de ingreso manual de una formula

Como se muestra en la pantalla luego de la eleccion de la opcion tiene la posibilidad de $salir$ del ambiente o $aceptar$, y continuar con el ingreso de la formula. Cuando se presiona el boton de $aceptar$ se muestra la pantalla de la gura 3.

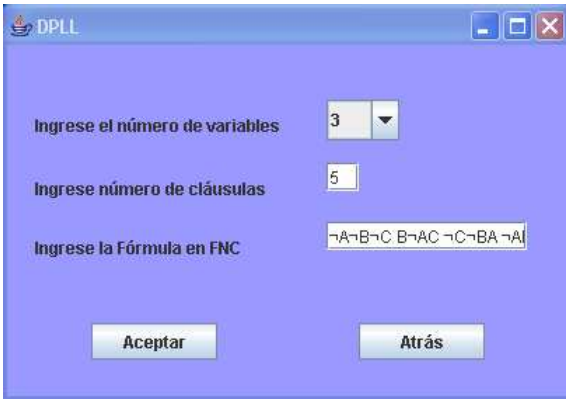



Figura 3: Ingreso de una formula

En esta pantalla se solicitan tres datos que son:

- Cantidad de variables de la formula (la formula a ingresar tiene 3 variables).
- Numero de clausulas (la formula a ingresar tiene 5 clausulas).
- La formula en CFN: aqu· se debe ingresar la formula para analizar la satisfacibilidad. Se pueden omitir los parentesis y los operadores logicos ($\vee$ y $\wedge$). En este caso se ingreso la siguiente formula omitiendo dichos operadores logicos y parentesis:

$$\neg A\neg B\neg CB\neg AC\neg C\neg BA\neg AB\neg C\neg BAC$$

que se corresponde a la siguiente formula :

$$(\neg A \vee \neg B \vee \neg C) \wedge (B\neg \vee A \vee C) \wedge (\neg C \vee \neg B \vee A) \wedge (\neg A \vee B \vee \neg C) \wedge (\neg B \vee A \vee C)$$

La pantalla de la gura 4 tiene como nalidad mostrar en forma gra ca, a traves de un arbol, las asignaciones de verdad que satisfacen la formula ingresada. Dicho arbol se interpreta de la siguiente manera: cada variable se muestra con un rectangulo con su nombre y este puede tener dos ramas, la rama de la derecha signi ca que a la variable se le ha asignado el valor $verdadero$ y la rama de la izquierda signi ca que tiene asignado el valor $falso$. De cada nodo siempre saldra una rama, porque signi ca que ese es el valor de verdad que debe tener asignado para poder satisfacer la formula (en el caso que la formula sea satisfacible).

El usuario puede extraer la asignaciones de verdad que hacen verdadera la formula visualizando el arbol como sigue. Comienza el recorrido del arbol desde la ra·z, cada nodo tiene el nombre de la variable y obtiene su asignacion con solo ver cual es la rama que sale del nodo. Luego se sigue la rama hasta otro nodo y se procede de la misma manera hasta que se haya examinado todos los nodos de dicho arbol. En el caso de nuestro ejemplo, con solo visualizar el arbol se puede obtener las siguientes asignaciones para las variables, $t(B) = F$ y $t(A) = F$, las cuales satisfacen la formula ingresada. Sobre la variable $C$ no aparece ninguna asignacion de verdad, lo signi ca que puede tener asignado cualquier valor de verdad.

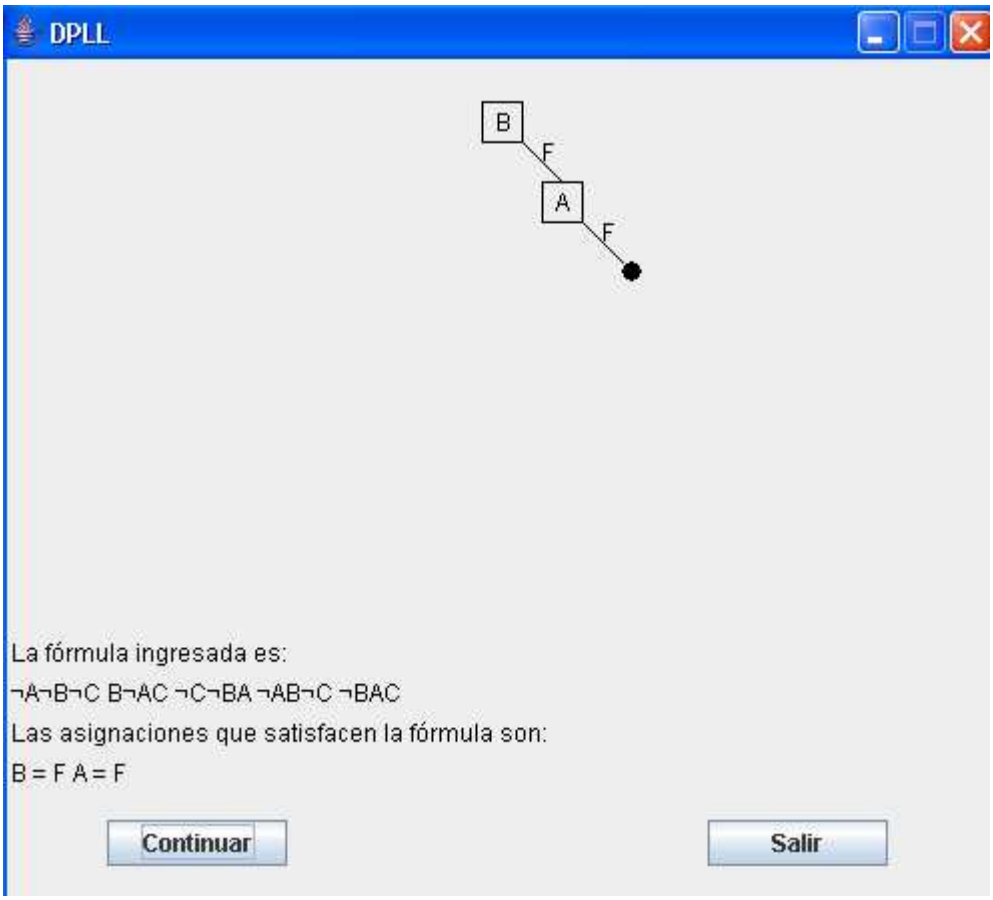



Figura 4: Resultado de una asignacion de verdad $t(A) = F$ y $t(B) = F$.

## 6. CONCLUSIONES

La idea central del presente trabajo fue la de desarrollar un ambiente de ejecucion que permita utilizar las distintas componentes del sistema DPLL_RN, sin necesidad de que el usuario tenga que conocer los detalles de la implementacion de dichas componentes, y que ademas, provea un entorno de manera que la intefase con el usuario sea amigable. Por otro lado, tambien se obtuvo un algoritmo h·brido basado en el algoritmo DPLL, el cual es uno de los mas usados para propositos practicos, combinado con las capacidades de aprender y generalizar de las redes neuronales. A partir de aqu· y de los resultados obtenidos [8, 11] se puede esperar que esta herramienta sea un punto de comienzo para lograr resultados mas robustos, es decir, trabajar con una cantidad mayor de variables, hecho fundamental en el crecimiento exponencial del problema SAT, y por ende en la complejidad de los algoritmos necesarios para resolver instancias de este problema. Finalmente, es importante destacar que en la bibliograf·a existente no se ha propuesto ningun algoritmo con las caracter·sticas que tiene el algoritmo DPRN [1, 7].

## REFERENCIAS

[1] Carlos Ansotegui and Felip Manya. An introduction to satis ability algorithms. *Revista Iberoamericana de Inteligencia Atificial*, (20), 2004.

[2] Stephen A. Cook and David G. Mitchell. Finding Hard Instances of the Satis ability Problem: A Survey, 1997.

[3] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Comunications of the ACM*, (5):394 397, 1962.

[4] Martin Davis, Ron Sigal, and Elaine Weyuker. *Computability, Complexity and Languages*. Academic Press, 2nd edition, 1994.

[5] M R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. 1979.

[6] Allen Van Gelder. Mkcnf formulas generator. *University of California, Santa Cruz*.

[7] J. Gu, P. W. Purdom, J. Franco, and B. W. Wah. Algorithms for satis ability (sat) problem: A survey. *DIMACS Volume Series on Discrete Mathematics and theoretical Computer Science*, (35):19 151.

[8] Carlos Kavka and Patricia Roggero. Solving 3SAT Problems using Neural Networks. *Artificial Intelligence an Applications*, pages 431 435, September 2001.

[9] Christos Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization - Algorithms and Complexity*. Dover Publications, 1998.

[10] Institute Parallel and Dsitributed High Performance Systems. Stuttgart neural network simulator user's manual. *Stuttgart University, Germany*.

[11] Patricia Roggero. *Aplicación de Técnicas de Inteligencia Computacional para Resolver Instancias del Problema de Satisfacibilidad*. PhD thesis, Departamento de Informatica - UNSL, Mayo 2005.

[12] Andrew Slater. *Investigations into satisfiability search*. PhD thesis, Computer Sciences Laboratory, Research School of Information Sciences and Engineering, ANU, 2004.

[13] W. M. Spears and K. A. De Jong. Davis Putnam Logemann and Loveland Algorithms. pages 118 121, 1990.