Optimización por Enjambre de Partículas para Satisfacción de Fórmulas Booleanas

Victor González Chamorro[1], Marcos Villagra[1], Benjamín Barán[1, 2]

[1] Universidad Católica de Asunción
Asunción, C.P. 1683, Paraguay
{victor_gonzalez, marcos_villagra}@uca.edu.py
[2] Universidad Nacional de Asunción
Asunción, C.P. 2169, Paraguay
bbaran@cba.com.py

Resumen. PSO es una metaheurística inspirada en el comportamiento social de enjambres, aplicada exitosamente para resolver problemas de optimización. Este trabajo propone una alternativa PSO para resolver el clásico problema de satisfacibilidad booleana conocido como SAT. Se implementa por primera vez un PSO con *Funciones de Aptitud Adaptativa*, técnica ya utilizada en Algoritmos Genéticos para escapar de óptimos locales. Para probar la efectividad de esta implementación se comparan tres Funciones de Aptitud Adaptativa, (1) Adaptación de Pesos, (2) Funciones de Refinamiento y (3) una mezcla de ambos. Para examinar experimentalmente esta propuesta, se realiza una comparación con un algoritmo PSO existente, llamado PSOIDP, un algoritmo de Colonia de Hormigas conocido como MaxMin-ACO y *WALKSAT* un popular algoritmo de búsqueda local. Los resultados experimentales demuestran que el PSO supera a PSOIDP y MaxMin-ACO, pero no puede vencer al WALKSAT, obteniendo resultados comparables solo en instancias no estructuradas.

Palabras Claves: Optimización por Enjambre de Partículas - PSO, problema de satisfacibilidad booleana - SAT, Funciones de Aptitud Adaptativa, WALKSAT, Elitismo, Optimización por Colonias de Hormigas.

1 Introducción

1.1 El problema SAT

El problema de satisfacibilidad booleana o SAT, fue el primer problema que se demostró NP-Completo [1]. En consecuencia, es uno de los problemas más importantes en las ciencias de la computación. SAT se define formalmente como un conjunto de variables booleanas $X = \{x_1, ..., x_n\}$ y una fórmula booleana o sentencia $\phi : \{0,1\}^n \rightarrow \{0,1\}$ donde n representa el número de variables. El objetivo es encontrar una asignación de variables $m = \langle x_1, ..., x_n \rangle \in \{0,1\}^n$ tal que $\phi(m) = 1$. En

este caso, a m se le llama *modelo* de ϕ. La fórmula se dice *satisfacible* si un modelo existe, de lo contrario se dice es *insatisfacible*. Un literal es una variable o su negación. Una cláusula es una disyunción de literales, i.e. literales unidos por un operador lógico $\vee$. La fórmula ϕ está en forma normal conjuntiva o CNF (del inglés, *Conjuctive Normal Form*) si $\phi(m) = c_1(m) \wedge ... \wedge c_p(m)$, donde cada c_i es una cláusula y p es el número de cláusulas. La familia de sentencias *k*-CNF tiene exactamente *k* literales por cláusula. SAT puede escribirse en notación CNF sin perder generalidad [2], y *k*-SAT contiene sentencias que están escritas en la forma *k*-CNF. Mientras 2-SAT puede ser resuelto en tiempo polinomial, *k*-SAT es *NP-COMPLETO* para $k \geq 3$ [3].

En instancias aleatorias (no estructuradas) de 3-SAT, se pueden observar rápidas transiciones en la solubilidad de instancias SAT. Este fenómeno es conocido como *transición de fase*. Las instancias difíciles se encuentran en la transición de fase, donde el cociente entre *p* y *n* es $p/n \approx 4.3$ [4]. Este valor se conoce como *punto de cruce* (del inglés, *crossover point*). Las instancias SAT estructuradas son obtenidas de aplicaciones prácticas y problemas reales [5], e.g., verificación de circuitos lógicos, planificación, síntesis de software, etc.

MAX-SAT es una variante de optimización de SAT. Dado un conjunto de cláusulas, MAX-SAT es el problema de encontrar una asignación *m* que maximice el número de cláusulas satisfechas. En MAX-SAT ponderado, cada cláusula c_i tiene un peso w_i asignado, mientras que en MAX-SAT no ponderado, $w_i = 1$, para toda i. La siguiente ecuación define la función objetivo de MAX-SAT:

$$f(m) = \sum_{i=1}^{p} w_i \cdot c_i(m) \tag{1}$$

donde $c_i(m) = 1$ si la cláusula c_i es satisfecha con la asignación *m*, y 0 de otra forma. Entonces, cuando se busca una solución para MAX-SAT, de hecho se está buscando modelos para SAT. Por eso, las soluciones se refieren a asignaciones y viceversa [6]. Usando la función objetivo de MAX-SAT podemos definir SAT como un problema de optimización donde se busca maximizar el número de cláusulas satisfechas.

1.2 Optimización por Enjambre de Partículas

En 1995 Kennedy y Eberhart [7] desarrollaron el primer algoritmo de enjambre de partículas para simular el vuelo sincrónico de las aves. Esta propuesta resultó ser un buen algoritmo de optimización para funciones matemáticas no-lineales continuas [8, 9]. Básicamente, PSO trabaja con un conjunto de soluciones candidatas denominado *enjambre*. Cada miembro del enjambre es una partícula, y esta posee un vector de solución denominado *posición.* Cada partícula conoce la mejor posición encontrada por el enjambre o *mejor global*. Si se definen subconjuntos de partículas del enjambre, cada uno de estos se denomina *vecindad.* Para este caso, cada partícula conoce la mejor posición de su propio vecindario o *mejor local.* Una vecindad define como las partículas interactúan entre ellas, y determina como la información se propaga en el enjambre afectando la convergencia del algoritmo [10]. A continuación se presenta el algoritmo 1 correspondiente al PSO tradicional.

```
Algoritmo 1: PSO (Particle Swarm Optimization)
  Inicializar partículas del enjambre
  mientras no se cumpla condición de finalización hacer
            Evaluar Partículas
            para toda partícula en el enjambre hacer
                Actualizar Posición
  fin-mientras
fin
```

En una iteración *t*, cada partícula P_i conoce su posición actual $X_i(t) = \{x_{i1},...,x_{in}\}$, la velocidad con la cual alcanzó su posición actual $V_i(t) = \{v_{i1},...,v_{in}\}$, y la mejor posición por ella encontrada hasta ese momento $b_i = \{b_{i1},...,b_{in}\}$. Todas las partículas del enjambre conocen la mejor posición global $g = \{g_1,...,g_n\}$. Si se definieron vecindades, entonces cada partícula conoce también la mejor posición local $l = \{l_1,...,l_n\}$. Así, a cada iteración *t*, las partículas se actualizan conforme a:

$$V_i(t+1) = \omega \cdot V_i(t) + ra_1 \cdot \theta_1(b_i - X_i(t)) + ra_2 \cdot \theta_2(g - X_i(t)) + ra_3 \cdot \theta_3(l - X_i(t)) \quad (2)$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \quad (3)$$

donde ω es conocido como *inercia* y se utiliza para controlar la influencia del valor anterior de la velocidad. Por su parte ra_1, ra_2 y ra_3 son números aleatorios uniformemente distribuidos en el intervalo [0,1], mientras que θ_1, θ_2 y θ_3 son parámetros conocidos como *constantes de aceleración* [10].

1.3 Algoritmos de Comparación.

Para comparar el desempeño de los algoritmos PSO propuestos en este trabajo, se utilizan tres algoritmos que resuelven SAT. (1) El *WALKSAT,* un reconocido algoritmo de búsqueda local para resolver SAT propuesto por Selman et. al. [11]. (2) El PSO *con partículas conducidas por instinto* o PSOIDP [12], un algoritmo para SAT que utiliza una función de instinto inspirada en algoritmos de Colonias Hormigas y un mecanismo de envejecimiento de cláusulas para mejorar las soluciones durante la búsqueda; y finalmente (3) MaxMin-SAT, un algoritmo de Colonias de Hormigas o ACO (del inglés *Ant Colony Optimization*) para resolver SAT que utiliza Funciones de Aptitud Adaptativa [6]. Cabe destacar que los algoritmos de inteligencia colectiva no pueden competir en general con los algoritmos del estado del arte de búsqueda local para SAT. Sin embargo algunos autores sugieren que los algoritmos evolutivos equipados con técnicas adicionales como funciones de aptitud adaptativa, operadores específicos del problema y optimización local pueden lograr buenos resultados [6, 13, 17, 18, 19]. En consecuencia, este trabajo propone implementar por primera vez un PSO con técnicas adicionales para resolver SAT, en busca de mejorar los resultados obtenidos por el PSO tradicional.

2 PSO para SAT

Para resolver SAT con PSO, cada partícula posee una asignación de valores de verdad que representa la posición $X_i(t)$ de la misma. La actualización de posición de una partícula se inspira en los trabajos de Secrest [14] y Lima y Barán [15]. La idea consiste en calcular la siguiente posición de una partícula en base a su posición actual X_i, la mejor posición global g y la mejor posición local l. Entonces, se utilizan los siguientes parámetros:

- A_1 , probabilidad de seleccionar información de la posición actual;
- A_2 , probabilidad de seleccionar información de la vecindad; y
- A_3 , probabilidad de seleccionar información de la mejor posición global.

donde $A_1+A_2+A_3 = 1$. En el algoritmo 2 se presenta la función de actualización:

```
Algoritmo 2: Función Actualizar Partículas()
desde j = 1 hasta n hacer
      ρ = número aleatorio en el intervalo [0,1]
      si ρ ≤ A3 entonces xij = gj
      sino - si ρ ≤ A3 + A2 entonces xij = lj
      fin-si
fin-desde
```

Si la variable de asignación aparece tanto en el mejor local como en el mejor global, tiene mayor probabilidad (A_2+A_3) de ser seleccionada. Si la posición actual tiene la misma asignación que la mejor posición global y la mejor posición local, no hay posibilidad de alterar ese valor. A raíz de esta dificultad, esta propuesta agrega un procedimiento de *ruido,* ejecutado en lugar del PSO con probabilidad $p_{ruido} < 0.1$[1]. Este procedimiento selecciona aleatoriamente una variable que pertenece a una cláusula insatisfecha para la asignación de g y la invierte. Este procedimiento puede destruir la mejor posición global por lo que también se propone una versión con *elitismo* [16] que hace una copia de g y la guarda como mejor solución.

2.1 Estructura de Vecindad Hipercúbica

La idea original fue tomada de Abdelbar y Abdelshahid [12]. Según dicho trabajo, esta topología mejora la performance del PSOIDP en comparación con el PSO tradicional. En consecuencia, este trabajo implementa una estructura hipercúbica de vecindad que provee un buen balance entre explotación y exploración de soluciones [12]. Sean P_i y P_j dos partículas, se dice que P_i es vecino de P_j si la representación binaria de sus posiciones $X_i(t)$ y $X_j(t)$ respectivamente, difieren exactamente en un bit. Para calcular un vecino de P_i se varía un bit de su vector de posición y luego se busca en el enjambre a dicha partícula. En caso de no encontrarse dicho vecino, se lo crea temporalmente para evaluar su *fitness*. En la figura 1 se muestra un ejemplo de vecindad hipercúbica.

[1] Este valor fue obtenido en pruebas experimentales realizadas en este trabajo.

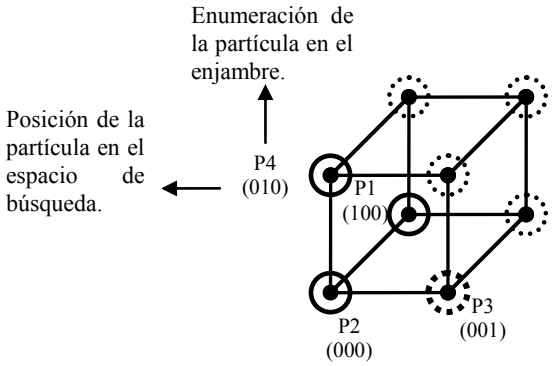

Figura 1. Vecindad hipercúbica. Como ejemplo se muestra a la partícula P2 con 3 vecinos: P1, P3 y P4. El punteado especial de P3 indica que no forma parte del enjambre. Las partículas con punteado fino no corresponden a la vecindad.

2.2 Funciones de Aptitud Adaptativa

En esta sección se presentan cada una de las funciones de aptitud adaptativas utilizadas en este trabajo. Estas son: (1) Adaptación de Pesos (SAW), (2) Función de Refinamiento (RF) y por último (3) una combinación de las dos anteriores (RFSAW).

2.2.1 Adaptación de Pesos Paso a Paso (*Stepwise Adaptation of Weights*)

Estudiada por Eiben y Van der Hauw [17], utiliza el principio de adaptación de pesos de las cláusulas difíciles de satisfacer. Los pesos w_i se van adaptando en la etapa de búsqueda de la solución, de manera a identificar las cláusulas difíciles de satisfacer. Los pesos se inicializan con w_i = 1. Después de una determinada cantidad de evaluaciones de la función de aptitud, estos son ajustados de acuerdo a:

$$w_i \leftarrow w_i + 1 - c_i(g) \tag{4}$$

donde g es la mejor solución global. En este trabajo, se actualizan los pesos cada 250 evaluaciones de la función de aptitud MAXSAT[2]. Este esquema de adaptación incrementa solo los pesos que corresponden a cláusulas insatisfechas de g .

2.2.2 Función de refinamiento (*Refining Function*)

Se basa en la observación de la existencia de muchas cadenas de bits con un mismo valor de evaluación para la función de aptitud. Esto hace imposible a un algoritmo distinguir una cadena de otra en SAT [18]. En consecuencia, se propone capturar conocimiento heurístico adicional con una función de refinamiento $r:\{0,1\}^n \rightarrow [0,1)$ asignando pesos a las variables, y utilizando esta función de refinamiento junto con la función de aptitud de MAXSAT, como se muestra a continuación:

[2] Este valor se extrajo del trabajo de Eiben y Van der Hauw [17].

$$f(m) = \sum_{i=1}^{p} w_i \cdot c_i(m) + \alpha \cdot r(m) \quad (5)$$

donde la influencia de $r(m)$ es controlada por $\alpha > 0$, conocido como *parámetro de influencia*. Utilizando $\alpha \in [0,1)$ se consigue la discriminación entre soluciones que satisfacen el mismo número de cláusulas [18]. A continuación se presenta en el algoritmo 3 la función de refinamiento:

```
Algoritmo 3: Función refinamiento(Xi)
Inicializar Variables
desde j=1 hasta n hacer
   SPK = SPK + K(xij)*wvj
   SVAP = SVAP + ABS(wvj)
fin-desde
retornar 0.5 * (1 + (SPK / (1 + SVAP)))
```

donde la función *K* se define como *K(1) = 1* y *K(0) = -1*, *ABS* retorna el valor absoluto de un número, *SVAP* es la sumatoria de valores absolutos de los pesos de las variables, *SPK* es la sumatoria de pesos de variables multiplicadas por la función *K* y $wv_j \in \mathbb{R}$ es el peso correspondiente a la variable x_{ij}. Los pesos con valores positivos altos indican que la variable correspondiente está favorecida a tener valor 1, mientras que los pesos con valores negativos indican que la variable tiende a ser 0. Inicialmente todos los pesos wv_j son 0 y estos son adaptados en cada iteración, luego de actualizar la posición de las partículas. La adaptación de los pesos, presentada en el algoritmo 4, apunta a escapar de un óptimo local.

```
Algoritmo 4: Función Adaptar Pesos()
Inicializar Variables
desde j=1 hasta n hacer
     wvj =  wvj- (K(gj)* CARD(Uj(g)))
fin-desde
```

donde g es el mejor individuo actual, la función $U_j(g)$ representa el conjunto de cláusulas insatisfechas que contienen a la variable g_j, y la función $CARD(U_j(g))$ denota la cardinalidad del conjunto $U_j(g)$. El esquema híbrido utiliza las funciones SAW y RF en conjunto, esperando acelerar el proceso de búsqueda y adaptación [19]. El algoritmo 5 muestra la adaptación de pesos para el esquema híbrido.

```
Algoritmo 5: Función Adaptar Pesos()
Inicializar Variables
desde j=1 hasta n hacer
     wvj =  wvj - (K(gj)*(SPCI(g ,j)))
fin-desde
```

donde la función *SPCI* calcula la sumatoria de los pesos de las cláusulas insatisfechas que contienen a la variable $g_j \in g$. Los pesos de las cláusulas son adaptados de acuerdo a SAW.

3 Resultados Experimentales

En esta sección se presenta el ambiente experimental y los resultados obtenidos con las variantes del PSO propuestos y los otros algoritmos escogidos para la comparación: (1) WALKSAT, (2) ACO con Función de Refinamiento y (3) PSOIDP.

3.1 Ambiente Experimental

Se realizaron pruebas experimentales sobre diversas instancias de SAT. Entre estas se consideran 8 instancias estructuradas extraídas de SATLIB[3], importantes por su aplicación práctica, y 8 instancias en transición de fase generadas aleatoriamente con WalkSat_V46[4], importantes por el fenómeno arriba citado. Se impuso un límite de 300 segundos al tiempo de corrida de cada algoritmo, realizándose 10 corridas para cada instancia por cada algoritmo. Los algoritmos PSO y los de comparación fueron implementados en lenguaje JAVA y compilados con JAVAC v1.6. Las corridas fueron realizadas en un computador personal con procesador AMD 2800+ de 2.08 GHz con 1 GB de RAM y sistema operativo Windows XP. La tabla 1 presenta la nomenclatura de los algoritmos implementados para este trabajo en orden alfabético. Se agrega el símbolo "+" cuando se utiliza la función de *ruido* y la letra "E" cuando se implementa *elitismo*.

Tabla 1. Tabla de nomenclatura de algoritmos.

Algoritmo	*Descripción*
ACORF	ACO con Función de Refinamiento (RF).
PSOIDP	PSO con partículas conducidas por instinto.
P-PSO	PSO tradicional.
PSORF	PSO con Función RF.
PSORFSAW	PSO con Función RF y SAW.
PSOSAW	PSO con Función SAW.
WALKSAT	Algoritmo de búsqueda local WALKSAT.

Los parámetros impuestos al WALKSAT son: un máximo de 10^3 inversiones (*flips)* por iteración y probabilidad de ruido $p_{walk} = 0.5$. Por su parte, los parámetros del PSOIDP fueron los utilizados en [12]. A su vez, los parámetros utilizados para el ACORF fueron extraídos de [6]. Por último, los parámetros para los PSO utilizados en este trabajo fueron afinados experimentalmente con un conjunto de instancias aleatorias distintas a las utilizadas en las pruebas reportadas. Estos parámetros son: enjambre con 50 partículas, $(A_1, A_2, A_3) = (0.375, 0.5, 0.125)$ y $p_{ruido} = 0.1$. En la afinación de α se obtienen los valores de 0.75 y 1.5 para PSORF y PSORFSAW respectivamente. Se recuerda que se utilizó la vecindad hipercúbica de la figura 1.

[3] Instancias Aim-100-1_6-no-1, Aim-100-2_0-yes1-1, Anomaly, Flat30-60, Hole6, Hole7, Par8-2-c, Par8-5-c, de la librería disponible en http://www.satlib.org

[4] Paquete para generar instancias SAT en formato CNF. Creado por Bart Selman. www.cs.cornell.edu/selman

3.2 Resultados Experimentales

Cuatro criterios son utilizados para evaluar cada uno de los algoritmos. El primero es el promedio de la tasa de éxitos (TE) que se calcula contando las corridas exitosas y dividiendo por el total de corridas. El segundo criterio es el tiempo promedio de las corridas exitosas (TPCE). El tercer criterio es el promedio de clausulas satisfechas (PCS) que se calcula contando la cantidad de cláusulas resueltas y dividiendo por el total de corridas. El último criterio considerado es la desviación estándar del PCS.

En la tabla 2 se puede observar los 15 algoritmos utilizados para resolver 16 instancias del SAT (ocho instancias estructuradas y ocho instancias aleatorias). Considerando que cada instancia fue resuelta 10 veces por cada algoritmo, queda claro que se reportan resultados de 15*16*10 = 2400 corridas. Debido a la gran cantidad de datos obtenidos en estas pruebas, se decidió presentar solo algunos rankings de algoritmos de forma a ilustrar las principales conclusiones del trabajo. Se hacen rankings separados para instancias estructuradas y aleatorias.

La tabla 2 presenta rankings para instancias estructuradas. Se nota la clara superioridad del WALKSAT pues obtiene el primer puesto en ambos rankings. El PSORFSAW+E fue el mejor de los algoritmos PSO propuestos en este trabajo al considerar el promedio de PCS. Sin embargo, al considerar la sumatoria de TE, PSOSAW+E obtiene un mejor resultado.

Tabla 2. Rankings de Promedio PCS y Sumatoria de TE para instancias estructuradas.

Ranking	*Algoritmo*	*Prom. PCS*	*Ranking*	*Algoritmo*	*Sum. TE*
1	WALKSAT	227.750	1	WALKSAT	4
2	PSORFSAW+E	227.325	2	PSOSAW+E	3.4
3	PSORFSAW+	227.300		PSORFSAW+E	3.4
4	PSOSAW	227.275	4	PSOSAW	3.2
5	PSOSAW+E	227.262		PSORFSAW+	3.2
6	PSOSAW+	227.250	6	PSORFSAW	3
	PSORFSAW	225.250	7	PSOSAW+	2.9
8	P-PSO+E	225.887	8	P-PSO+E	1
9	P-PSO+	225.775	9	P-PSO+	0.5
10	PSORF+E	225.612		PSORF+E	0.5
11	PSORF+	225.500	11	PSORF	0.4
12	ACORF	225.475		ACORF	0.4
13	PSORF	225.150	13	P-PSO	0.3
14	P-PSO	224.875	14	PSORF+	0.2
15	PSOIDP	216.512	15	PSOIDP	0

La tabla 3 muestra los resultados al considerar las instancias no estructuradas. Nuevamente el WALKSAT obtiene el primer puesto, confirmando su excelente desempeño, ya conocido en la literatura [11]. Sin embargo, al considerar el ranking de TE puede observarse que el PSOSAW es tan bueno como el WALKSAT aunque no tenga un buen desempeño al considerar el promedio de PCS.

Es interesante observar que en ambos rankings la función RF optimiza la cantidad de cláusulas resueltas y la función SAW mejora la tasa de éxitos del PSO tradicional.

Tabla 3. Rankings de Promedio PCS y Sumatoria de TE para instancias aleatorias.

Ranking	*Algoritmo*	*Prom. PCS*	*Ranking*	*Algoritmo*	*Sum. TE*
1	WALKSAT	217.350	1	WALKSAT	2.8
2	PSORF+E	216.487		PSOSAW	2.8
3	PSORF+	216.462	3	PSOSAW+E	2.6
4	P-PSO+E	216.412	4	PSORFSAW+	2.5
5	P-PSO+	216.250	5	PSORFSAW	2.4
6	PSOSAW+E	216.225	6	PSORFSAW+	2.1
7	PSORFSAW+	216.162	7	PSOSAW+	1.9
	PSORF	216.162	8	PSORF+	0.5
9	PSORFSAW+	216.150	9	P-PSO+	0.3
10	P-PSO	215.950	10	P-PSO+E	0.2
11	PSOSAW+	215.925	11	ACORF	0.1
	PSOSAW	215.925	12	PSORF+E	0.1
13	PSORFSAW	215.900	13	PSOIDP	0
14	ACORF	215.662		P-PSO	0
15	PSOIPD	211.437		PSORF	0

4 Conclusiones

Considerando que hasta la fecha no se ha publicado ningún algoritmo PSO exitoso para resolver SAT, se destaca que este trabajo presenta un novedoso enfoque que implementa un PSO con Funciones de Aptitud Adaptativa que demostró ser competitivo en instancias no estructuradas, sin lograr superar al reconocido algoritmo de búsqueda local WALKSAT, pero superando ampliamente al PSOIDP considerado como uno de los mejores algoritmos PSO para la resolución de problemas SAT [12]. Al mismo tiempo, cabe destacar que los algoritmos PSO propuestos en este trabajo superan también a otra técnica evolutiva basada en población, el MaxMin-SAT [6], mostrando su potencial para resolver problemas SAT.

Cabe destacar que en la mayoría de los casos, el uso de ruido y de elitismo ha resultado benéfico para el desempeño de los algoritmos implementados, conforme se aprecia en las tablas 2 y 3. Estos alentadores resultados indican que sería interesante aplicar otras técnicas que permitan mejoras aún mayores en el PSO como probar nuevas topologías de vecindad y otras técnicas de ruido.

Referencias

1. Cook, S.: The complexity of theorem proving procedures. In: Proceedings of the 3rd. ACM Symposium on Theory of Computing, Shaker Heights, Ohio, pp. 151–156 (1971)
2. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM 7(3), pp. 201–215 (1960)

2. Tseitin, G.: On the Complexity of Derivation in Propositional Calculus. Studies in Constructive Mathematics and Mathematical Logic 2, pp. 115–125 (1968)
3. Garey, M., Johnson, D.: Computers and Intractability: a Guide to the Theory of *NP*-completeness. W.H. Freeman & Company, San Francisco, California (1979)
4. Crawford, J., Auton, L.: Experimental Results on the Crossover Point in Satisfiability Problems. In: Proceedings of the 11th National Conference on Artificial Intelligence (1993)
5. Gu, J. et. al. Algorithms for the Satisfability Problem: A Survey. DIMACS Series 35, pp. 19-151 (1997)
6. Villagra, M., Barán, B.: Ant Colony Optimization with Adaptive Fitness Function for Satisfiability Testing. Proceedings of WoLLIC 2007, LNCS 4576, pp. 351–360 (2007)
7. Kennedy, J., Eberhart, R.: Particle Swarm Optimization. Proceedings of the 1995 IEEE international conference on neural networks (ICNN'95), Vol. 4, pp. 1942-1948 (1995)
8. Eberhart, R., Shi, Y.: Particle Swarm Optimization: Developments, Applications and Resources. Proceedings of the 2001 Congress on Evolutionary Computation, Vol. 1, pp. 81-86 (2001)
9. Engelbrecht, A.: Computational Intelligence: an Introduction. John Wiley & Sons, Ltd. (2002)
10. Ling, I.: Particle Swarm Optimization for Solving Constraint Satisfaction Problems. Thesis (M.Sc.), School of Interactive Arts and Technology, Simon Fraser University (2005)
11. Selman, B., Kautz, H., Cohen, B.: Noise strategies for improving local search. In: Proceedings of the AAAI'94. Vol. 1, pp. 337–343 (1994)
12. Abdelbar, A., Abdelshahid, S.: Swarm Optimization with Instinct-Driven Particles. Proceedings of IEEE Congress on Evolutionary Computation 2003 (CEC 2003), pp. 777-782 (2003)
13. P. Pinto, T. Runkler, J. Souza.: An Ant Algorithm for Static and Dynamic Max-Sat Problems. Bio-Inspired Models of Network, Information and Computing Systems, Vol. 1, pp. 1 – 8 (2006)
14. Secrest, B.: Traveling Salesman Problem for Surveillance Mission Using Particle Swarm Optimization. MS Thesis, AFIT/GCE/ENG/01M-03, School of Engineering, Air Force Institute of Technology, Wright-Patterson (2001)
15. Lima, J., Barán, B.: Aplicación de Optimización de Enjambre de Partículas al Problema del Cajero Viajante Bi-objetivo. Inteligencia Artificial: Revista Iberoamericana de Inteligencia Artificial, Vol. 32, pp. 67-76 (2006)
16. Nedjah, N., Abraham, A., Macedo Mourelle, L.: Genetic Systems Programming. Studies in Computational Intelligence, Vol. 13 (2006)
17. Eiben, A., Van de Hauw, J.: Solving 3-SAT with Adaptive Genetic Algorithms. Proceedings of the Fourth IEEE Conference on Evolutionary Computation, pp. 81–86 (1997)
18. Gottlieb, J., Voss, N.: Improving the Performance of Evolutionary Algorithms for the Satisfiability Problem by Refining Functions. Proceedings of 5th Parallel Problem Solving From Nature (PPSN V), LNCS 1498, pp. 755–764 (1998)
19. Gottlieb, J., Marchiori, E., Rossi, C.: Evolutionary Algorithms for the Satisfiability Problem. Evolutionary Computation 10, pp. 35-50 (2002)