Effectively indexing Data Warehouses: A Constraint Programming Perspective

Youcef Ariouat[1]*, Benamor Ziani[1], and Youcef Ouinten[1]

[1]LIM Laboratory, Amar Telidji University of Laghouat, Bp 37G, Ghardaia Road, Laghouat 03000, Algeria
*Corresponding author Email: y.ariouat@cu-barika.dz

***Abstract*— Data warehouses have become, nowadays, at the core of decisional systems. The Index Selection Problem (ISP) is a challenging problem in data warehouses physical design. Regarding the NP-hard nature of this problem, existing solutions rely heavily on heuristics which constitutes a major drawback. In this paper, we propose a novel exact approach to the ISP based on Constraint Programming (CP). We formulate the problem as a Constraint Optimization Problem in a declarative way, then its resolution is automatically supported by a generic CP solver. Our proposed approach has also the advantage of being declarative, flexible, and expandable as it allows incorporating various kinds of user preferences, expressed as constraints, as well as choosing or defining new search strategies. Experimental results confirm our expectations and show that our approach scales well enough to solve much larger realistic instances in a faster and more effective way compared to well-known state-of-the-art approximation approaches.**

Keywords*— *data warehouses; bitmap join index; constraint programming; index selection problem; constraint optimization problem; decisional systems.

I. INTRODUCTION

This Data warehouses (DWs) are at the core of today's decision support systems. Their main role is to transform the data they contain into strategic indicators for smart decision making. Such indicators are expressed by means of complex analytical queries that often scan huge volumes of historically collected data. Indexing remains one of the most effective techniques used by the data warehouse administrator (DWA) for data access improvement since it allows finding the desired information without the need to scan the whole data [1]–[4]. Particularly, bitmap join indexes (BJIs) [5] have proven to be an efficient solution suited to avoid costly join operations by performing fast binary operations on the index level rather than scanning the data [3]. Furthermore, the binary nature of BJIs leads to their size being significantly smaller and makes them highly amenable to compression and encoding.

Although BJIs can significantly improve the query processing performance, selecting just those that are interesting is yet a big challenge. In the literature, this is known as the ISP which is proven to be NP-hard [6]. The complexity stems from the potential huge search space that the DWA faces. Indeed, with k attributes, that appear in a given workload, $(2^k - 1)$ different indexes can be considered, leading to $(2^{2^k-1} - 1)$ possible index configurations [4]. Consequently, the increase in the number of attributes triggers a rapid growth of possible indexes, leading to a large search space.

In this particular context, works devoted to solve the BJI selection problem (BJI-SP), may be clustered into three categories: heuristic-based algorithms [7], metaheuristics-based algorithms [1], [8]–[11] and data mining-based techniques [4], [12]–[15]. Heuristic-based approaches [7] try, in a first phase, to prune its combinatorial search space into a small subset of possible candidate indexes. Then, in a second phase, greedy search algorithms (bottom-up or top-down search) are used to recommend a final set of indexes regarding a storage budget specified by the DWA. Despite their simplicity, these approaches rely heavily on heuristics and does not guarantee the optimal solution. Indeed, in case of severely limited disk space, such approaches may result in not considering some very good indexes [16], [17]. Classical metaheuristic algorithms have also been used to solve the BJI-SP in DW context. Among these methods, we can mention the Ant Colony Optimization [8], Genetic Algorithms [10], Artificial Immune Systems [9], Binary Particle Swarm Optimization [11] and Multi-objective Evolutionary Algorithms [1]. Other directions to tackle the BJI-SP have been proposed and are based on data mining techniques [4], [12]–[15]. The intuitive idea is that the importance of an index is strongly correlated with its appearance frequency in the workload. Thus, these approaches use the concept of closed frequent itemset [14] or the maximal frequent itemset technique [4] to efficiently prune the initial search space. A second phase is necessary for selecting the best index configuration that fits within the reserved storage space using greedy algorithms. Although meta-heuristics and data mining approaches have an attractive theoretical success, it is well-known that they still suffer from their parameter settings challenging. Indeed, finding the best parameter values is not a trivial task for the administrator and is very hard to set.

In this paper, we seek to address the shortcomings of existing approximation techniques by proposing an exact approach that would allow an effective search strategy and suggest better solutions. Specifically, through the application of constraint programming (CP).

Indeed, an exact resolution for the ISP can be very computationally expensive due to the NP-hard nature of the problem. Fortunately, the last two decades have seen the rise of effective paradigms for the exact resolution of NP-hard problems, such as CP [18], [19]. The latter has been proved to be a very effective paradigm in solving large, particularly combinatorial and computationally hard, problems in different fields [20]. Therefore, CP has been successfully used in both academic and industry to solve a variety of optimization problems such as production planning, scheduling, packing, airport traffic-control, hardware validation, and robotics, to name but a few [18], [20]. The considerable development of CP can be explained by the significant increase in the processing power of the computers as well as advanced CP search strategies that result in developing effective and scalable solvers [21].

To the best of our knowledge, in the literature, the BJI-SP has never been addressed using CP techniques. This will be at the core of this work. Our main contributions can be summarized as follows: First, we describe a system architecture for efficiently solving BJI-SP. Second, we present a CP formulation for the problem. Third, we demonstrate the relevance of our approach through an implementation under *Choco* [22], an open-source solver used in industry and research. We also report preliminary results that demonstrate that our approach has good performances and outperforms well known index selection techniques.

II. BACKGROUND

To facilitate the understanding of our approach, this section is intended to briefly sketch key notions including the ISP statement, cost models and basic concepts of CP that are used in modelling and solving the ISP.

A. Index Selection in Data Warehouses: Problem Statement

In a relational context, data warehouses are often modelled according to a star schema with rather a large fact table F and a set of descriptive dimension tables $D = \{d_1, d_2, \ldots, d_n\}$. The dimension tables are linked to the fact table through foreign key relationships. Data stored in a data warehouse are often analysed by means of complex star queries that join the central fact table with multiple referenced dimension tables. In practice, star queries processing may take hours or even days and, thus, BJIs are widely used to improve query processing performance [1], [3]–[5]. For a given workload, a naive approach consists in exhaustively materializing all possible BJIs. Nevertheless, the space limitation of the system would hinder the DWA from doing this. Therefore, the problem of selecting an appropriate set of indexes can be formalized as an optimization problem with constraints as follows: Given a representative workload consisting of a set of n queries $W = \{q_1, q_2, \ldots, q_n\}$, and given a storage constraint S_{max}, the goal is to provide an index configuration (set of indexes) C_{opt}t among all possible configurations so that the cost of processing the workload W using C_{opt} is minimum subject to the limit on the total indexes' size, S_{max} i.e.:

$$\begin{cases} \sum_{i=1}^{n} cost(q_i, C_{opt}) \text{ is minimium.} \\ \sum_{j=1}^{m} size(I_j) \leq S_{max}. \end{cases} \quad (1)$$

Where, $cost(q_i, C_{opt})$ is the cost of processing query q_i using the configuration C_{opt} and $size(I_j)$ is the needed disk space to store the index $I_j \in C_{opt}$.

B. Cost Models

Cost models are used to estimate both the Index storage and the workload processing costs. In this paper, we use the mathematical cost models that are initially proposed in [14] and are widely used in the most closely related works [4], [7]–[9], [13], [15]. Below, we describe the cost models using the parameters summarized in Table I.

TABLE I. COST MODELS PARAMETERS

Symbol	Description
$\|A\|$	Number of tuples of a table A, the cardinality of set A, or the cardinality of the attribute A.
S_p	Disk page size in bytes.
$\|\|X\|\|$	Number of pages needed to store the table X.
S_P	Page pointer size in bytes.
$w(X)$	Size of tuples of a table X .
m	B-tree order, $m = 1 + S_p/(w(A) + S_P)$.
d	Number of bitmaps used for processing a given query.
N_r	Number of tuples accessed by a given query, $N_r = d \times \|F\|/\|A\|$

1) Index size estimation

The size in bytes of a BJI I_A built on attribute A, called size(I_A), depends on the domain cardinality of attribute A and the number of tuples in the fact table and is estimated by [14]:

$$size(I_A) = \frac{|A| \times |F|}{8} bytes. \quad (2)$$

Where, $|A|$ is the cardinality of attribute A, and $|F|$ is the number of tuples in the fact table F.

If C denotes an index configuration and $Size(C)$ represents the size of the configuration C, we have:

$$Size(C) = \sum_{j \in C} size(I_j). \quad (3)$$

2) Workload cost estimation

The number of input/output (I/O) operations to perform when processing each query of the workload is typically used as a measure of the utility of an index. The cost of processing a query Q_r in presence of an index I_A created on attribute A, called $cost_{BJI}(Q_r, I_A)$, is given by:

$$cost_{BJI}(Q_r, I_A) = log_m|A| - 1 + \frac{|A|}{m-1} + d\frac{|F|}{8S_P} + ||F||(1 - e^{-\frac{N_r}{||F||}}). \quad (4)$$

Where, m is the B-tree order, d is the number of bitmaps used to evaluate Q_r using I_A, S_P is the page pointer size in bytes, $N_r = d \times |F|/|A|$ is the number of read tuples for the query Q_r, and $||F||$ is the number of pages needed to store the fact table F.

If the generated index configuration C do not cover a given query Q_r, the query cost is augmented by the cost of joins not yet performed due to the absence of a BJI precomputing them in C. Assuming all joins are achieved using the hash-join method, and if ϕ_r denotes the set of dimension tables containing non-indexed attributes involved by query Q_r, the overall cost of hash-joins needed for answering query Q_r, called $cost_{Hash}(Q_r, \phi_r)$, can be estimated by [14] :

$$cost_{Hash}(Q_r, \phi_r) = 3 \times (||F|| + \sum_{i \in \phi_r} ||D_i||) \quad (5)$$

Bearing the above in mind, the total cost of the workload W for the generated index configuration C, called $Cost(W, C)$, is computed by:

$$Cost(W, C) = \sum_{r \in W} \sum_{j \in C} cost_{BJI}(Q_r, I_j) + \sum_{r \in W} cost_{Hash}(Q_r, \phi_r) \quad (6)$$

C. Basic concepts on constraint programming

Constraint satisfaction problems (CSPs) are a class of problems with many real-world applications. A CSP is formally described by a triplet $\langle X; D; C \rangle$, where:

- $X = \{x_1, x_2, \dots x_n\}$ is a set of decision variables,
- $D = \{d_1, d_2, \dots, d_n\}$ is a set of domains (possible values). Each variable x_i can take on values from its domain d_i.
- $C = \{c_1, c_2, \dots, c_k\}$ is a set of constraints (restrictions). Each constraint c_i involves a finite number of variables and restricts the values that this subset of variables can simultaneously take.

A solution to a CSP is the assignment of values to each of the decision variables from their domains, in such a way that every constraint in C is satisfied. It is possible to search for one solution, all solutions, an optimal, or at least a good solution, given some objective function. In the last case, the CSP is said to be a Constraint Optimization Problem (COP). More formally, a COP is a CSP of the form $\langle X; D; C; O \rangle$, where O is an objective function for ranking solutions. An optimal solution is the one which minimizes / maximizes the objective function [21].

CP is a paradigm for efficiently solving combinatorial problems, particularly CSPs and COPs. While traditionally problem resolution describes how a solution should be computed, CP employs a more declarative approach. The emphasis lies on developing high-level modelling languages and general solvers where the user describes the structure of the problem without specifying how to solve it [20].

Solving a practical problem using CP is made of two stages that are named the model stage and the resolution stage. First, the problem must be modelled as a CSP or a COP, which means it is needed to precise the definition of variables and their domains, the specification of the constraints between variables and the definition of the objective function on related variables (if the problem is a COP). Second, the resolution stage combines various algorithms to solve the problem according to its modelling. It relies on two main operations: *constraint propagation* and *search strategy*. Propagation is the process of reducing variables domains by using constraints to filter incoherent values. When constraint propagation is unable to further reduce the domains of variables, search strategies are then used to decide how to explore the search space by computing decisions. A decision involves a variable, a value and a branching strategy and triggers new constraint propagation [21], [22].

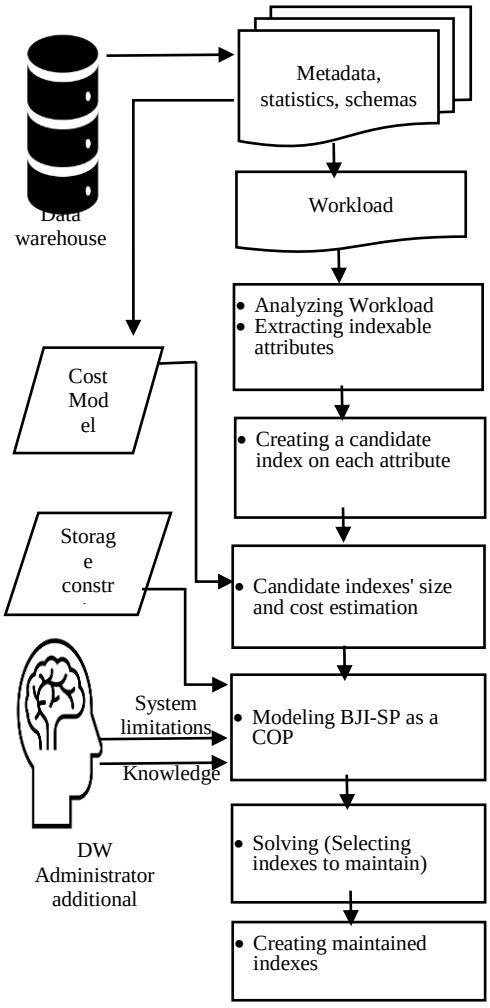
Fig. 1 Architecture of our CP-based Approach for the BJI-SP

It is worth pointing that in such strategies, the solution space is usually organized as a search tree and the shape of this tree is typically defined by the choice of the next variable to branch on, and the corresponding value assignment. This choice has a huge influence on the size of the search tree, and hence, the search effort required to solve the problem. Thus, for efficiency reasons, most recent CP solvers offer the user the possibility to determine the most convenient search strategy or even to control the search behavior by tuning the search process using generic search heuristics known as black-box search strategies. The latter, like Impact-Based Search heuristic (IBS) [23], domain over weighted degree heuristic (DomOverWDeg) [24], and Activity-Based Search heuristic (ABS) [25], are generic search heuristics provided by most of the solvers that can be readily used with any model and can perform well on a broad range of problems [21].

III. Our Approach: Modelling and Solving the BJI-SP with Constraint Programming

A. General principle of our approach

Our proposed approach (illustrated in Figure 1) exploits the information extracted from the warehouse's data (notably,



DWH schema and statistics such as attribute cardinality, and attribute selectivity) as well as DWA expertise, formalized as constraints, to recommend an index configuration improving data access time.

In the first step, the input workload is syntactically analyzed by an automatic parser to identify all attributes that might be useful for indexing, indexable attributes. The latter are those non-key dimension table attributes present in the WHERE clauses of the considered queries. After that, on each indexable attribute, a candidate BJI is built, and the cost models are used to estimate its size as well as the cost of processing each query in its presence. Subsequently, The BJI-SP is modelled as a COP by means of a set of decision variables, a set of constraints and an objective function for ranking the solutions. At this stage, many useful user-defined constraints can be easily expressed and straightforwardly integrated into the problem model by the DWA. Using these constraints can be of great help to intelligently reduce the problem search space and thus, accelerate the search process or adapt to certain constraints imposed by the system (e.g., a limited number of indexes per table). This important modelling step is detailed in the next subsection. Afterwards, a constraint solver is employed to conduct an automated search for solutions. lastly, each index from the optimal set found is implemented by executing CREATE statements (ORACLE), and workload queries are updated by inserting appropriate HINT statements.

B. CP Model for the BJI-SP

In this section, we describe our CP model for the BJI-SP. Symbols and the variables used in the formulation are presented in Table II.

TABLE II. CP BJI-SP MODEL PARAMETERS

Symbol	Description
$A = \{A_1, \ldots, A_n\}$	A set of indexable attributes
$I = \{I_1, \ldots, I_n\}$	A set of candidate indexes
$D = \{D_1, \ldots, D_d\}$	A set of dimension tables
$W = \{Q_1, \ldots, Q_q\}$	A representative set of decisional queries (Workload)
$X = \{x_1, \ldots, x_n\}$	Indexation decision variables.
$Y = \{y_{11}, \ldots, y_{qd}\}$	Dimension table loading decision variables.
$H = \{h_1, \ldots, h_q\}$	Hash-joining decision variables.
F	Fact table
$\|A\|$	Cardinality of set A
$\|\|X\|\|$	Number of pages needed to store the table X
S_{max}	Storage space reserved for the indexes to be selected.
I_i	index built on attribute A_i
$Size(I_i)$	Estimated size of index I_i built on attribute A_i
C_{qi}	The Cost of responding query Q_q using BJI I_i
$\alpha_{iq} \in \{0,1\}$	Equals to 1 if attribute A_i is involved in query Q_q.
$\beta id \in \{0,1\}$	Equals to 1 if attribute A_i belongs to the dimension table D_d.

Given an input workload W consisting of p queries $\{Q_1, \ldots, Q_p\}$, we syntactically parse each query Q_q and extract a set of indexable attributes. A candidate index is then created on each indexable attribute. Let $I = \{I_1, \ldots, I_n\}$ be the set of n candidate indexes. We denote C_{qi} the cost of processing the query Q_q using the index I_i. As explained above, the objective is to identify an index configuration $I_c \subseteq I$ that minimizes the total workload cost computed by summing up the cost of processing each query. Therefore, for each query Q_q in W, our model should make decisions on:

- The indexes to maintain, i.e., the indexes that will profit the most the query Q_q,
- The dimension tables that must be loaded to answer Q_q, when needed,
- The joins to perform, when needed.

To express these decisions, the model we propose uses three sets of binary decision variables as follows:

- X, *the indexation decision variable set*, where each variable x_i represents whether the index I_i, built on attribute A_i, will be maintained in the final index configuration ($x_i = 1$) or not ($x_i = 0$).
- Y, *the loading decision variable set*, where each variable y_{qd} indicates whether dimension table D_d is to be loaded and joined with the fact table F when answering query Q_q ($y_{qd} = 1$) or not ($y_{qd} = 0$).
- H, *the hash-joining decision variable set*, where each variable h_q indicates whether a hash-join operation is needed when answering query Q_q ($h_q = 1$) or not ($h_q = 0$).

  We recall that hash-join operations are needed when a query is not totally covered by indexes.

Furthermore, our model employs two constant Boolean matrices α and β:

- The matrix α indicates whether an attribute A_i is accessed by a query Q_q ($\alpha_{iq} = 1$) or not ($\alpha_{iq} = 0$).
- The matrix β indicates whether an attribute A_i belongs to a dimension table D_d ($\beta_{id} = 1$) or not ($\beta_{id} = 0$).

Thus, our CP model for the BJI-SP is expressed as follows:

Minimize:

$$\sum_{q=1}^{|W|}\sum_{i=1}^{|A|} x_i \times C_{qi} + 3 \times h_q \times (||F|| + \sum_{i=1}^{|D|} y_{id} \times ||D_d||) \quad (7)$$

Subject to:

$$\sum_{i=1}^{|A|} x_i \times size(I_i) \leq S_{max} \quad (8)$$

$$\%Gain = 100 \times (1 - WCost_{IConf}/WCost_{Hash}) \quad (11)$$

$$\sum_{i=1}^{|A|} x_i \times \alpha_{iq} = \sum_{i=1}^{|A|} \alpha_{iq} \Leftrightarrow h_q = 0 \quad (9)$$

$$\sum_{i=1}^{|A|} \alpha_{iq} \times \beta_{id} \times (1 - x_i) \geq 1 \Leftrightarrow y_{qd} = 1 \quad (10)$$

Where:

- The objective function in (7) aims to minimize the total cost of the query workload,
- Equation (8) expresses a knapsack constraint ensuring that the set of selected indexes consumes no more than the bound S_{max}.
- Constraint in (9) links the indexation decision variable X with the hash-joining decision variable H. It states that, for a query Q_q, a join operation is not needed (i.e, $h_q = 0$) if and only if each attribute A_i involved in Q_q (i.e, $\alpha_{iq} = 1$) is indexed i.e, ($x_i = 1$).
- Finally, the constraint in (10) states whether a dimension table D_d must be loaded to answer a query Q_q (i.e, $y_{qd} = 1$) or not. The table D_d must be loaded if and only if there exists at least one non-indexed attribute A_i (i.e, $x_i = 0$) belonging to D_d (i.e, $\beta_{id} = 1$) and A_i is involved in the query Q_q (i.e, $\alpha_{iq} = 1$).

IV. EXPERIMENTAL STUDY

A. Experimental Setup

We implemented our CP model using the Java library Choco 4.0.8[1] [22] which is a well-known free open-source library dedicated to CP. All the computational experiments have been run on an Intel(R) Core (TM) i3 CPU 8100@3.60 GHz machine with 8.0 GB RAM and running Windows 10 Pro x64 operating system.

We compared our approach's performance with that of two approaches that are best so far in the context of selecting BJIs in relational data warehouses: *(i) The Data mining (DM) based approach* and *(ii) The Genetic Algorithms (GA) based approach.*

- The DM-based approach proposed in [4] was implemented using Java Development Kit. The threshold value for this approach was set to 0.05 as suggested by the authors.
- The improved version of the GA-based approach proposed in [11] was implemented using the Java Genetic Algorithms Package API[2]. We also maintained the same parameter setup as suggested by the authors, that is, the population size, the crossover probability, the mutation rate, and the number of iterations set to 70, 0.8, 0.01 and 200 respectively.

In all experiments, the quality of the solutions found is measured in terms of percentage gain rates on the workload cost when using the resulted index configuration relative to its cost without indexes. It is given by:

where, $WCost_{IConf}$ represents the cost of the workload W using the resulting Index Configuration ($IConf$) and $WCost_{Hash}$ represents the workload cost in absence of useful indexes in $IConf$ (computed using the Hash-join method). For GA, the workload cost was computed and averaged over five independent runs because of its probabilistic behavior.

[1] http://www.choco-solver.org/
[2] http://jgap.sourceforge.net

B. Data sets

The APB-1 release II Benchmark of the OLAP Council [26] was used to generate the data warehouse and ORACLE 12c DBMS environment was used to implement the data warehouse and obtain data statistics. The APB-1 benchmark simulates a realistic On-Line Analytical Processing (OLAP) business situation. The star schema of this benchmark consists of one fact table *Actvars* (24,786,000 tuples) and four dimension-tables: *ProLevel* (9,000 tuples), *TimeLevel* (24 tuples), *CustLevel* (900 tuples), and *ChanLevel* (9 tuples). As a workload, we have considered the same workload as in [7] consisting of 60 star join queries using several selection predicates defined on one or more attributes. The workload is composed of single block queries (i.e., not nested queries) that belong to several categories: Simple SELECT queries, queries of type COUNT(*) with and without aggregations and queries using aggregation functions such as SUM, MIN, MAX and AVG.

The experiments were conducted according to two scenarios:

- *Smaller Size Problem (SSP):* in this scenario, we have syntactically analyzed the workload and extracted a set of 12 indexable attributes (i.e., non-key attributes present in the WHERE clauses of each query), namely: {*all, line, group, retailer, year, quarter, month, family, gender, division, class, city*}.
- *Larger Size Problem (LSP):* in this scenario, since the benchmark doesn't offer a significantly large number of indexable attributes, similarly to [11], the indexable attributes set was augmented to 20 by adding 8 new attributes: {*day, type, supplier, category, status, educational, marital, state*}. Furthermore, a significant workload consisting of 260 OLAP queries was considered in this case: the former SSP 60 queries plus 200 new OLAP queries generated using APB-1 benchmark queries templates.

C. Tests and results

The experiments were carried out in two steps: (i) An empirical study to measure the impact of search strategies on the performances of our approach, (ii) An evaluation of the performance of our approach against DM and GA approaches for both SSP and LSP instances of the problem. In the following, we detail the tests performed and the obtained results.

1) Impact of search strategies

The search strategy is an important factor that can drastically affect the solving process. It determines how the search tree is built by choosing, at each node of the search tree, a non-assigned variable and a value belonging to its domain. Recent CP solvers offer a number of generic, problem-independent search strategies for users to choose from. Choosing an effective one is a high problem-dependent task. The objective of this first experiment is to drive an empirical evaluation of the impact of search strategies on the effectiveness of our approach. This allows us to choose the best strategy to adopt for the rest of the experiments. To achieve this, we first assess how well our method performs in both SSP and LSP scenarios when using three good representatives of the state of the art for black-box generic search heuristics, namely: *DomOverWDeg* [24], *IBS* [23] and *ABS* [25]. Then, to further improve search heuristics performances, we associate each search heuristic with search plugins (repairing mechanisms) *LC* [27] and *COS* [28] respectively (yielding a total of 18 different cases). For each of these settings, we measure the number of visited nodes (shown as a measure of the size of the search tree), backtracks performed and time in milliseconds to find and prove the optimal solution (if it exists).

Table III shows the obtained results as the maximum storage size reserved for indexes (S_{max}) was increased systematically from 1000 MB to 2000 MB and then to 3000 MB. Note that DomOverWDeg is the default search strategy in *Choco* Solver and thus, the results obtained by this strategy are what most users would get.

TABLE III. THE IMPACT OF SEARCH STRATEGIES ON THE EFFICIENCY OF OUR APPROACH

Search Strategy	S_{max} (MB)	SSP scenario			LSP scenario		
		Time	*Nodes*	*Back-tracks*	*Time*	*Nodes*	*Back-tracks*
ABS	1000	106	372	491	171	1 225	1 597
	2000	120	309	403	285	1 384	1 737
	3000	127	476	619	296	1 662	2 036
IBS	1000	060	15	29	095	94	153
	2000	074	28	39	116	202	369
	3000	081	47	65	131	171	305
DomOver-WDeg	1000	069	41	57	099	110	191
	2000	084	71	87	120	194	339
	3000	095	59	61	178	276	381
ABS-LC	1000	108	373	493	163	1 223	1593
	2000	120	309	403	234	1 562	2 045
	3000	124	476	619	270	1 673	2 056
IBS-LC	1000	060	14	27	092	89	163
	2000	075	29	37	111	173	319
	3000	083	47	65	120	144	259
DomOver-WDeg-LC	1000	066	33	49	085	107	187
	2000	080	68	81	119	174	305
	3000	085	59	61	167	254	343
ABS-COS	1000	108	384	515	163	1 204	1 555
	2000	125	307	399	285	1 935	2 742
	3000	124	475	617	299	1 662	2 041
IBS-COS	1000	**060**	**12**	**23**	**090**	**85**	**173**
	2000	**070**	**24**	**31**	**106**	**160**	**299**
	3000	**078**	**43**	**61**	**112**	**127**	**225**
DomOver-WDeg-COS	1000	065	40	61	094	110	187
	2000	089	68	81	102	126	215
	3000	083	59	63	173	246	327

The experiments results highlight that all search strategies were able to solve both SSP and LSP scenarios to optimality in less than 300 milliseconds. The *ABS* heuristic performance was the worst in all terms (resolution time, number of nodes of tree search and number of backtracks). The default search strategy of *Choco* solver (*DomOverWDeg*) was much more efficient. Nevertheless, IBS heuristic was by far the best since the search trees were always smaller. Our experiments with the *LC* and *COS* search plugins revealed that when used on top of the three search criteria, the *LC* plugin provided better results in 55,55% of instances and equal performances in 22,22%. The *COS* heuristic enhanced performances much further, with a rate of 72,22% of improvements and 16,66% of equal performances. Based on these findings, for the rest of our experimentation, we maintained the IBS heuristic associated with *COS* repairing mechanism as the search strategy for our model.

2) Performance study and comparative analysis

In this section, we provide a performance evaluation of our approach (referred to as CPBJIS) against DM and GA based approaches for both SSP and LSP scenarios. The quality of each approach is measured in terms of its gain rate (%Gain) and its running time (in milliseconds). For the stochastic algorithm GA, the best gain rate over five independent runs (%BestGain), the average gain rate (%AvgGain), and the average execution time (AvgTime) are reported. Performance evaluations are performed under storage size constraint, where the maximal storage space S_{max} was increased from 500 MB to 6500 MB in 500 MB increments. The results of the experiments are presented in Table IV and Table V for the SSP and LSP sets respectively. In both tables, the best results are presented in a bold font and the last row provides, for each approach, an average total execution time and gain rate.

TABLE IV. CP-BJIS PERFORMANCE VS DM AND GA FOR SSP SCENARIO

S_{max} (MB)	GA			DM		CP-BJIS	
	%Best-Gain	*%Avg-Gain*	*Avg-Time*	*%Gain*	*Time*	*%Gain*	*Time*
500	**20,88**	19,34	1 235	18,32	120	**20,88**	**55**
1000	34,97	32,82	2940	34,97	144	**35,05**	**52**
1500	45,02	44,47	3110	44,48	142	**45,79**	**60**
2000	52,68	51,01	3175	45,79	176	**53,42**	**69**
2500	**60,69**	60,24	3340	58,48	176	**60,69**	**72**
3000	61,81	60,97	3715	61,81	198	**62,95**	**75**
3500	**64,06**	63,83	3550	63,60	198	**64,06**	**70**
4000	63,73	63,38	3210	63,60	183	**64,86**	**78**
4500	64,06	63,93	193320	65,49	193	**65,98**	**65**
5000	69,30	67,14	3205	65,49	193	**70,44**	**68**
5500	70,44	70,44	3680	65,49	192	**71,56**	**78**
6000	**72,36**	**72,36**	3560	65,49	193	**72,36**	**72**
6500	**73,48**	**73,48**	3320	65,49	193	**73,48**	**74**
AVG	57,96	57,19	3181	55,27	177	**58,58**	**68**

Results of the SSP scenario, presented in Table IV, showed that whatever the value of the storage space S_{max}, CP-BJIS outperforms GA and DM approaches for both measures (execution time and gain rate). The CP-BJIS approach computes solutions in average 2.6 times faster than DM and 46.77 times faster than GA. In terms of gain rates, as the space allocated for indexes grows, all algorithms converge towards selecting indexes with better performances, except for the DM algorithm that stops converging when S_{max} exceeds 4500 MB. Moreover, this algorithm was less efficient and failed to reach any best solution. This might be due to the additional minimal frequency constraint of this approach. Indeed, even when the storage space is not too tight, this constraint may result in missing some non-frequent, but useful, candidate indexes that could favour the convergence towards an optimal solution. On the other hand, GA performs better than the DM method (even though the latter was almost 18 times faster), it generates the best solutions 5 times out of 13 (in its best performances %BestGain) and its average performance rates (%AvgGain) were close to the best ones (obtained via the CP-BJIS). However, in all the cases, CP-BJIS remains the best one with 100% of best solutions found in the shortest computing time.

TABLE V. CP-BJIS PERFORMANCE VS DM AND GA FOR LSP SCENARIO

S_{max} (MB)	GA			DM		CP-BJIS	
	%Best-Gain	*%Avg-Gain*	*Avg-Time*	*%Gain*	*Time*	*%Gain*	*Time*
500	**22,58**	22,51	8733	**22,58**	280	**22,58**	**74**
1000	40,00	38,09	9538	40,00	301	**40,80**	**96**
1500	47,36	46,21	10274	45,97	320	**48,36**	**103**
2000	53,40	51,14	9713	51,35	385	**53,55**	**101**
2500	**57,49**	55,82	10115	53,59	371	**57,49**	**107**
3000	59,93	59,49	10589	59,48	412	**60,60**	**111**
3500	**63,07**	62,08	10972	62,25	407	**63,07**	**137**
4000	65,12	64,16	11365	64,48	435	**65,43**	**149**
4500	67,42	66,67	11021	66,27	359	**67,66**	**155**
5000	69,93	69,46	12274	67,22	322	**70,28**	**150**
5500	72,13	71,84	11713	67,22	330	**72,88**	**128**
6000	**74,63**	74,17	10115	71,44	321	**74,63**	**130**
6500	75,58	75,46	10589	71,44	319	**75,79**	**124**
AVG	59,13	58,25	10539,31	57,18	350,92	**59,47**	**120,38**

Results of the LSP scenario, presented in Table V, showed slight improvements (less than 2%) in the average gain rates of all approaches compared to those obtained in the SSP scenario. By contrast, the average run time needed to obtain these performances is significantly higher (3.31 times higher for GA, 1.98 for DM and 1.77 for CP-BJIS). This increase in run time is due to the increase in the number of candidate indexes which went from 12 to 20. Comparing the average computation time of the three methods for this larger workload, CP-BJIS was 2.91 times and 87.82 times faster than the DM and GA approaches respectively. These results showed that our approach is likely to scale better than the two other approaches when increasing the number of indexable attributes and the size of the workload. In terms of solutions' quality, CP-BJIS was able to generate the best solutions in all the cases (13 out of

13), GA generated 4 best solutions (in its best performances %BestGain) and only one best solution was found by the DM algorithm. In summary, as also mentioned for the SSP scenario, the CP-BJIS approach has again considerably outperformed both GA and DM approaches in all aspects; execution time, gain rates and the number of best solutions found.

V. CONCLUSIONS AND PERSPECTIVES

The ISP has always been a focal issue in database design considering its potential huge search space. The difficulty of solving this problem illustrates the limitations of existing approaches which rely, often, on heuristics and meta-heuristics trying to provide the best trade-off between the quality of solutions and the execution time. Despite being practical, however, these approaches do not guarantee to find the optimal solution and to determine their best set-up parameters is highly challenging.

In this paper, we have proposed a new exact approach to address the ISP in the DW environment. More precisely, we have provided an innovative alternative to state-of-the-art approaches by using the CP paradigm to model and solve the problem. Thus, we have considered the ISP as a COP, and a corresponding CP model was specified and implemented using Choco, a well-known open-source CP solver. The resolution process is then automatically supported by the solver.

The effectiveness and efficiency of our approach have been validated on APB-I, a fairly large benchmark data warehouse, over several experiments. At first, we conducted an empirical study to choose the best search strategy, for our model, from the state-of-the-art generic search heuristics. Obtained results show that our model performs better with the IBS search heuristic associated with COS repairing mechanism. Afterwards, we performed a comparative study against two well-known approaches, the GA and the DM. Considering smaller and larger instances of the ISP, the experimental results confirm the superior performance of our approach for both cases. Our approach achieves more accurate solutions with a CPU time that is much shorter than its two other competitors.

Overall, the outstanding performance of our proposed approach indicates that CP is a promising technology for the ISP. It can prove optimality for much larger test instances, and thus, it can be used not only as a physical design framework expandable on user constraints, but also as a benchmark to measure other approaches' solutions quality.

As future work, we plan to extend our model to the selection of multi-attribute indexes. Another interesting direction consists in accelerating the search process by developing a custom search strategy dedicated to the ISP or by integrating approximate search strategies such as meta-heuristics with Large Neighborhood Search.

REFERENCES

[1] L. Toumi and A. Ugur, “Static and incremental dynamic approaches for multi-objective bitmap join indexes selection in data warehouses,” *Journal of Supercomputing*, vol. 77, no. 4, pp. 3933–3958, Apr. 2021, doi: 10.1007/s11227-020-03423-7.

[2] R. Kain, D. Manerba, and R. Tadei, “The index selection problem with configurations and memory limitation: A scatter search approach,” *Comput Oper Res*, vol. 133, Sep. 2021, doi: 10.1016/j.cor.2021.105385.

[3] S. Amjad, I. Jellouli, M. Yahyaoui, and L. Benameur, “Efficient of bitmap join indexes for optimising star join queries in relational data warehouses,” *International Journal of Computational Intelligence Studies*, vol. 9, no. 3, 2020, doi: 10.1504/ijcistudies.2020.10031847.

[4] B. Ziani and Y. Ouinten, “An improved approach for automatic selection of multi-tables indexes in ralational data warehouses using maximal frequent itemsets,” *Intelligent Decision Technologies*, vol. 7, no. 4, 2013, doi: 10.3233/IDT-130169.

[5] E. O’Neil, P. O’Neil, and K. Wu, “Bitmap index design choices and their performance implications,” 2007. doi: 10.1109/IDEAS.2007.4318091.

[6] S. Chaudhuri, M. Datar, and V. Narasayya, “Index selection for databases: A hardness study and a principled heuristic solution,” *IEEE Trans Knowl Data Eng*, vol. 16, no. 11, 2004, doi: 10.1109/TKDE.2004.75.

[7] L. Bellatreche and K. Boukhalfa, “Yet another algorithms for selecting bitmap join indexes,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 6263 LNCS. doi: 10.1007/978-3-642-15105-7_9.

[8] H. Drias and I. Frihi, “ACO based approach and integrating information retrieval technologies in selecting Bitmap Join Indexes,” in *Proceedings - 2010 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2010*, 2010, vol. 1. doi: 10.1109/WI-IAT.2010.180.

[9] A. Gacem and K. Boukhalfa, “Immune algorithm for bitmap join indexes,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012, vol. 7665 LNCS, no. PART 3. doi: 10.1007/978-3-642-34487-9_68.

[10] R. Bouchakri and L. Bellatreche, “On simplifying integrated physical database design,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011, vol. 6909 LNCS. doi: 10.1007/978-3-642-23737-9_24.

[11] L. Toumi, A. Moussaoui, and A. Ugur, “Particle swarm optimization for bitmap join indexes selection problem in data warehouses,” *Journal of Supercomputing*, vol. 68, no. 2, 2014, doi: 10.1007/s11227-013-1058-9.

[12] P. Ameri, J. Meyer, and A. Streit, “On a new approach to the index selection problem using mining algorithms,” 2015. doi: 10.1109/BigData.2015.7364084.

[13] K. Aouiche and J. Darmont, “Data mining-based materialized view and index selection in data warehouses,” *J Intell Inf Syst*, vol. 33, no. 1, 2009, doi: 10.1007/s10844-009-0080-0.

[14] K. Aouiche, J. Darmont, O. Boussaïd, and F. Bentayeb, “Automatic selection of bitmap join indexes in data warehouses,” in *Lecture Notes in Computer Science*, 2005, vol. 3589. doi: 10.1007/11546849_7.

[15] L. Bellatreche, R. Missaoui, H. Necir, and H. Drias, “A Data Mining Approach for Selecting Bitmap Join Indices,” *Journal of Computing Science and Engineering*, vol. 1, no. 2, 2007, doi: 10.5626/jcse.2007.1.2.177.

[16] P. Kołaczkowski and H. Rybiński, “Automatic index selection in RDBMS by exploring query execution plan space,” *Studies in Computational Intelligence*, vol. 223, 2009, doi: 10.1007/978-3-642-02190-9_1.

[17] R. Schlosser, J. Kossmann, and M. Boissier, “Efficient scalable multi-attribute index selection using recursive strategies,” in *Proceedings - International Conference on Data Engineering*, 2019, vol. 2019-April. doi: 10.1109/ICDE.2019.00113.

[18] E. C. Freuder, “Progress towards the Holy Grail,” *Constraints*, vol. 23, no. 2, 2018, doi: 10.1007/s10601-017-9275-0.

[19] I. Mami, R. Coletta, and Z. Bellahsene, "Modeling view selection as a constraint satisfaction problem," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011, vol. 6861 LNCS, no. PART 2. doi: 10.1007/978-3-642-23091-2_33.

[20] E. C. Freuder, "Constraints: The ties that bind," in *Proceedings of the National Conference on Artificial Intelligence*, 2006, vol. 2.

[21] J. G. Fages and C. Prud'Homme, "Making the first solution good!," in *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, 2018, vol. 2017-November. doi: 10.1109/ICTAI.2017.00164.

[22] C. Prud'homme, J.-G. Fages, and X. Lorca, "Choco Documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING SAS (2016)," *URL http://www. choco-solver. org*, 2019.

[23] P. Refalo, "Impact-based search strategies for constraint programming," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3258, 2004, doi: 10.1007/978-3-540-30201-8_41.

[24] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais, "Boosting systematic search by weighting constraints," in *Frontiers in Artificial Intelligence and Applications*, 2004, vol. 110.

[25] L. Michel and P. van Hentenryck, "Activity-based search for black-box constraint programming solvers," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012, vol. 7298 LNCS. doi: 10.1007/978-3-642-29828-8_15.

[26] O. Council, "Apb-1 olap benchmark, release ii." 1998. Accessed: May 07, 2022. [Online]. Available: http://www.olapcouncil.org

[27] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal, "Last conflict based reasoning," *Frontiers in Artificial Intelligence and Applications*, vol. 141, 2006.

[28] S. Gay, R. Hartert, C. Lecoutre, and P. Schaus, "Conflict ordering search for scheduling problems," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015, vol. 9255. doi: 10.1007/978-3-319-23219-5_10.