applied sciences

MDPI

Article

An Approach to Aid Decision-Making by Solving Complex Optimization Problems Using SQL Queries

Jose Torres-Jimenez [1], Nelson Rangel-Valdez [2], Miguel De-la-Torre [3] and Himer Avila-George [3,*]

[1] CINVESTAV-Tamaulipas, Ciudad Victoria 87130, Tamaulipas, Mexico; jtj@cinvestav.mx
[2] División de Estudios de Posgrado e Investigación, Cátedras CONACyT—Tecnológico Nacional de México, Instituto Tecnológico de Ciudad Madero, Madero 89440, Tamaulipas, Mexico; nelson.rangel@itcm.edu.mx
[3] Departamento de Ciencias Computacionales e Ingenierías, Universidad de Guadalajara, Guadalajara 46600, Jalisco, Mexico; miguel.dgomora@academicos.udg.mx
* Correspondence: himer.avila@academicos.udg.mx

Abstract: In combinatorial optimization, the more complex a problem is, the more challenging it becomes, usually causing most research to focus on creating solvers for larger cases. However, real-life situations also contain small-sized instances that deserve a researcher's attention. For example, within a web development context, a developer might face small combinatorial optimization cases that fall in the following situations to solve them: (1) the development of an ad hoc specialized strategy is not justified; (2) the developer could lack the time, or skills, to create the solution; (3) the efficiency of naive brute force strategies might be compromised due to the programming paradigm use. Similar situations in this context, combined with a recent increasing interest in optimization information from databases, open a research area to develop easy-to-implement strategies that compete with those naive approaches and do not require specialized knowledge. Therefore, this work revises Structured Query Language (SQL) approaches and proposes new methods to tackle combinatorial optimization problems such as the Portfolio Selection Problem, Maximum Clique Problem, and Graph Coloring Problem. The performance of the resulting queries is compared against naive approaches; its potential to extend to other optimization problems is studied. The presented examples demonstrate the simplicity and versatility of using a SQL approach to solve small optimization problem instances.

Keywords: decision support system; combinatorial optimization; Three-Coloring Graph Problem; Portfolio Selection Problem; Maximum Clique Problem; SQL queries

Citation: Torres-Jimenez, J.; Rangel-Valdez, N.; De-la-Torre, M.; Avila-George, H. An Approach to Aid Decision-Making by Solving Complex Optimization Problems Using SQL Queries. *Appl. Sci.* **2022**, *12*, 4569. https://doi.org/10.3390/app12094569

Academic Editors: José Antonio Calvo-Manzano, Jezreel Mejía-Miranda and Giner Alor-Hernández

Received: 1 February 2022
Accepted: 27 April 2022
Published: 30 April 2022

1. Introduction

The Structured Query Language (SQL) is designed to store, manipulate, and query relational databases. SQL features include its ease of use and the expression of data retrieval requires using a high-level language. SQL is widely used in different sectors, and it is supported by the leading database management system (DBMS) vendors. A DBMS consists of interrelated data and programs to manipulate the stored data. Industry, government, health, and research are just a few examples of the sectors that use SQL to manage information. The data are compartmentalized into tables, records, and attributes, facilitating access and retrieval. The use of SQL allows processing large amounts of data, and this capability opens up the possibility of using SQL to solve combinatorial optimization problems, including clustering [1], informetric data processing [2], and many others [3–8].

Combinatorial optimization is a branch of applied mathematics related to operational research, algorithmic theory, and computational complexity theory. Besides, combinatorial optimization algorithms aim to solve problem instances with a large search space through the efficient search of the space of possible solutions. Strategies to solve such problems can face many shortcomings, mainly concerning the size of the search space. Problem instances of a large size represent a challenge for many approaches and since decades ago

have constituted the main focus in the related literature. However, some overestimated small-sized real-life problems may require additional attention, especially when they are related to NP-complete or NP-hard problems. In particular, there are interesting small cases that lie within pretty specific situations such as: (1) the sizes are small and can be easily managed; (2) the development of an ad hoc or specialized strategy is not justified; (3) the software developer lacks the time or skills to develop a solution; (4) implementing a naive brute force strategy that is computationally expensive. Indeed, the efficiency of the implementation is compromised by the programming style and the particularities of the programming language.

In the last decade, there has been an increasing interest in developing solutions for optimization problems whose data are stored in a DBMS. Brodsky et al. [9] focused on decision optimization through the development of a query language that supports business decisions. In [10,11], the authors develop SolveDB, a tool that integrates optimization problem solvers into SQL databases. Sakanashi and Sakai [12] proposes the definition of optimization problems through SQL and their transformation into constraints problems. More recently, the work in [13] used SQL to model machine learning problems; then, it mapped them into an equivalent graph to be used in the TensorFlow Python platform to solve them. Ergo, naturally, the next question emerges: Is the use of SQL convenient to solve small optimization problem instances? Therefore, this work addresses its research to answer such a question properly. This research intends to provide a simple approach whereby IT personnel, using SQL queries, can solve complex optimization problem instances to support the decision-making process. This work proposes the development and use of SQL statements for three classical optimization problems: three-coloring, the Portfolio Selection Problem (PSP), and the Maximum Clique Problem (MCP).

In order to assess the proposed approach, two experiments were designed: (a) the first experiment refers to the PSP and the MCP; (b) the second experiment validates the proposed approach with the problem of the community network and the channel assignment problem [14]. It is relevant to say that solutions to the PSP and MCP were previously reported, and their solutions are included here for completeness purposes. However, solutions for three-coloring problems and community self-referencing problems are presented for the first time.

The remainder of this article is organized as follows. Section 2 revises the use of SQL queries in the Portfolio Selection Problem (PSP) and the Maximum Clique Problem (MCP); it presents a novel SQL statement to solve the Three-Coloring Graph Problem (3CG). Section 3 describes the experimental design used to validate the use of SQL queries as optimizers and presents the main results. Section 4 analyzes the obtained results. Finally, Section 5 presents the main insights derived from the research.

2. Materials and Methods

This section revises the solution for the Portfolio Selection Problem (PSP) [15] and Maximum Clique Problem (MCP) [16] using a SQL approach. Furthermore, it proposes for the first time a novel SQL solution for solving instances of the 3-Coloring Graph Problem (3CG).

2.1. The Portfolio Selection Problem

An investment portfolio is a set of instruments listed on the stock market in which a person or company decides to invest. It is composed of fixed- and variable-income securities; the idea is to reduce the risk by combining different instruments such as stocks, bonds, real estate, and money market accounts.

The PSP involves the selection of an investment portfolio through which it is possible to achieve a given rate of return and minimize the risk at the same time. According to Markowitz [17], the process to select a portfolio consists of two stages: (1) the first stage begins with observation and experience and ends with expectations of the future behavior

of securities, and (2) the second stage starts with the expectations and ends with the selection of the portfolio.

The PSP is an optimization problem in which the goal is to maximize the predicted profit. As a result, the solution to the PSP is contingent on a collection of securities and an investment quantity $\mathcal{Q}$. The optimal size of the investment portfolio, the selection of securities in which it must be invested, and the quantity to be put in each security are all part of a solution to the problem.

Equation (1) formally defines the PSP. The problem uses a vector $e = \{e_1, e_2, \cdots, e_K\}$ to optimize the distribution of an investment quantity $\mathcal{Q}$. Each element $e_i \in e$ is associated with the ith security from the set $S = \{s_1, s_2, \cdots, s_K\}$ and takes a value j, which represents a quantity q_j from the set $\mathcal{I} = \{q_1, q_2, \cdots, q_N\}$ of investors' wealth portions. The matrix $\beta_{N \times K}$ contains cells $\beta_{i,j}$ with the expected return of the securities s_j given an investment q_i. Let us note that $N = |\mathcal{I}|$, $K = |\mathcal{S}|$.

$$\begin{aligned} &\textit{Maximize} \qquad f(e, \mathcal{I}, \beta, \mathcal{Q}) = \sum_{j=1}^{K} \beta_{e_j,j} \\ &\textit{subject to} \\ &\qquad \sum_{j=1}^{K} q_{e_j} = \mathcal{Q} \\ &\qquad e_j = i | q_i \in \mathcal{I} \end{aligned} \tag{1}$$

The β matrix specifies the expected return given an amount to invest and the specific security in which to invest. In the first column, the set $\mathcal{I}$ is shown, which represents the different choices of proportions, in ascending order, that the investor has chosen. The other columns represent the S set, one column for each additional security. In cell $\beta_{i,j} \in \beta$, the return of investing the percentage $q_i \in \mathcal{I}$ on security $s_j \in S$ is found. The index i of the proportion $q_i \in \mathcal{I}$ that must be invested in security $s_j \in S$ to maximize Equation (1) corresponds to the value for each element $e_j \in e$. The proportion q_1 of the first row of the matrix β must be 0 to allow for the possibility that a security does not belong to a portfolio. The set $\mathcal{I}$ is formed by dividing the quantity to be invested $\mathcal{Q}$ among the N different proportions. Equation (2) shows the procedure for obtaining q_i where $2 \leq i \leq N$.

$$q_i = (i-1) * \frac{\mathcal{Q}}{N-1} \tag{2}$$

In summary, an instance of the PSP is established with the tuple $(S, \mathcal{I}, \beta, \mathcal{Q})$, i.e., a set of securities S, a set $\mathcal{I} = \{q_1, q_2, \cdots, q_N\}$ of investors' wealth portions, a matrix $\beta_{N \times K}$ of expected securities' returns, and the amount of investment $\mathcal{Q}$. Table 1 shows an instance of the PSP, where $\mathcal{Q} = 200$, $N = 11$, $\mathcal{I} = \{0, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200\}$, $K = 5$, and $S = \{s_1, s_2, s_3, s_4, s_5\}$. The expected returns are established for each pair (q_i, s_j); a row filled with zeros was added so that the size of the portfolio is not restricted.

Torres-Jimenez et al. [15] developed an exact solution based on SQL queries to identify the investment portfolio e that maximizes the objective function shown in Equation (1) given an instance of the PSP as a matrix β and a quantity $\mathcal{Q}$. Figure 1 shows the generalization, in BNF notation, for any PSP instance. Given the β matrix as a database table with fields $\mathcal{I}$ and $s_i \in S$, the FROM clause contains the Cartesian product of as many β tables as the number of $K - 1$ securities of the instance that is going to be solved using a SQL query. The SELECT clause will select the field representing $\mathcal{I}$ from each table in the clause FROM. Besides, the SELECT clause includes an extra field that represents the summation of the returns of the fields in the previous filter. The WHERE clause will contain the total of the investments of the previously selected fields; this summation equals $\mathcal{Q}$ (proportion to be invested). Finally, the SQL query will sort the results, in descending order, according to the extra field representing the summation of the returns. The solution will be displayed in the first row of the SQL query result.

Table 1. Database table comprising the information of a PSP instance $(S, \mathcal{I}, \beta, \mathcal{Q})$.

$\mathcal{I}$	s_1	s_2	s_3	s_4	s_5
0	0	0	0	0	0
20	6	5	3	4	4
40	9	8	5	6	5
60	13	11	8	8	9
80	16	13	10	10	10
100	18	15	12	11	11
120	20	16	15	13	12
140	22	17	16	14	17
160	23	18	17	15	18
180	26	19	19	16	19
200	28	20	20	17	22

SELECT	<security investments>, <profit definition>
FROM	<vector e definition>
WHERE	<constraint definition>
ORDER BY	profit **DESC LIMIT** 1

<security investments> ::=
$\beta_1.\mathcal{I}, \ldots, \beta_{K-1}.\mathcal{I}$
<profit definition> ::=
$(\beta_1.s_1 + \beta_2.s_2 + \ldots + \beta_{K-1}.s_{K-1})$ as profit
<vector e definition> ::=
β as β_1, β as β_2, ..., β as β_{K-1}
<constraint definition> ::=
$(\beta_1.\mathcal{I} + \beta_2.\mathcal{I} + \ldots + \beta_{K-1}.\mathcal{I} \leq \mathcal{Q})$

Figure 1. BNF format of the general SQL query that solves the PSP.

2.2. Maximum Clique Problem

A clique in a graph is defined as a subgraph with pairwise adjacent vertices. The number of vertices contained in a clique determines its size. Finding the largest clique in a graph is an NP-complete problem, and it is referred to as the Maximum Clique Problem (MCP). Applications of this problem appear in different domains such as stock markets [18], collaboration in industry networks [19], social network analysis [20], detection and resolution of conflicts between aircraft [21], and matching protein structures [22], among others.

The MCP can be defined as follows: given a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, is there a subgraph $S \subseteq G$ such that S is a maximum clique $\mathcal{K}_p^*$?

$$\mathcal{K}_p^* = \{S : S = \max\{|G'| : G' \subseteq G, G' \text{ is a clique}\}\} \tag{3}$$

A clique $\mathcal{K}_p$ is a subgraph $S = \{V', E'\}$ that is complete and has $p = |V'|$ nodes. A subgraph S is complete if every pair of nodes $i, j \in V'$ has $(i, j) \in E'$, where $i \neq j$. The mathematical formulation for the MCP is shown in Equation (4), which is based on a graph $G = (V, E)$, where $\overline{E}$ is the complement of the set of edges [23]. The complement refers to those edges that do not belong to E. The formulation maximizes the number of nodes that have in common all their possible edges.

$$\begin{aligned} &\max \sum_{i=1}^{n} x_i, \quad s.t. \quad x_i + x_j \leq 1, \forall (i,j) \in \overline{E}, \\ &\qquad\qquad\qquad\quad x_i \in \{0,1\}, i = 1, \ldots, n \end{aligned} \tag{4}$$

$$\text{where,} \quad x_i = \begin{cases} 1 & \textbf{if } \text{node } i \text{ belongs to the maximum clique} \\ 0 & \textbf{otherwise} \end{cases}$$

Given an instance of the MCP, as a graph $G = (V, E)$, and an integer K, $1 \leq K \leq N$, Torres-Jimenez et al. [16] introduced an exact approach, based on SQL queries, to find the maximum clique in a given G graph.

Figure 2 shows the general SQL query in BNF notation [24] that solves MCP instances. The query uses tables for the set of edges and the set of nodes. The table for the nodes comprises the nodes of the graph V that will be changed. The table for the edges contains two columns (n_1 and n_2) to represent an edge per row and an additional column s to identify the clique. The value one is associated with each edge in the original set E. It serves as a grouping point for counting the edges found in a combination of nodes obtained during the execution of the statement. The solution of this query reports whether or not a clique of size K was found.

Figure 2 shows the structure of the query that solves the MCP and can be stated as follows: given the $G = (V, E)$ graph as the SQL tables *edges*, *nodes* with fields n_1, n_2 and s in *edges* and v in *nodes*, the FROM clause specifies the Cartesian product of K tables as the nodes in the maximum clique of the instance to be solved. In the instance G, the SELECT clause specifies the subset of the maximum size of a clique. Furthermore, the SELECT clause has an extra field representing the size of the clique located in the solution process. The WHERE clause will include the condition that must be satisfied so that the SQL query forms a clique; this condition involves testing that the number of edges formed using the chosen nodes is equal to $\binom{K}{2}$. Let us note that the WHERE clause in $< constraint\ definition >$ contains the $\binom{K}{2}$ conditions joined by logical ORs, which are necessary to validate a clique (each condition implies a logical ORand verifies that an edge is formed). Finally, the SQL query extracts a single solution if it exists to demonstrate the solution to the problem.

SELECT	<subset of nodes>
FROM	<subset definition>
WHERE	<constraint definition>
DESC LIMIT	1

<subset of nodes> ::=
$V_1.v, \ldots, V_K.v$
<subset definition> ::=
nodes as V_1, *nodes* as V_2, ..., *nodes* as V_K
<constraint definition> ::=
EXISTS (SELECT $e.s$ FROM *edges* as e
WHERE $e.n1 = V_1.v\ AND\ e.N2 = V_2.v$ OR ...
GROUP BY $e.s$ HAVING $Count(*)$ = <combinations>)
<combinations> ::=
$\frac{K*(K-1)}{2}$

Figure 2. General SQL query in BNF format to solve the MCP.

It is important to emphasize that although some approaches, before solving the problem of determining the proportions to be invested in each security, first determine the size of the portfolio and the elements that constitute it, the proposed SQL query approach simultaneously solves both problems using the same search space.

2.3. Three-Coloring Graph Problem

Graph Coloring Problems and their generalizations help model various scheduling and assignment problems. Some examples are manufacturing systems [25], DSS for timetabling [26], DSS to solve the scheduling problem of an Earth-observing satellite constellation [27], and self-organization in modern networking [28], among others.

In this section, we present the three-coloring problem on graphs (3CG) and propose a SQL statement for solving small instances of the 3CG problem. The 3CG is a well-known combinatorial problem. The optimization version of this problem searches for the minimum number of colors such that any pair of adjacent vertices of a graph does not have the same color.

The Graph Coloring Problem has applications in several real-world problems such as scheduling [29] or register allocation in a microprocessor [30].

In the scheduling area, different jobs that interfere with each other in using particular resources (e.g., time) must be scheduled to perform a particular job. The jobs and their conflicts can be modeled through a graph. The jobs and their competition for resources can be modeled using a graph. Some instances of the job scheduling problem can be mapped to a 3CG problem.

In register allocation, there is a process that maps the values of program code to a limited set of physical registers in a microprocessor target architecture. The efficient allocation of the registers directly impacts the performance of the execution of a program. An aliveness analysis is conducted to define pairs of program variables that are needed simultaneously (this means that each of the pairs cannot share the same physical register) to find the minimum number of physical registers needed to execute a program efficiently. The pairs of variables represent the edges of a graph, and the program variables represent the graph's nodes. Then, the minimum number of colors needed to guarantee that a pair of adjacent nodes do not have the same color indicates the minimum number of computer registers needed to execute a program efficiently.

The 3CG problem also has applications in real-world problems, e.g., in the resource coordination for wireless environments. In [31], it was shown that an Overlapped Basic Service Sets (OBSSs) environment could be modeled as a planar graph; therefore, the 3CG problem can be used to solve the channel assignment problem [32] in those networks.

In general, the 3CG problem can be formally defined as follows: a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges linking the nodes, which is the color function $\mathcal{C} : V \leftarrow \{1, 2, \cdots, k\}$ that minimizes the value k and assigns a color to each vertex $v_i \in V$ such that $\mathcal{C}(v_i) \neq \mathcal{C}(v_j)$, for all $(v_i, v_j) \in E$.

SQL Approach for Solving 3CG

This subsection presents a SQL statement proposed to solve a 3CG problem instance. An instance of the 3CG problem is described as a graph $G = (V, E)$. Once the query has been created, it is executed in a database management system. The following paragraphs illustrate how to construct a SQL query to solve a 3CG problem instance. The SQL statements show how to solve the problem for K colors; after that, solving 3CG implies using a value of $K = 3$.

The proposed SQL query only uses a table that stores the K colors available for solving a K-coloring instance. The table is called *colors* and is composed by the fields $color, b_k, b_{k-1}, \ldots, b_1$. The field *color* stores the integer value for the different colors. The fields b_i, for $1 \leq i \leq k$, have binary values: if the value is 1, it indicates that edge i has color "i"; if the value is 0, it indicates that edge i does not have color "i".

Figure 3 shows the generalization of the SQL query to solve MCG instances. The query is given using the BNF notation [24]. The clause SELECT <*color of nodes*>, <*colors used*> defines that the results from the query will contain the color for each node and the list of colors used. Due to clause LIMIT 1, there will be only one tuple resulting from the query. The clause ORDER BY $b_k, b_{k-1}, \ldots, b_1$ is used to ensure that the solution, if it exists, contains the minimum number of colors (solutions that maximize the number of $b_i = 0$ will appear first). The definition of the set of colors that the nodes can use is performed in the clause FROM <*color definition*>. The constraint of the MCG problem that tells that no pair of nodes that form an edge in the graph must share colors is specified in the clause WHERE <*constraint definition*>, which will contain an expression like $V_i.color \neq V_j.color$ whenever $(v_i, v_j) \in E$, for instance $G = (V, E)$.

SELECT	<color of nodes>, <colors used>
FROM	<color definition>
WHERE	<constraint definition>
ORDER BY	$b_k, b_{k-1}, \ldots, b_1$ **LIMIT** 1

<color of nodes> ::=
$V_1.color$ as $color_{v_1}$, ..., $V_n.color$ as $color_{v_n}$
<colors used> ::=
$(V_1.b_k$ OR $V_2.b_k$ OR ... OR $V_n.b_k)$ as b_k,
$(V_1.b_{k-1}$ OR $V_2.b_{k-1}$ OR ... OR $V_n.b_{k-1})$ as b_{k-1},
$\vdots$
$(V_1.b_1$ OR $V_2.b_1$ OR ... OR $V_n.b_1)$ as b_1
<color definition> ::=
$colors$ as V_1, $colors$ as V_2, ..., $colors$ as V_n
<constraint definition> ::=
$V_i.color <> V_j.color$ AND ...

Figure 3. BNF syntax of a SQL query that solves coloring problems in graphs.

3. Experimental Design and Results

The experimental design in this section validates the proposed SQL statements as tools for optimization. It defines two experiments for this purpose. Section 3.1 presents the first experiment, which evaluates the performance of a SQL statement. The performance of SQL is assessed by solving the 3CG in planar graphs and by comparing its performance with the performance of naive brute force strategies; for both cases, the instances were chosen according to the context that matches the special conditions of this research's subject of study. Such particular conditions demonstrate the suitability of SQL statements for application under special situations. The second experiment demonstrates the easy adaptation of the proposed SQL queries to solve other related optimization problems, i.e., a Special Case of the Community Network Problem, the Knapsack Problem, and the Channel Assignment Problem.

3.1. First Experiment: Performance of Using SQL Statements

This experiment used SQL queries to solve different optimization problems. It was developed in two phases. The first phase evaluates exclusively the performance of solving the 3CG in planar graphs using SQL. The use of planar graphs is motivated due to their application in the solution of the channel assignment problem in an overlapped Basic Service Sets environment [31], which can be modeled through planar graphs. As a result, even though the approach has an exponential growth in time, it is observed that cases with even dozens of nodes are solved in around ten seconds. The second phase extends the performance evaluation on the other revised problems. The PSP, MCP, and 3CG are solved using SQL statements and compared in performance against naive brute force strategies under a small set of random instances. The results show competitiveness in using SQL queries over brute force algorithms. The remainder of this section details better each of the phases of this experiment and their results.

3.1.1. Optimization of the 3CG for Planar Graphs Using SQL

The set of random planar graphs was generated using the `C++` class library LEDA (http://www.algorithmic-solutions.com/index.php, accessed on 10 April 2022). In particular, each instance was generated using the function `random_planar_graph(G,N,M)`. This function first generates a maximal planar graph with N nodes, $n \geq 3$, and $3n - 6$ edges. For $N = 1$, a maximal planar graph is formed by a single isolated node. For $N = 2$, the graph consists of two nodes and one edge; for $N = 3$, the graph consists of three nodes and three edges. For $N > 3$, a random maximal planar map with $N - 1$ nodes is constructed first, and then, an additional node is put into a random face. The function first generates a maximal planar graph and then deletes all but M edges to generate a graph with M edges.

Table 2 shows the set of 48 random planar graphs generated using LEDA. The table is split into two parts: Columns 2 and 3 contain the number of nodes and edges of each instance, respectively. The instances include planar graphs with some nodes varying from 20 to 45. The random instances are formed by random planar graphs of size $N = \{20, 30, 35, 40, 45\}$. For each graph $G = (V, E)$, the number of nodes $N = |V|$ and edges $M = |E|$ is included. Furthermore, the time (in secs.) spent by the SQL approach on finding a three coloring is reported. The last column presents whether or not a three-coloring was possible.

Table 2. Results of the set of random instances for the 3-coloring problem solved.

Case	N	M	Time (s)	3-Coloring?
1	20	26	0.27	Yes
2	20	26	0.27	Yes
3	20	26	0.27	Yes
4	20	26	0.27	Yes
5	20	30	0.02	Yes
6	20	30	0.02	Yes
7	20	30	0.02	Yes
8	20	30	0.02	Yes
9	25	41	0.19	Yes
10	25	41	0.19	Yes
11	25	41	0.19	Yes
12	25	41	0.19	Yes
13	25	45	0.19	Yes
14	25	45	0.19	Yes
15	25	45	0.19	Yes
16	25	45	0.19	Yes
17	30	44	5.94	Yes
18	30	44	5.95	Yes
19	30	44	5.96	Yes
20	30	44	5.95	Yes
21	30	48	2.27	Yes
22	30	48	6.69	Yes
23	30	48	2.40	Yes
24	30	48	0.04	No
25	35	57	10.78	Yes
26	35	57	10.73	Yes
27	35	57	10.76	Yes
28	35	57	10.75	Yes
29	35	64	0.01	No
30	35	64	0.01	No
31	35	64	0.01	No
32	35	64	0.01	No
33	40	58	618.55	Yes
34	40	58	623.89	Yes
35	40	58	616.83	Yes
36	40	58	620.44	Yes
37	40	72	4.55	Yes
38	40	72	4.55	Yes
39	40	72	4.55	Yes
40	40	72	4.57	Yes
41	45	67	1175.25	Yes
42	45	69	1.47	No
43	45	71	1.51	No
45	45	73	1128.23	Yes
44	45	75	41.52	No
46	45	77	2.21	No
47	45	79	1.48	No
48	45	81	1.48	No

The random instance generators were implemented using C++ and compiled with g++. We used a desktop computer with an Intel(R) Core(TM) 2 processor, 3Gb RAM, the Ubuntu 15.04 operating system, and MySQL 5.5 as the DBMS. Table 2 also shows the results from finding a three-coloring for the random planar graphs. The results are shown in Columns 4 and 5. Column 4 shows the time spent by the DBMS to find a solution if it exists. Column 5 shows if a three-coloring was possible or not for the instances considered. According to the results, only 37 of the 48 instances were three-colorable. The DBMS spent a time of up to 1175.25 s to determine non-three-colorability. On the other side, the DBMS only required at most 41.52 s to determine that there was no solution for the remaining 11 instances.

3.1.2. Naive Brute Force Algorithms

The PSP, MCP, and 3CG are related to a broader range of optimization problems. After a careful revision of the literature of related problems, it was observed that there are real-life cases of a small size that involve only a few nodes (thirty or forty) in the case of MCP and 3CG, or for the PS,P a few securities (between twenty and forty), for example in the case of the PSP, Knapsack, Cargo Loading Problem, and Cryptocurrency [6,8]; or the cases of MCP, Signal Processing Problem [7], and Community Detection [5,33]; or in Time Tabling and Channel Assignment and Graph Coloring [3,4,14,29,31]. Hence, this section validates the proposed approach using small random instances that match real-life sizes found in the literature.

The validation of the performance of the use of SQL to solve optimization problems takes into consideration the following conditions: (1) the cases are small and can be managed by a DBMS; (2) the development of an ad hoc or specialized strategy is not justified; (3) the software developer lacks the time, or skills, to develop programs for finding solutions for the optimization problems; or (4) implementing a naive brute force strategy is expensive, and efficiency is compromised due to the programming language use. As a result, the proper strategies for comparison were naive brute force strategies because they are the most common strategy to be implemented under the previous situations. Their implementation also used the script language Python, and the set of instances was randomly generated using an Erdös–Rényi random graph generator and a modification of the random generator from Kumar and Singh [34] to create PSP instances.

Table 3 summarizes the results derived from the comparison. The experiment set a time limit of 72,000 s for solving the problem instances; the cases that were not solved within such a time limit are named as *Unsolved*. The results were classified according to the problem analyzed, the PSP, MCP, and 3CG. The first column shows the problem, and the second is the instance data. The third column shows the time in seconds spent by the naive brute force algorithm to solve them. The last column is the corresponding time needed for the respective SQL query.

Let us point out that both approaches followed an exponential growth in the time needed to find a solution. However, the time needed for running the respective SQL query was significantly lesser than the time needed for the naive brute force algorithm.

3.2. Second Experiment: Other Problems

This experiment demonstrates the easy adaptation of the proposed SQL queries to solve other related optimization problems. In order to achieve this, it used two cases of study. The first of them was a particular case of the Community Network Problem, where it models a problem in scientometrics as an MCP problem instance (see Section 3.2.1). The second case of the study shows how to use the PSP and 3CG to solve the Knapsack and the Channel Assignment Problem. As a result of these experiments, the ease of using the proposed SQL statements to solve a broader range of optimization problems is evidenced.

Table 3. Time in seconds spent by brute force strategies and SQL approaches in different instances.

Problem	Instance	Brute Force Solver	SQL Approach
PSP	Securities = 10, $\mathcal{Q} = 5000$, $U(0, 50, 000)$	0.02	0.01
PSP	Securities = 20, $\mathcal{Q} = 10,000$, $U(0, 50, 000)$	4.27	0.65
PSP	Securities = 30, $\mathcal{Q} = 15,000$, $U(0, 50, 000)$	4743.36	0.11
PSP	Securities = 40, $\mathcal{Q} = 20,000$, $U(0, 50, 000)$	Unsolved	903
MCP	$N = 20, M = 30$	2.74	4.12
MCP	$N = 30, M = 60$	2336.26	36.49
3CG	$N = 20, M = 30$	530.363	0.03
3CG	$N = 30, M = 60$	Unsolved	0.23
3CG	$N = 40, M = 125$	Unsolved	43

3.2.1. Case of Study: Special Community Network Problem

This section presents another case of study to demonstrate that the use of SQL queries can be helpful in the solution of problems found in the analysis of data related to a research community. Remarkably, the problem consists of identifying self-referencing groups in a research community. Scientometrics is concerned with measuring and analyzing science, technology, and innovation data. One application in this area is derived from the solution to the MCP; it consists of identifying groups that intentionally or unintentionally reference each other in articles in a research community. For this purpose, it is required to identify all pairs of references between members in a research community, i.e., identifying a CLIQUE in the organization of the references between authors.

To show an example, let us take the database described as follows. Let us suppose that the information relating to the articles published by a research community in a set of journals and their citations is organized in a database with the tables shown in Table 4. This database organizes the core data required to describe a publication derived from a journal or conference, i.e., the name of its publisher, address, articles, and citations. Then, self-referencing groups can be seen as groups in which all their members refer to each other in their articles, i.e., a set of authors that can be identified as referencing other members in their publications. We can model it as a CLIQUE problem to find such groups and their cardinality (the number of members who form the group).

Table 4. Typical organization scheme for database tables that contain the publishing information of the articles in a set of journals.

(a)	(b)	(c)	(d)
Journals	**Articles**	**Authors**	**Citations**
id_journal	id_article	id_author	id_journal
name	title	name	id_article
publisher	id_author	affiliation	cited_by
address	id_journal		year_cited
	year		
	volume		
	pages		

Firstly, let us consider the graph $G = (V, E)$ formed by the set of nodes V, where each node represents a different author in the database. This graph will contain an edge (v_i, v_j) if the author i has at least one article in which he/she has cited one article of author j and author j also has at least one article in which he/she has cited author i. Then, the solution of the MCP on graph G will yield a self-referencing group, and the cardinality of such a group will be its size.

The SQL statement presented in Figure 4 can be used to construct the graph G from a database as the one described previously. Figure 4a presents the identification of all the pairs (i, j), i.e., author i has been cited by author j. Figure 4b shows how to identify the edges of the graph; to do so, it performs a join operation between two tables resulting from the statement presented in Figure 4a, then it chooses only those tuples where the author and the author that cites appear in both cites, indicating a possible self-referencing. Then, the construction of the graph is straightforward; all the different authors appearing are the nodes, and the tuples in the result of this statement will become the edges.

(a)

```
SELECT
  a.id_author,
  r.cited_by
FROM
  Authors AS a,
  Articles AS p,
  Citations As r
WHERE
  a.id_author = p.id_author AND
  p.id_article = r.id_article
```

(b)

```
SELECT
DISTINCT t1.id_author, t1.cited_by
FROM
  (SELECT
    a.id_author AS id_author,
    r.cited_by AS cited_by
  FROM
    Authors AS a,
    Articles AS p,
    Citations AS r
  WHERE
    a.id_author = p.id_author  AND
    p.id_article = r.id_article) AS t1,
  (SELECT
    a.id_author AS id_author,
    r.cited_by AS cited_by
  FROM
    Authors AS a,
    Articles AS p,
    Citations AS r
  WHERE
    a.id_author = p.id_author  AND
    p.id_article = r.id_article) AS t2,
WHERE
  t1.id_author = t2.cited_by AND
  t1.cited_by = t2.id_author
```

Figure 4. BNF syntax of a SQL query that determines the G graph used to identify possible self-referencing groups. (**a**) To identify all the pairs (i, j). (**b**) To identify the edges of the graph.

To solve an MCP instance, see Section 2.2, where we described the solution of such an optimization problem using a SQL query. With this case of study, the relevance of the use of SQL in solving a problem that is important in scientometrics is pointed out.

3.2.2. Case Study: Knapscak and Channel Assignment Problem

This section shows how to adapt the SQL queries that solve the PSP and 3CG to solve instances of the Knapsack and Channel Assignment problems.

Let us begin with the Knapsack Problem: defined by a maximum weight W, a set of objects $O = \{o_1, o_2, \cdots, o_n\}$ having each object o_i an associated weight w_i and value v_i. The problem searches to maximize the value of a chosen subset of objects $O' \subseteq O$ while avoiding exceeding the maximum capacity W (see [34]). This problem has a straightforward transformation into the PSP. A tuple $(S, \mathcal{I}, \beta, \mathcal{Q})$ can be defined as follows. The set S of

securities is formed by the n objects; hence, $S = \{s_1, s_2, \cdots, s_n\}$. The investment portions I are defined according to the weights of the objects. The quantity $\mathcal{Q}$ will be equal to the maximum weight W. Finally, the expected return matrix β will set a cell $\beta_{i,j} = v_j$ if the investment q_i is equal to its weight; otherwise, $\beta_{i,j} = 0$. The SQL query can be used without modification to solve the Knapsack Problem with this transformation in mind.

The Channel Assignment Problem (CAP) (inspired by its implementation in [31]) is almost as simple as the Knapsack one, except for a minor change in the SQL query to be fully applied.

Taking into account the definition in [14], the CAP is a natural generalization of graph coloring. The problem associates weights with the edges of a graph. For example, the special case of the CAP called One-Bounded CAP is equivalent to graph coloring. It searches to assign values to nodes between $\{1, \cdots, K\}$ with the smallest integer K possible such that the difference $|v(x) - v(y)| \geq 1$, where (x, y) are edges of the graph G that defines the instance of the problem. When a value l is used instead of 1 in CAP, the problem is called l-bounded CAP, and this one generalizes graph coloring.

The SQL query defined in Section 2.3 to solve the 3CG can be slightly modified to solve the more general l-CAP problem. This only requires modifying the $< constraint\ definition >::= V_i.color <> Vj.color AND \cdots$ in the SQL proposed, such that it transforms into something like this: $< constraint\ definition >::= (V_i.color - Vj.color) >= weight(v_i, v_j) AND \cdots$, where $weight(v_i, v_j)$ is the information concerning the weight associated with the $edge(v_i, v_j)$ in the graph instance. This latter information can be introduced as a constant when expanding the SQL statement or as a table for the database.

The use of the solutions for the PSP, MCP, and 3CG to solve other optimization problems shows the validity of using SQL queries as an alternative to solve complex optimization problems.

4. Discussion

This section discusses several points of view derived from implementing the approach based on SQL queries to solve optimization problems.

The structure of SQL statements offers a straightforward method to create the queries. The typical steps for any of the presented queries are: (1) The definition of the required solutions in the SELECT clause; (2) The definition of the search space in the FROM clause; (3) The definition of the necessary conditions (constraints) to obtain a feasible solution in the WHERE clause. Additionally, the creation of the queries requires a previous definition of the database tables that they will use. This analysis gives evidence of the simplicity of the approach.

In order to demonstrate the advantages of the proposed solutions, two experiments were proposed. The first experiment validated the performance of using SQL queries over different sets of instances. One set involved random planar graphs, and the optimization problem solved was the 3CG. The time spent solving such instances was up to 20 min, in graphs with several dozen nodes. These instances show that the time spent is significantly impacted by the number of conditions in the WHERE clause; the time decreases with a larger number of conditions.

Another set of instances involved small-sized random cases. The performance of using SQL queries was compared against the performance of naive brute force strategies. In particular, the comparison with brute force strategies is justified since, in most cases, DBMS users have no experience in creating complex optimization algorithms, and brute force algorithms are the simplest to implement; moreover, the implementation of such algorithms used an ordinary script language (in this case, Python). The analysis of the results from this experiment demonstrates that a quick implementation (i.e., a naive brute force algorithm) could be a bad choice for solving an optimization problem and that the use of a SQL query is a good alternative. Furthermore, they showed that using a SQL query is an easy-to-implement strategy and a more convenient option than brute force algorithms.

Let us point out the independence of the platform required for implementation since it will be in a system that can store information in a database.

The second experiment demonstrates the versatility of the solution of using a SQL query applying the transformation of instances of the Knapsack Problem into PSP instances and the extension of the SQL query for the 3CG to solve the Channel Assignment Problem. Hence, SQL statements to optimize problems are easy to adapt and can be applied with little effort to other optimization problems. This experiment raises the primary correlation between the 3CG, PSP, and MCP, which is their ability to extend existing solution approaches to other optimization problems. This justification became the focus in the present research to analyze their SQL statements.

Table 5 presents a summary of the time and space complexity analysis of the SQL statements. Each row represents a different SQL statement, one per considered optimization problem. Columns 2 and 3 show the time and space complexities, respectively. Column 4 shows the initial memory required by the database tables. The time complexity of all the SQL statements depends on the number of join operations and their cost. For the PSP and MCP, the time complexity is due to the size of the tables involved, being n, and the statements involving k join operations. For the 3CG, the size of the tables is k, the number of colors, and there are n join operations. The growth of the space complexity is in accordance with the results from the join operations, given that every result must be stored for the next operation; therefore, the space complexity can be equal to the time complexity in the worst case. Finally, let us point out that the initial memory required by the database depends on the tables that it initially contains.

Table 5. Time and space complexity analysis.

Strategy	Time Complexity	Space Complexity	Memory for DB Tables
PSP SQL	$O(n^k)$	$O(n^k)$	$O(n^2)$
MCP SQL	$O(n^k)$	$O(n^k)$	$O(n+m)$
3CG SQL	$O(k^n)$	$O(k^n)$	$O(K)$

Finally, we can point out the possibility that the approaches could help managerial decision-making. Under circumstances where the time spent is not a problem or the sizes of the instances are small, a DB manager can include the SQL statement as part of a report that meets the necessity of the information needed by a decision-maker person. The SQL statements are ready-to-use tools that provide easy-to-integrate results within a bigger computer system. As a matter of fact, the use of SQL statements opens the possibility of optimizing on-the-go without restructuring or recompiling part of the system to integrate different platforms. The presented solutions are pretty convenient for a web system without infrastructure for optimization since SQL could solve some specialized real-life situations.

5. Conclusions

This paper presented a novel approach for solving complex optimization problem instances using an approach based on SQL queries. The solutions can support a decision-making process. The approach was evidenced in the solution of three classical optimization problems (Portfolio Selection Problem (PSP), Maximum Clique Problem (MCP), and Three-Coloring Problem (3CG)) using only SQL queries, and the validity and performance of this approach were assessed.

The model for solving the 3CG problem using a SQL query was tested on a set of 48 random planar graphs. Using the query, the solution of such instances required less than 20 min per instance. Furthermore, the performance of the SQL queries was tested by comparing them against naive brute force strategies. Results on the latter showed that the naive algorithm might be the wrong choice when all the circumstances are not considered and that the use of SQL queries represents, compare to them, a simple, efficient, and easy-to-implement strategy.

The versatility of the solutions based on SQL queries was studied on two cases of studies. One of them was presented to show how to use SQL queries to solve Community Network problems related to data analysis in a research community. The case study proposed a solution to the problem of self-reference groups. The other case of study demonstrated how other optimization problems can be easily adapted to exploit the use of the proposed queries.

In general, the simplicity of the SQL language makes it easier to use than a procedural language because SQL does not require complex structures or programming skills to state the solution for an optimization problem instance. Furthermore, the use of SQL queries makes the proposed approach easy to use and understand. However, the approach has limitations related to the performance of the DBMS used to solve the queries; this can affect the size of the instances that can be solved according to some time constraints.

For this reason, it is essential to emphasize that the proposal to use SQL queries to solve optimization problems does not pretend to compete with other strategies regarding speed. Still, it does so concerning the quality of the solutions and, above all, in the simplicity and novelty. Readers interested in knowing the best-reported algorithms to solve the MCP, PSP, and 3CG could review the works [35–38].

A final conclusion from the research and results shown in this paper is that it is possible to solve relevant optimization problems using a high-level non-procedural language without writing a line of code, i.e., it could be easy to develop a basic DSS to help decision-makers to have solutions for complex optimization problem instances.

Author Contributions: Conceptualization, J.T.-J.; methodology, J.T.-J., N.R.-V. and H.A.-G.; software, N.R.-V.; validation, J.T.-J., N.R.-V., M.D.-l.-T. and H.A.-G.; formal analysis, J.T.-J.; investigation, J.T.-J.; resources, M.D.-l.-T. and H.A.-G.; data curation, N.R.-V.; writing—original draft preparation, J.T.-J., N.R.-V. and H.A.-G.; writing—review and editing, J.T.-J., N.R.-V., M.D.-l.-T. and H.A.-G.; visualization, N.R.-V.; supervision, J.T.-J.; project administration, J.T.-J.; funding acquisition, J.T.-J. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The second author acknowledges the support from CONACYT Projects No. 1340, 3058, A1-S-11012, 236154, and 280081 and to the Laboratorio Nacional de Tecnologías de la Información (LANTI).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Suresh, L.; Simha, J.B. Efficiency Analysis of Efficient SQL based Clustering Algorithm. *Int. J. Comput. Sci. Netw. Secur.* **2008**, *8*, 293–298.
2. Wolfram, D. Applications of SQL for informetric frequency distribution processing. *Scientometrics* **2006**, *67*, 301–313. [CrossRef]
3. Socala, A. Tight Lower Bound for the Channel Assignment Problem. *ACM Trans. Algorithms* **2016**, *12*, 1–19. [CrossRef]
4. Ganguly, R.; Roy, S. A Study on Course Timetable Scheduling using Graph Coloring Approach. *Int. J. Comput. Appl. Math.* **2017**, *12*, 469–485.
5. Lu, Z.; Wahlström, J.; Nehorai, A. Community Detection in Complex Networks via Clique Conductance. *Sci. Rep.* **2018**, *8*, 5982. [CrossRef] [PubMed]
6. Cho, M. The Knapsack Problem and Its Applications to the Cargo Loading Problem. *J. Anal. Appl. Math.* **2019**, *13*, 48–63.
7. Douik, A.; Dahrouj, H.; Al-Naffouri, T.; Alouini, M.S. A Tutorial on Clique Problems in Communications and Signal Processing. *arXiv* **2020**, arXiv:1808.07102.
8. Aljinovic, Z.; Marasovic, B.; Šestanovic, T. Cryptocurrency Portfolio Selection—A Multicriteria Approach. *Mathematics* **2021**, *9*, 1677. [CrossRef]
9. Brodsky, A.; Egge, N.; Wang, X. Supporting Agile Organizations with a Decision Guidance Query Language. *J. Manag. Inf. Syst.* **2012**, *28*, 39–68. [CrossRef]
10. Šikšnys, L.; Pedersen, T.B. SolveDB: Integrating Optimization Problem Solvers Into SQL Databases. In Proceedings of the 28th International Conference on Scientific and Statistical Database Management, Budapest, Hungary, 18–20 July 2016; pp. 1–12. [CrossRef]
11. Šikšnys, L.; Pedersen, T.; Nielsen, T.; Frazzetto, D. SolveDB+: SQL-Based Prescriptive Analytics. In Proceedings of the 24th International Conference on Extending Database Technology (EDBT), Nicosia, Cyprus, 23–26 March 2021; pp. 1–12. [CrossRef]

12. Sakanashi, G.; Sakai, M. Transformation of Combinatorial Optimization Problems Written in Extended SQL into Constraint Problems. In Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming (PPDP '18), Frankfurt am Main, Germany, 3–5 September 2018; pp. 1–13. [CrossRef]
13. Makrynioti, N.; Ley-Wild, R.; Vassalos, V. Machine learning in SQL by translation to TensorFlow. In Proceedings of the Fifth Workshop on Data Management for End-To-End Machine Learning (DEEM 2021), Virtual Event, China, 20–25 June 2021; pp. 1–11. [CrossRef]
14. Král, D. An exact algorithm for the channel assignment problem. *Discret. Appl. Math.* **2005**, *145*, 326–331. [CrossRef]
15. Torres-Jimenez, J.; Avila-George, H.; Rangel-Valdez, N.; Martinez-Pena, J. Optimization of investment options using SQL. In Proceedings of the 12th Ibero-American Conference on Advances in Artificial Intelligence–IBERAMIA, Bah ia Blanca, Argentina, 1–5 November 2010; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6433, pp. 30–39. [CrossRef]
16. Torres-Jimenez, J.; Rangel-Valdez, N.; Avila-George, H.; Gonzalez-Hernandez, L. MAXCLIQUE Problem Solved Using SQL. In Proceedings of the Third International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA), St. Maarten, The Netherlands Antilles, 22–26 May 2011; pp. 83–88. Available online: https://www.iaria.org/conferences2011/DBKDA11.html (accessed on 10 April 2022).
17. Markowitz, H. Portfolio Selection. *J. Financ.* **1952**, *7*, 77–91. [CrossRef]
18. Vizgunov, A.; Goldengorin, B.; Kalyagin, V.; Koldanov, A.; Koldanov, P.; Pardalos, P.M. Network approach for the Russian stock market. *Comput. Manag. Sci.* **2014**, *11*, 45–55. [CrossRef]
19. Antonelli, D.; Caroleo, B. An integrated methodology for the analysis of collaboration in industry networks. *J. Intell. Manuf.* **2012**, *23*, 2443–2450. [CrossRef]
20. Wen, X.; Chen, W.N.; Lin, Y.; Gu, T.; Zhang, H.; Li, Y.; Yin, Y.; Zhang, J. A maximal clique based multiobjective evolutionary algorithm for overlapping community detection. *IEEE Trans. Evol. Comput.* **2017**, *21*, 363–377. [CrossRef]
21. Lehouillier, T.; Omer, J.; Soumis, F.; Desaulniers, G. Two decomposition algorithms for solving a minimum weight maximum clique model for the air conflict resolution problem. *Eur. J. Oper. Res.* **2017**, *256*, 696–712. [CrossRef]
22. Balaji, S.; Swaminathan, V.; Kannan, K. A simple algorithm for maximum clique and matching protein structures. *Int. J. Comb. Optim. Probl. Inform.* **2010**, *1*, 2.
23. Bomze, I.M.; Budinich, M.; Pardalos, P.M.; Pelillo, M. The Maximum Clique Problem. In *Handbook of Combinatorial Optimization*; Kluwer Academic Publishers: Dordrecht, The Netherlands, 1999; Chapter 1, pp. 1–74.
24. Friedman, J. A computer system for transformational grammar. *Commun. ACM* **1969**, *12*, 341–348. [CrossRef]
25. Błażewicz, J.; Ecker, K.H.; Schmidt, G.; Weglarz, J. *Scheduling in Computer and Manufacturing Systems*; Springer Science & Business Media: New York, NY, USA, 2012.
26. Cangalovic, M.; Schreuder, J.A.M. Exact colouring algorithm for weighted graphs applied to timetabling problems with lectures of different lengths. *Eur. J. Oper. Res.* **1991**, *51*, 248–258. [CrossRef]
27. Wang, P.; Reinelt, G.; Gao, P.; Tan, Y. A model, a heuristic and a decision support system to solve the scheduling problem of an earth observing satellite constellation. *Comput. Ind. Eng.* **2011**, *61*, 322 –335. [CrossRef]
28. Ahmed, F.; Tirkkonen, O.; Peltomäki, M.; Koljonen, J.M.; Yu, C.H.; Alava, M. Distributed graph coloring for self-organization in LTE networks. *J. Electr. Comput. Eng.* **2010**, *2010*, 5. [CrossRef]
29. Marx, D. Graph Colouring Problems and Their Applications in Scheduling. *Period. Polytech. Electr. Eng.* **2004**, *48*, 11–16.
30. Cooper, K.D.; Dasgupta, A. Tailoring Graph-coloring Register Allocation For Runtime Compilation. In Proceedings of the International Symposium on Code Generation and Optimization (CGO '06), New York, NY, USA, 26–29 March 2006; pp. 39–49. [CrossRef]
31. Zheng, L.; Hoang, D.B. Applying graph coloring in resource coordination for a high-density wireless environment. In Proceedings of the 8th IEEE International Conference on Computer and Information Technology, Sydney, Australia, 8–11 July 2008; pp. 664–669. [CrossRef]
32. Fam, B.W.; Millen, J.K. The Channel Assignment Problem. In Proceedings of the 1983 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 25–27 April 1983; p. 107. [CrossRef]
33. Yan, B.; Gregory, S. Detecting Communities in Networks by Merging Cliques. In Proceedings of the 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems, Shanghai, China, 20–22 November 2009; pp. 1–5. [CrossRef]
34. Kumar, R.; Singh, P. Assesing solution quality of biojective 0-1 knapsack problem using evolutionary and heuristic algorithms. *Appl. Soft Comput.* **2010**, *10*, 711–718. [CrossRef]
35. Wu, Q.; Hao, J.K. A review on algorithms for maximum clique problems. *Eur. J. Oper. Res.* **2015**, *242*, 693–709. [CrossRef]
36. Ponsich, A.; Lopez Jaimes, A.; Coello Coello, C.A. A survey on multiobjective evolutionary algorithms for the solution of the portfolio optimization problem and other finance and economics applications. *IEEE Trans. Evol. Comput.* **2013**, *17*, 321–344. [CrossRef]
37. Beigel, R.; Eppstein, D. 3-coloring in time O (1.3289 n). *J. Algorithms* **2005**, *54*, 168–204. [CrossRef]
38. Guo, J.; Zhang, S.; Gao, X.; Liu, X. Parallel graph partitioning framework for solving the maximum clique problem using Hadoop. In Proceedings of the 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA), Beijing, China, 10–12 March 2017; pp. 186–192.