¿Qué es la Optimización Combinatoria y qué tipos de problemas resuelve?

David Martínez Galicia (DavidGalicia@outlook.es)
Doctorado en Inteligencia Artificial
Centro de Investigación en Inteligencia Artificial
Universidad Veracruzana

Resumen: Este trabajo tiene como objetivo brindar un panorama general del área de Optimización Combinatoria y sus aplicaciones. Específicamente se abordan los fundamentos de este campo de estudio, los problemas comúnmente investigados y los algoritmos heurísticos ocupados para su resolución. Dentro del análisis de los problemas se realiza una comparación de múltiples autores con el fin de resaltar las principales diferencias entre sus propuestas. Finalmente, se puede observar que existen diversos enfoques para resolver un problema. Sin embargo, es necesario analizar qué tipos de enfoques favorecen la búsqueda de mejores soluciones tomando en cuenta su costo computacional.

Palabras clave: Optimización combinatoria, modelado de problemas, algoritmos heurísticos.

---

1. ¿Qué es la Optimización Combinatoria?

La Optimización Combinatoria es un campo de estudio que busca la mejor solución dentro de un conjunto definido de soluciones para un problema [1]. Actualmente con la creciente capacidad de cómputo de los ordenadores, el desarrollo de programas que ayudan a solucionar problemas complicados ha tomado un papel importante en la Optimización Combinatoria. Sin embargo, aún siguen existiendo retos que enfrentar, por ejemplo, entre más elementos sean considerados para solucionar un problema, el número de posibles soluciones crece de forma impresionante. Este hecho hace que analizar todas las soluciones a un problema para encontrar cuál es la mejor no sea un proceso eficiente ni viable.

Este documento busca introducir algunos de los términos básicos, los problemas comunes y las herramientas más estudiadas en esta área. El reporte se organiza en 5 secciones: la sección 2 define qué es un problema de optimización, así como los elementos que lo componen; la sección 3 expone las principales técnicas (algoritmos) ocupadas en la Optimización Combinatoria; la sección 4 identifica tanto los problemas más estudiados como algunas propuestas para solucionarlos, y finalmente, la sección 5 resume la información presentada en este documento y ofrece algunas conclusiones.

2. ¿Qué es un problema de optimización?

Muchos de los problemas combinatorios, pueden ser vistos como problemas de optimización. Esto se debe a que se busca asignar valores a cada una de las variables del problema estudiado, con el fin de encontrar una solución óptima [2]. La definición de

un problema de optimización es puramente matemática, y aunque suene difícil, una vez comprendidos los elementos que lo conforman, el modelo matemático puede ser visto como una simplificación del problema en el mundo real que facilita su solución.

De forma general un problema de optimización se encuentra compuesto por:

a. Una descripción general de los parámetros necesarios para resolverlo.
b. Un objetivo que enuncia matemáticamente la tarea a realizar.
c. Un espacio de soluciones compuesto por todas las posibles soluciones.
d. Una función de evaluación que determina a las soluciones válidas [2].

3. ¿Qué tipo de soluciones existen a problemas de optimización?

Las soluciones a problemas de optimización generalmente son algoritmos, es decir, un programa computacional expresado como una serie de pasos para llevar a cabo una tarea específica. Sin embargo, se puede hacer una clasificación de algoritmos empleados en Optimización Combinatoria dependiendo de la complejidad del problema a resolver. La complejidad puede ser medida con respecto al tiempo (número de operaciones realizadas) o con respecto al espacio (cantidad de memoria usada).

Si hablamos específicamente del número de operaciones realizadas por un algoritmo, surge la clasificación de problemas P y NP. Donde la clase P define a problemas que se resuelven en un tiempo que es un polinomio de N, donde N es determinado por el tamaño de los datos de entrada. Y en contraparte, la clase NP que define a problemas que pueden ser verificados en tiempo polinomial, es decir, se puede comprobar si una solución satisface el problema, pero para que un problema clase NP sea solucionado se requiere de un algoritmo de orden superior [1].

Si un problema pertenece a un orden polinomial bajo o a un orden exponencial no tan elevado, el método más directo para encontrar la solución óptima es realizar una búsqueda exhaustiva, es decir, evaluar cada una de las posibles soluciones y determinar la mejor. Sin embargo, rara vez un problema de la vida real puede ser evaluado exhaustivamente. En consecuencia, existen algoritmos heurísticos que son diseñados para resolver problemas muy complejos en una forma más rápida que los métodos exhaustivos, sin embargo, sacrifica la precisión por la velocidad.

Los algoritmos heurísticos se pueden clasificar de la siguiente forma:

1. *Búsqueda local* es un tipo de algoritmo que construye o modifica una solución inicial para mejorarla, el proceso es repetido sobre las soluciones modificadas tratando de encontrar una solución más eficiente [3].
2. *Búsqueda global* es un tipo de algoritmo que explora todo el espacio de soluciones de un problema, generalmente, construyen un conjunto de soluciones que

optimizan en cada iteración, entre los cuales se destacan los algoritmos genéticos y la inteligencia colectiva [4].

3. *Algoritmos genéticos* son métodos de optimización que, entre otras aplicaciones, pueden emplearse para encontrar el valor o valores que consiguen optimizar una función [4].
4. *Inteligencia colectiva* son herramientas de optimización que basan su funcionamiento en el comportamiento de grupos de individuos que actúan colectivamente de un modo que parecen inteligente [5].

4. ¿Cuáles son algunos de los problemas estudiados por la Optimización Combinatoria?

Esta sección realiza un análisis de 5 problemas de optimización comúnmente resueltos con algoritmos heurísticos dentro del área de Optimización combinatoria. Cada problema será descrito tan con un enunciado como con su formulación matemática (objetivo, restricciones y función de evaluación). De la misma forma, para cada problema se incluyen 3 trabajos seleccionados de la literatura científica relacionada para comparar el enfoque propuesto de acuerdo, al tipo de algoritmo empleado, la representación ocupada y cómo se compone el vecindario de soluciones.

a. Problema de Satifacibilidad Booleana (SAT)

El problema SAT se ***define*** por la siguiente pregunta: ¿Es posible encontrar un conjunto de valores que dada una función booleana se obtenga una evaluación positiva? Para responder a este cuestionamiento es necesario identificar una función booleana. Una función booleana es una expresión compuesta por variables lógicas que se relacionan a través de operadores booleanos. Las variables lógicas generalmente tienen dos posibles resultados 0 y 1, que a su vez pueden interpretarse respectivamente como Falso y Verdadero. Supongamos que se tiene la siguiente función booleana:

$$F = (a \vee b) \wedge (\neg a \vee c \vee d) \wedge (c \vee \neg e) \wedge (f \vee \neg g \vee h) \quad (1)$$

Donde existen 8 variables identificadas por las letras (a, b, c, d, e, f, g, h) y tres operadores booleanos: la conjunción $(\wedge)$, la disyunción $(\vee)$ y la negación $(\neg)$. El ***objetivo*** de este problema es encontrar un conjunto de valores (0 y 1) que den como resultado 1. Específicamente la formulación del problema SAT como un problema de optimización recibe el nombre de ***MAX-SAT*** y fue estudiado por Steven Cook en 1971 [6]. Este modelado define que la función F de estar en un formato llamado Forma Normal Conjuntiva (CNF por sus siglas en inglés). La fórmula 1 es un ejemplo de una fórmula expresada en su forma normal conjuntiva, donde conjuntos de variables se agrupan entre paréntesis con disyunciones (cláusulas), y a su vez, estas cláusulas se unen por conjunciones.

La ***formulación*** del problema MAX-SAT es la siguiente: Dada una función F, se define a $f(F, a)$ como el número de literales no satisfechas por la asignación de valores a [2]. El problema de Máxima Satifacibilidad Booleana busca encontrar una asignación de valores que minimice el número de cláusulas no satisfechas a través de la siguiente ***función de evaluación***:

$$a^* \in \min_{a \in \ asignación(F)} f(F, a) \tag{2}$$

Entre los algoritmos más comunes se encuentran: ascenso por la colina [7], búsqueda tabú [8], y búsqueda con Backtracking [9], entre otros. La Tabla 1 hace una comparación de tres propuestas de algoritmos para solucionar el problema SAT en términos de la representación, construcción de la solución, y definición del vecindario.

Autor(es)	Yagiura y Ibaraki [7]	Selman et al [8]	Gu [9]
Algoritmo	Ascenso por la colina	Búsqueda Tabú	Búsqueda con Backtracking
Tipo de algoritmo	Búsqueda local	Búsqueda local	Búsqueda local
Representación de la solución	Vectores de dígitos binarios	Vectores de dígitos binarios	Vectores de dígitos binarios
Construcción de la solución	Solución completa	Solución completa	Solución parcial
Definición del vecindario	Intercambio de dos dígitos binarios	Intercambio de dos dígitos binarios	Cambio de un digito binario a la vez

Tabla 1. Cuadro comparativo de algoritmos para el problema SAT.

b. Problema de coloración de grafos (GCP)

El problema GCP se ***define*** por la siguiente pregunta: ¿Es posible encontrar el número mínimo de colores usados para identificar a los nodos de un grafo, de forma que, dos nodos conectados no tengan el mismo color? Para esto, es necesario introducir ciertos conceptos básicos procedentes de la teoría de grafos que nos ayudarán a entender mejor este problema.

Un grafo es un diagrama que se compone de nodos y líneas que conectan a estos nodos. Para que un conjunto de nodos y líneas sea considerado como un grafo, por lo menos este debe de contener dos nodos conectados por una línea. Los nodos también son conocidos como vértices y estos pueden representar un punto en un espacio, por ejemplo, una ciudad en una superficie plana o, un recorrido en un espacio tridimensional. A menudo los nodos son representados a través de las letras de un alfabeto [10]. En cambio, las líneas, también conocidas como aristas, es una conexión entre dos nodos, a menudo las aristas pueden tener un peso asociado, sin embargo, para GCP es irrelevante.

Comúnmente, el problema de coloración de grafos puede ser replanteado en términos de extensiones de territorio y mapas, donde su **objetivo** es, para el caso de un país (Figura 1), encontrar el número de colores mínimos y necesarios para que dos estados colindantes no tengan el mismo color [10].

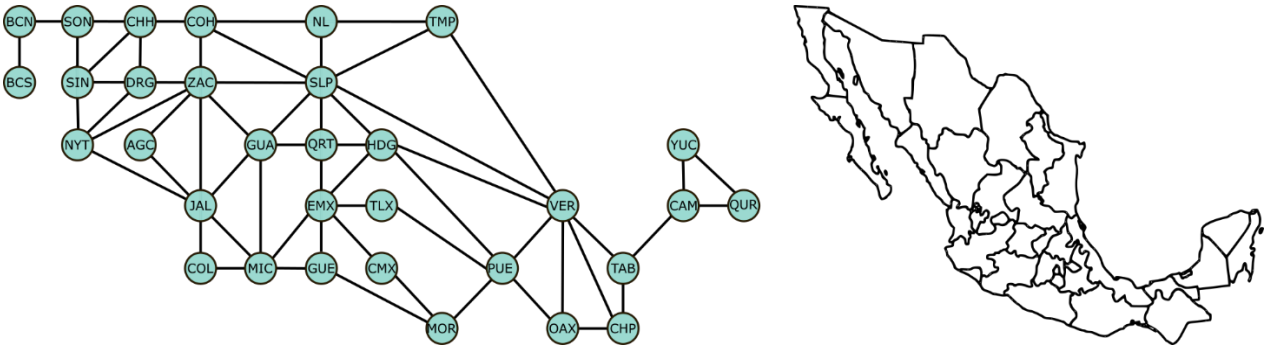

Figura 1. Representación de los estados colindantes de la República Mexicana con un grafo.

Específicamente la ***formulación*** del problema GCP como un problema de optimización recibe el nombre ***COL*** [2]. Este trabaja dado un grafo $G = (V, E)$, un conjunto de nodos V, y un conjunto de aristas E. El problema COL busca particionar una solución $V^* \subset V$ con $\min(K)$ conjuntos mutuamente exclusivos y no vacíos representados por $(V_1, \dots, V_k)$ con $V_1 \cup \dots \cup V_k = V$ y $V_i \cap V_j = \emptyset$ para todo $i, j \in \{1, \dots, k\}$ y $i \neq j$.

Entre las técnicas más comunes para su solución se encuentran: optimización mediante cúmulos de partículas (PSO) [11], optimización mediante colonia de abejas (ABC) [12], algoritmos genéricos (GA) [13], entre otros. La Tabla 2 hace una comparación de tres propuestas de algoritmos para solucionar el problema COL en términos de la representación, construcción, evaluación de la solución, y del vecindario.

Autor(es)	Aoki et al [11]	Douiri y Elbernoussi [13]	Bouziri y Jouini [14]
Algoritmo	PSO	Algoritmo genético	Búsqueda tabú
Tipo de algoritmo	Búsqueda global Inteligencia Colectiva	Búsqueda Global Cómputo evolutivo	Búsqueda Local
Representación de la solución	Vectores de números	Vectores de números	Vectores de números
Función de evaluación	$E = 1 - \left(\frac{conflictos}{arcos}\right)$	$E = \sum_{arista} conflicto$	$E = C * \lvert conflicto \rvert + X$ $X = \sum_{i=1}^{k} \sum_{j=1}^{\lvert C_i \rvert} i \times j$
Construcción de la solución	Solución completa	Solución completa	Solución completa
Definición del vecindario	Vecinos alterando a partículas.	Vecinos por cruza y mutación	Cambio de color de un nodo conflictivo

Tabla 2. Cuadro comparativo de algoritmos para el problema COL.

c. Problema de embalaje en contenedores (BPP)

A diferencia de los problemas previos, el problema BPP plantea una situación más comúnmente relacionada a la industria de envíos y aduanas, donde se busca optimizar el número de recursos usados por medio de una organización de objetos. Uno de las **formulaciones** matemáticas para este problema fue descrita por Nicholas Hall [15].

Dado un conjunto de n objetos de un determinado peso y n contenedores con una capacidad máxima de 1 unidad, el ***objetivo*** de este problema es encontrar el mejor arreglo de objetos de modo que, se minimicen el número de contenedores usados:

$$\min f = \sum_{j=1}^{n} Y_j \tag{3}$$

En la notación matemática la variable i representa a los objetos, mientras que la variable j representa a los contenedores. Para indicar el uso de contenedores y la pertenencia de objetos se incluyen dos variables de decisión: Y_j que toma el valor de 1 si el contenedor j es ocupado durante la solución, y X_{ij} que toma el valor de 1 si el objeto i se encuentra en el contenedor j. Asimismo, el peso de los objetos se representa por t_i. Para acotar más el problema, la función f se encuentra sujeta a las siguientes condiciones (también conocidas como restricciones):

1. La capacidad de los contenedores no debe ser sobrepasada.

$$\sum_{i=1}^{n} t_i \, X_{ij} \leq 1 \qquad\qquad j = 1, \dots, n \tag{4}$$

2. Cada elemento debe ser asignado a un contenedor.

$$\sum_{j=1}^{n} X_{ij} = 1 \qquad\qquad i = 1, \dots, n \tag{5}$$

Entre los algoritmos más comunes para la solución del problema BPP se encuentran múltiples heurísticas de búsqueda local especializadas, pero también la literatura reporta que otras técnicas como optimización por colonia de hormigas [16], optimización mediante cumulo de partículas [16] y algoritmos genéticos [17], también han sido usadas. La Tabla 3 hace una comparación de tres propuestas de algoritmos heurísticos especializados para BPP en términos de la representación, construcción, evaluación de la solución, y del vecindario.

Autor(es)	Hall et al [15]	Hall et al [15]	Hall et al [15]
Algoritmo	Next Fit (NF)	First Fit (FF)	Best Fit (BF)
Tipo de algoritmo	Búsqueda Local	Búsqueda Local	Búsqueda Local
Representación de la solución	Vectores y matrices de dígitos binarios	Vectores y matrices de dígitos binarios	Vectores y matrices de dígitos binarios
Función de evaluación	$E = \sum_{i=1}^{n}\sum_{j=1}^{n} t_i\, X_{ij} \leq 1$	$E = \sum_{i=1}^{n}\sum_{j=1}^{n} t_i\, X_{ij} \leq 1$	$E = \sum_{i=1}^{n}\sum_{j=1}^{n} t_i\, X_{ij} \leq 1$
Construcción de la solución	Solución parcial	Solución parcial	Solución parcial
Definición del vecindario	Si el objeto i entra en el contenedor actual se guarda ahí, sino se abre uno nuevo sin revisar si otros contenedores previos tienen espacio disponible.	El objeto i se guarda en el contenedor más cercano con suficiente espacio para el objeto.	El objeto i se guarda en el contenedor ocupado con menor capacidad disponible, pero con suficiente espacio para el objeto.

Tabla 3. Cuadro comparativo de algoritmos para el problema BPP.

d. Problema de planeación en máquina paralelas (PMSP)

Al igual que el problema BPP, el problema PMSP se relaciona directamente con aplicaciones industriales. La ***formulación*** del problema es la siguiente: Dado un conjunto de n trabajos $(J_1, \dots, J_n)$, donde cada una debe ser programada en una de las m máquinas disponibles $(M_1, \dots, M_m)$. Un trabajo puede ser ejecutado por una máquina a la vez y una máquina sólo puede procesar a lo mucho un trabajo a la vez. Si un trabajo j es procesado en una máquina i, tardará un tiempo de procesamiento positivo p_{ij}. Además, para cada trabajo se tiene un peso no negativo w_j. El ***objetivo*** de este problema es programar los trabajos, de forma que, la suma de los tiempos de terminación ponderados de los trabajos sea minimizados [18].

En este tipo de problemas, cuando $p_{ij} = p_j$ para todas las i y las j, las máquinas tienen la misma velocidad, es decir, los tiempos de procesamiento de un trabajo son idénticos en todas las máquinas. Cuando $p_{ij} = p_j/s_i$, donde p_j es el tiempo de procesamiento del trabajo j y s_i es la velocidad de la máquina i, entonces las máquinas son llamadas uniformes. Si los p_{ij} son arbitrarios entonces las maquinas son llamadas no relacionadas. Ambas máquinas uniformas y no relacionadas pertenecen a los problemas denominado como programación en máquinas paralelas no idénticas [18].

Blazewicz et al. [19] describe un ***modelado*** basado en la asignación de variables para un problema de máquinas paralelas uniformes con una unidad estándar de tiempo de procesamiento $(Q|p_j = 1|\sum C_j)$, donde C_j es el tiempo de terminada del trabajo j y $x_{ij}^k = 1$ si el trabajo j es el $k - ésimo$ procesado en la maquina i. La ***función objetivo*** está dada por:

$$f = \min \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{n} \frac{k}{s_i} x_{ij}^k \tag{6}$$

Bajo las restricciones de:

1. Toda máquina i debe tener asignado una lista de trabajos contiguos.

$$\sum_{k=1}^{n} x_{ij}^k \leq 1 \quad \forall i, k \tag{7}$$

2. El tiempo de terminación de una maquina i está dado por:

$$C_j^k = \sum_{l=1}^{k} \sum_{j=1}^{n} p_j x_{ij}^l \tag{8}$$

Entre los algoritmos más comunes para la solución del problema PMSP se encuentran múltiples heurísticas como búsqueda tabú [20], Ascenso por la colina [21] y algoritmos genéticos [22]. La Tabla 4 hace una comparación de tres propuestas.

Autor(es)	Terzi et al [20]	Kim et al [21]	Ak y Koc [22]
Algoritmo	Ascenso por la colina	Búsqueda tabú	Algoritmo genético
Tipo de algoritmo	Búsqueda Local	Búsqueda Local	Búsqueda Global Cómputo evolutivo
Representación de la solución	Matriz	Lista de listas	Vectores de $m \cdot j$ elementos
Función de evaluación	$E = \max_i \sum_{l=1}^{k} \sum_{j=1}^{n} p_j x_{ij}^l$	$E = \max_i \sum_{l=1}^{k} \sum_{j=1}^{n} p_j x_{ij}^l$	$E = \max_i \sum_{l=1}^{k} \sum_{j=1}^{n} p_j x_{ij}^l$
Construcción de la solución	Solución completa	Solución completa	Solución completa
Definición del vecindario	Intercambios e inserciones	Intercambio de dos trabajos.	Cruza a través de BPX [23].

Tabla 4. Cuadro comparativo de algoritmos para el problema PMSP.

e. Problema de cobertura de conjuntos (SCP)

El problema SCP es uno de los problemas combinatorios más estudiados debido al gran número de aplicaciones en mundo real. El SCP puede ser ***formalizado*** de la siguiente forma:

Dado un conjunto finito $A = \{a_1, \dots, a_m\}$ y una familia $F = \{A_1, \dots, A_n\}$ de subconjuntos $A_i \subseteq A$ que cubren a A, es decir, cada elemento de A aparece por lo menos en un subconjunto de F, el SCP tiene como ***objetivo*** encontrar el subconjunto con el tamaño más pequeño posible $C \subseteq F$ que cubre A [2].

Frecuentemente el SCP puede ser modelado como un problema de optimización, agregando al modelo variables de decisión. x_i es una variable que se asocia a cada subconjunto A_i para indicar si fue seleccionado como candidato para cubrir C. Cada subconjunto $A_i \;\; i = 1, \dots, n$, es representado por una matriz $m \times n$ llamada B, con $b_{ij} = 1$ si y solo si $a_j \in A$ y $b_{ij} = 0$ de forma contraria. Cada una de las filas de B corresponden a un elemento de A, donde una columna i cubre una fila j si y solo si el elemento a_i es contenido en el subconjunto A_i, es decir, $b_{ij} = 1$. Además, si $w(A_i)$ es el peso de A_i, se define $c_i = w(A_i)$ como el costo asociada de incluir la columna i en la solución [2]. Cuando el problema no cuenta con pesos, los pesos son asignados como 1.

La ***función objetivo*** del problema de optimización CSP está definida por:

$$\min f = \sum_{i=1}^{n} c_i \cdot x_i \tag{9}$$

Sujeta a la siguiente restricción:

1. Cada elemento de A debe ser cubierto por al menos un subconjunto de F.

$$\sum_{i=1}^{n} b_{ji} \cdot x_i \geq 1 \qquad j = 1, \dots, m \tag{10}$$

Entre los algoritmos más comunes para la solución del problema SCP se encuentran múltiples heurísticas como búsqueda tabú [24], optimización mediante cúmulos de partículas [25], algoritmos genéticos [26] y algoritmos meméticos [27]. La Tabla 5 hace una comparación de tres propuestas de algoritmos en términos de la representación, construcción, evaluación de la solución, y del vecindario.

Autor(es)	Caserta [24]	Balaji y Revathi [25]	Aickelin [26]
Algoritmo	Búsqueda tabú	PSO	Algoritmo genético
Tipo de algoritmo	Búsqueda Local	Búsqueda Global Inteligencia colectiva	Búsqueda Global Cómputo evolutivo
Representación de la solución	Vectores de dígitos binarios	Vectores de valores continuos	Permutaciones
Función de evaluación	$f = \sum_{i=1}^{n} c_i \cdot x_i$	$f = \sum_{i=1}^{n} c_i \cdot x_i$	$f = \sum_{i=1}^{n} c_i \cdot x_i$
Construcción de la solución	Solución completa	Solución completa	Solución completa
Definición del vecindario	Intercambios de un elemento (1 - 0)	Vecinos alterando a partículas.	Cruza a través de PUX [27].

Tabla 5. Cuadro comparativo de algoritmos para el problema PMSP.

5. Conclusiones

La Optimización Combinatoria es un campo de estudio que abarca un gran número de problemas con distintas aplicaciones en el mundo real. Sin embargo, existen tantos tipos algoritmos de algoritmos que muchas veces es difícil seleccionar un enfoque efectivo para la resolución del problema estudiado. Este reporte brinda un pequeño resumen acerca de las técnicas más empleadas y sus componentes. Es importante mencionar que la definición y el modelado de un problema muchas veces pueden dar indicios sobre qué tipos de representaciones, evaluaciones y algoritmos pueden ser más prometedores. Parte del trabajo futuro no solo incluye una revisión sistemática sobre qué tipos de algoritmos tienen un buen desempeño, sino también qué tipos de mecanismos benefician la búsqueda de soluciones óptimas de acuerdo a las características de un problema.

Referencias

1. Schrijver, A. (2003). Combinatorial Optimization - Polyhedra and Efficiency. Springer.
2. Hoos, H., & Sttzle, T. (2004). Stochastic Local Search: Foundations & Applications. Morgan Kaufmann Publishers Inc.
3. Martí, R. (2003). Procedimientos metaheurísticos en optimización combinatoria. Universitat de València.
4. Eiben, A., & Smith, J. (2015). Introduction to Evolutionary Computing. Springer Publishing Company, Incorporated.
5. Solé, R., Moses, M., & Forrest, S. (2019). Liquid brains, solid brains. Philosophical transactions of the Royal Society of London. Series B, Biological sciences, 374.
6. Cook, S. (1971). The Complexity of Theorem-Proving Procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing (pp. 151–158). Association for Computing Machinery.

7. Yagiura, M. & Ibaraki T. (1998). Efficient 2 and 3-Flip Neighborhood Search Algorithms for the MAX SAT: Experimental Evaluation. Journal of Heuristics, 7, 105–116.
8. Selman, B., Kautz, H., & Cohen, B. (1999). Noise Strategies for Improving Local Search-Proceedings of the National Conference on Artificial Intelligence, 1.
9. Jun Gu, Paul W. Purdom, John Franco, & Benjamin W. Wah (1996). Algorithms for the Satisfiability (SAT) Problem: A Survey. In DIMACS Series in Discrete Mathematics and Theoretical Computer Science (pp. 19–152). American Mathematical Society.
10. Malaguti, E., Monaci, M., & Toth, P. (2011). An Exact Approach for the Vertex Coloring ProblemDiscret. Optim., 8(2), 174–190.
11. Aoki T., Aranha C., & Kanoh H. (2015). PSO Algorithm with Transition Probability Based on Hamming Distance for Graph Coloring Problem. In 2015 IEEE International Conference on Systems, Man, and Cybernetics (pp. 1956-1961).
12. Fister, I. & Brest, J. (2012). A Hybrid Artificial Bee Colony Algorithm for Graph 3-Coloring. In Proceedings of the 2012 International Conference on Swarm and Evolutionary Computation (pp. 66–74). Springer-Verlag.
13. Sidi Mohamed Douiri, & Souad Elbernoussi (2015). Solving the graph coloring problem via hybrid genetic algorithms. Journal of King Saud University - Engineering Sciences, 27(1), 114 - 118.
14. Bouziri, H., & Jouini, M. (2010). A tabu search approach for the sum coloring problem. Electronic Notes in Discrete Mathematics, 36, 915-922.
15. Nicholas G. Hall, Soumen Ghosh, Roland D. Kankey, Sridhar Narasimhan, & Wan Soo T. Rhee (1988). Bin packing problems in one dimension: Heuristic solutions and confidence intervals. Computers & Operations Research, 15(2), 171 - 177.
16. Xu, C. (2011). A Review of the Application of Swarm Intelligence Algorithms to 2D Cutting and Packing Problem. In Advances in Swarm Intelligence (pp. 64–70). Springer Berlin Heidelberg.
17. Mohamed Amine Kaaouache, & Sadok Bouamama (2015). Solving bin Packing Problem with a Hybrid Genetic Algorithm for VM Placement. In Cloud Procedia Computer Science, 60, 1061 - 1069.
18. Kai Li, & Shan-lin Yang (2009). Non-identical parallel-machine scheduling research with minimizing total weighted completion times: Models, relaxations and algorithms. Applied Mathematical Modelling, 33(4), 2145 - 2158.
19. Blazewicz, J., Dror, M., & Weglarz, J. (1991). Mathematical programming formulations for machine scheduling: A survey. European Journal of Operational Research, 51(3), 283-300.
20. Terzi, M., Arbaoui, T., Yalaoui, F., & Benatchba, K. (2020). Solving the Unrelated Parallel Machine Scheduling Problem with Setups Using Late Acceptance Hill Climbing. Springer.
21. Kim, S.I., Choi, H.S., & Lee, D.H. (2006). Tabu Search Heuristics for Parallel Machine Scheduling with Sequence-Dependent Setup and Ready Times. In Computational Science and Its Applications - ICCSA 2006 (pp. 728–737). Springer Berlin Heidelberg.
22. Bilgesu Ak, & Erdem Koc (2012). A Guide for Genetic Algorithm Based on Parallel Machine Scheduling and Flexible Job-Shop Scheduling. Procedia - Social and Behavioral Sciences, 62, 817 - 823.
23. Kocamaz, M., çiçekli, U., & Soyuer, H. (2009). A developed encoding method for parallel machine scheduling with permutation genetic algorithm. In Proceedings of the European and Mediterranean Conference on Information Systems, EMCIS 2009

24. Caserta, M. (2007). Tabu Search-Based Metaheuristic Algorithm for Large-scale Set Covering Problems. Springer US.
25. Balaji, S., & Revathi, N. (2015). A new approach for solving set covering problem using jumping particle swarm optimization method. Natural Computing, 15.
26. Aickelin, U. (2008). An Indirect Genetic Algorithm for Set Covering Problems. Journal of the Operational Research Society, 53.
27. Aickelin, U., & Dowsland, K. (2008). An Indirect Genetic Algorithm for a Nurse Scheduling Problem. Computers & Operations Research, 31, 761-778.