Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 52 (2015) 169 – 177

Procedia Computer Science

6th International Conference on Ambient Systems, Networks and Technologies, ANT 2015

A linear programming approach for bitmap join indexes selection in data warehouses

Lyazid TOUMI[a,*], Abdelouahab MOUSSAOUI[a], Ahmet UGUR[b]

[a]*Intelligent Systems Laboratory (LSI), Computer Science Department, Sciences Faculty, Ferhat ABBAS University of Setif-1, 19000 Setif, Algeria*
[b]*Computer Science Department, Central Michigan University, 48859 Mount Pleasant, MI, USA*

Abstract

Data warehousing is the crucial part of business intelligence applications. The data warehouse physical design is a hard task due to a large number of possible choices involved. The bitmap join indexes selection problem is crucial in the data warehouse physical design. All proposed approaches to solve the bitmap join indexes selection problem are based on statistics such as data mining or meta-heuristics such as genetic algorithm and particle swarm optimization. In the present work, we propose a new approach based on mixed-integer linear programming for solving the bitmap join indexes selection problem. Several experiments are performed to demonstrate the effectiveness of the proposed approach and the results are compared to the well known approaches that are best so far: the data mining, the genetic algorithm and particle swarm optimization based approaches. The mixed-integer linear programming is found to be faster and more effective than the genetic algorithm, particle swarm optimization and data mining approaches for solving the bitmap join indexes selection problem.

Peer-review under responsibility of the Conference Program Chairs

Keywords: Data warehouse, Query optimization, Bitmap join index, Bitmap join indexes selection problem, linear programming;

1. Introduction

The data warehouses are the main resources for the business intelligence applications in order to make effective decisions. A data warehouse ($\mathcal{DW}$) can be implemented using an available database management system ($\mathcal{DBMS}$) such as Oracle, Microsoft SQLServer and IBM DB2. The increase in the data volume in a $\mathcal{DW}$ influences the cost of data warehouse administration and degrades the performance. The techniques proposed in the classical relational databases based on the join algorithms such as hash join, merge join and nested loop join have been shown their limits due to the complexity of the query workload executed on $\mathcal{DW}$[1]. The single table indexation techniques used in classical relational databases, such as B-Tree, hash and bitmap indexes are limited in the context of $\mathcal{DW}$[2,3,4]. Bitmap join index ($\mathcal{BJI}$) proposed by O'Neil et al. allows multiple tables indexation[5]. The $\mathcal{BJI}$ allows pre-calculation of joins between one or more tables[5,2]. Two variants of bitmap join indexes exist: bitmap join indexes on single non-key attribute ($\mathcal{SBJI}$) and bitmap join indexes created on multiple non-key attributes ($\mathcal{MBJI}$).

* Corresponding author. Tel.: +213-670-060356. Fax: +213-36-842030
E-mail address: lyazid.toumi@univ-setif.dz

Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2015.05.052

The index selection problem ($\mathcal{ISP}$) in database physical design is to select a configuration of indexes to be materialized to minimize the query workload cost. The $\mathcal{ISP}$ is a crucial problem in the physical design[6]. The $\mathcal{ISP}$ are widely tackled in the literature both in traditional and distributed databases[7,8,3,6,9,10]. The bitmap join index selection problem ($\mathcal{BJISP}$) is more difficult than $\mathcal{ISP}$ and known to be NP-hard[11]. Two variants of $\mathcal{BJISP}$ exists. The first one is to select a subset of candidate $\mathcal{SBJI}$ called ($\mathcal{SBJISP}$) and the second one is to select a subset of candidate $\mathcal{MBJI}$ called ($\mathcal{MBJISP}$).

The $\mathcal{SBJISP}$ deals with 2^n-1 possibilities and the $\mathcal{MBJISP}$ deals with $2^{2^n}-1$ possibilities, n being the number of non-key attributes. Several approaches to solve $\mathcal{BJISP}$ in both forms exists: data mining techniques ($\mathcal{DM}$)[11,12] and meta-heuristic methods, such as genetic algorithm ($\mathcal{GA}$)[13,14], binary particle swarm optimization ($\mathcal{BPSO}$)[14], and artificial immune system ($\mathcal{AIS}$)[15].

The above approaches are based on statistics or meta-heuristics. In the present work, we propose a mixed-integer linear programming ($\mathcal{MILP}$) formulation of the $\mathcal{SBJISP}$ to obtain an optimal solution. An internal bitmap is utilized for accurately incorporating the cost of joins involved into the model. The integer linear programming ($\mathcal{ILP}$) has been used to solve the $\mathcal{ISP}$ in classical databases and found to be effective for obtaining higher quality solutions[9,10,16]. The $\mathcal{MILP}$ can be solved using a commercial $\mathcal{MILP}$ solver (such as CPLEX 10.1). Several experiments were performed to demonstrate the effectiveness and advantages of the $\mathcal{MILP}$ approach and compared to the two well known methods that are best so far: the data mining based approach ($\mathcal{DM}$), the genetic algorithm ($\mathcal{GAI}$) based and particle swarm optimization ($\mathcal{BPSO}$) based approaches.

The rest of this paper is organized as follows: the problem statement is presented in Section 2, the formulation using the mixed-integer linear programming approach is described in Section 3, experimental results are presented in Section 4, and conclusions are presented in Section 5.

2. Problem statement

The single bitmap join indexes selection problem ($\mathcal{SBJISP}$) is formalized as follows[11,12]:

- $\mathcal{DW}$ with a set of dimension tables $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_m\}$ and a fact table $\mathcal{F}$.
- Query workload $\mathcal{Q} = \{\mathcal{Q}_1, \mathcal{Q}_2, ..., \mathcal{Q}_r\}$ defined on the $\mathcal{DW}$ schema.
- The set of candidate non-key dimension attributes $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_k\}$ extracted from $\mathcal{Q}$.
- The storage space constraint $\mathcal{S}$.

The problem is to identify a configuration of indexes $C = \{\mathcal{SBJI}_1, \mathcal{SBJI}_2, ..., \mathcal{SBJI}_n\}$ defined on non-key attributes in $\mathcal{A}$ such that the global cost of the query workload $GlobalCost(\mathcal{Q}, C)$ is minimized and the storage constraint $\mathcal{S}$ is satisfied.

2.1. Cost model

The cost model is used to estimate $GlobalCost(\mathcal{Q}, C)$ measured with the input/output ($\mathcal{I}/\mathcal{O}$) operations needed for the execution of queries in the query workload $\mathcal{Q}$. The global cost function $GlobalCost(\mathcal{Q}, C)$ is defined as follow:

$$\sum_{r\in Q}\sum_{k\in C} CostIndx(\mathcal{Q}_r, \mathcal{SBJI}_k) + \sum_{r\in Q} CostJoin(\mathcal{Q}_r, \phi_r) \tag{1}$$

$CostIndx(\mathcal{Q}_r, \mathcal{SBJI}_k)$ is the cost to execute a query $\mathcal{Q}_r$ in presence of useful $\mathcal{SBJI}_k$ in the configuration C, the $CostIndx(\mathcal{Q}_r, \mathcal{SBJI}_k)$ is defined as follow:

$$\log_n |\mathcal{A}_k| - 1 + \frac{|\mathcal{A}_k|}{n-1} + \mathcal{B}\frac{|\mathcal{F}|}{8\mathcal{P}} + |\mathcal{F}|(1 - e^{-\frac{\mathcal{R}}{|\mathcal{F}|}}) \tag{2}$$

where $|\mathcal{F}|$ is the number of tuples in the fact table $\mathcal{F}$, $|\mathcal{A}_k|$ is the cardinality of the domain of attribute $\mathcal{A}_k$, $\mathcal{B}$ is the number of bitmaps used to evaluate a given query, $\mathcal{P}$ is the size of disk pages measured in bytes, $\mathcal{R}$ is the number of read tuples for a given query using $\mathcal{SBJI}$ and finally n is the order of the B-tree.

$CostJoin(Q_r, \phi_r)$ is the execution cost of the query Q_r in the absence of useful indexes in the configuration C, where $\phi_r \subseteq \mathcal{D}$ represents the set of dimension tables containing attributes of Q_r without useful indexes in the configuration C[11].

All joins performed in $CostJoin$ are implemented by the hash-join method. The number of $\mathcal{I}/\mathcal{O}$ operations needed to join two tables $\mathcal{T}_1$ and $\mathcal{T}_2$ using hash-join method is given by (see[1]):

$$3 \times (\|\mathcal{T}_1\| + \|\mathcal{T}_2\|) \tag{3}$$

where $\|\mathcal{T}\|$ is a number of pages needed to store table $\mathcal{T}$. The order of joins is important when joining dimension tables in ϕ_r with the fact table $\mathcal{F}$. We have assumed that the join order is performed with the minimum selectivity method[17].

$$Size(\mathcal{SBJI}_k) = (\frac{|\mathcal{A}_k|}{8} + 16) \times |\mathcal{F}| \tag{4}$$

The storage space required to store $\mathcal{SBJI}_k$ is defined as $Size(\mathcal{SBJI}_k)$, where the size depends on the domain cardinality of $\mathcal{A}_k$ and the number of tuples in the fact table[11], and given by Eq.(4).

3. Problem formulation

We describe a mixed-integer linear ($\mathcal{MILP}$) formulation to find an optimal single bitmap join index ($\mathcal{SBJI}$) configuration. Notations used in the model are presented in Table 1 followed by the constraints and objective function.

Table 1. Model notations.

	Notation	Definition
Sets	$\mathcal{A}$	Candidate attributes for indexation
	$\mathcal{D}$	Dimension tables
	$\mathcal{Q}$	Queries in the query workload
	$\mathcal{W}$	Join combination, which is equal to $2^{\mathcal{D}} - 2$
Indices	k	$k \in \{1, 2, ..., \|\mathcal{A}\|\}$
	i	$i \in \{1, 2, ..., \|\mathcal{D}\|\}$
	r	$r \in \{1, 2, ..., \|\mathcal{Q}\|\}$
	w	$w \in \{1, 2, ..., \|\mathcal{W}\|\}$
Constants	$a_{i,k} \in \{0, 1\}$	Equals to 1 if attribute k is part of dimension table i
	$b_{r,k} \in \{0, 1\}$	Equals to 1 if attribute k itself is accessed by query r
	$\lambda_{w,i} \in \{0, 1\}$	Equals to 1 if dimension table i is in the set w (see Algorithm in Fig).
	$c_{r,k}$	The execution cost of query r in presence of index built on attribute k
	co_w	The execution join cost between dimension tables in w and the fact table F
	s_k	The storage cost of the $\mathcal{SBJI}$ built on the non-key attribute x_k
	S	Maximum storage space allowed to store index configuration
Decision Variables	$x_k \in \{0, 1\}$	Equals to 1 if $\mathcal{SBJI}$ is created on attribute k.
	$\alpha_{r,i} \in \{0, 1\}$	Equals to 1 if table i is loaded to answer the query r.
	$\beta_{r,w} \in \{0, 1\}$	Equals to 1 if the join operations w is needed to answer the query r.
	γ_r	Denote the execution cost of query r in absence of useful $\mathcal{SBJI}$.

3.1. SBJISP constraints

The constraints for the model is given below:

$$\alpha_{r,i} \geq b_{r,k}(1 - x_k)a_{i,k} \qquad \forall r \in \{1,..,|Q|\},\ \forall i \in \{1,..,|\mathcal{D}|\},\ \forall k \in \{1,..,|\mathcal{A}|\} \tag{5}$$

$$\beta_{r,w} \geq 1 + \sum_{i=1}^{|\mathcal{D}|} \lambda_{w,i}(\alpha_{r,i} - 1) \quad \forall r \in \{1,..,|Q|\},\ \forall w \in \{1,..,|\mathcal{W}|\} \tag{6}$$

$$\gamma_r \geq \beta_{r,w} co_w \qquad \forall r \in \{1,..,|Q|\},\ \forall w \in \{1,..,|\mathcal{W}|\} \tag{7}$$

$$\sum_{k=1}^{\mathcal{A}} s_k x_k \leq S \tag{8}$$

$$x_k \in \{0,1\} \qquad \forall k \in \{1,..,|A|\} \tag{9}$$

$$\alpha_{r,i} \in \{0,1\} \qquad \forall r \in \{1,..,|Q|\}, \forall i \in \{1,..,|\mathcal{D}|\} \tag{10}$$

$$\beta_{r,w} \in \{0,1\} \qquad \forall r \in \{1,..,|Q|\}; \forall w \in \{1,..,|\mathcal{W}|\} \tag{11}$$

$$\gamma_r \geq 0 \qquad \forall r \in \{1,..,|Q|\} \tag{12}$$

The cost to execute the query Q_r that uses attribute $\mathcal{A}_k$ is $c_{r,k}x_k$ if the $\mathcal{SBJI}_k$ exist. Otherwise, in the absence of the $\mathcal{SBJI}_k$, the Eq.(5) is used to identify dimension table $\mathcal{D}_i$ containing attribute $\mathcal{A}_k$ to be loaded for answering Q_r. In order to answer Q_r, all identified dimension tables by Eq.(6) are joined with the fact table using hash-join method. The problem here is how to formulate the joining operations between identified dimension tables and the fact table. All possible join combinations of dimension tables need to be considered. To address this problem, a bitmap table called λ is generated using the algorithm presented in Fig. 3. The λ bitmap table simply represents the power set of dimension tables excluding the empty set and contains $2^{|\mathcal{D}|} - 1$ rows and $|\mathcal{D}|$ columns, where each row (a bitmap) indicates dimension tables to be joined with the fact table. The dimension tables are listed in the minimum selectivity order. For example, in Fig. 4, $\mathcal{D}_2$, $\mathcal{D}_1$, $\mathcal{D}_3$ is the minimum selectivity order assumed and the row 2 indicates a join between dimension table $\mathcal{D}_1$ and the fact table $\mathcal{F}$, the row 3 indicates a join between dimension tables $\mathcal{D}_1$, $\mathcal{D}_3$ and the fact table $\mathcal{F}$. The cost of join(s) corresponding to a row in the bitmap table λ, co_w, is computed by Eq.(3). The Eq.(7) is used for computing the cost of Q_r in the absence of useful indexes in the index configuration to be selected. The Eq.(8) is the knapsack constraint that controls the size of the index configuration.

Require: $\mathcal{D}$, the set of dimension tables
1: $P \leftarrow 2^{|\mathcal{D}|} - 1$
2: **for** $w \leftarrow 1$ to P **do**
3: $t \leftarrow w$
4: $i \leftarrow 1$
5: **repeat**
6: $\lambda_{w,i} \leftarrow t$ mod 2
7: $t \leftarrow t$ div 2
8: $i \leftarrow i + 1$
9: **until** $i > |\mathcal{D}|$
10: **end for**

Fig. 1. Algorithm to generate the bitmap table λ.

w	D_2	D_1	D_3
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Fig. 2. An example of bitmap table λ.

3.2. SBJISP objective Function

The objective is to minimize the query workload cost, the objective function presented in Eq.13 is similar to the cost model represented in Section 2.1.

$$min : \sum_{r=1}^{Q}(\gamma_r + \sum_{k=1}^{A} c_{r,k}x_k) \tag{13}$$

4. Experimental Results

4.1. Problem instances

The benchmark APB-I is used for the data warehouse generation [18] and ORACLE 11g DBMS environment is used for the implementation. In this benchmark, the star schema contains four dimension tables: CHANLEVEL (9 tuples), CUSTLEVEL (900 tuples), PRODLEVEL (9,000 tuples), TIMELEVEL (24 tuples) and the fact table ACTVARS (24,786,000 tuples). Two classes of experiments are performed [14]:

- **the class of moderate size problem set** ($\mathcal{CMP}$) that contains 250 $\mathcal{OLAP}$ queries and 16 non-key attributes from dimension tables: {division, line, family, group, class, status, year, quarter, month, day, state, city, retailer, type, gender, all} with cardinalities: 4, 15, 75, 300, 605, 5, 2, 4, 12, 5, 45, 255, 99, 10, 2 and 9 respectively.
- **the class of larger size problem set** ($\mathcal{CLP}$) that contains 500 $\mathcal{OLAP}$ queries and 20 non-key attributes from dimension tables: {all, year, retailer, quarter, month, day, line, group, family, division, class, gender, city, state, type, educational, marital, supplier, status, category} with cardinalities: 9, 2, 99, 4, 12, 5, 15, 300, 75, 4, 605, 2, 255, 45, 10, 6, 4, 15, 5 and 3 respectively.

4.2. Performance Study

A set of experiments were used to analyze the efficiency of the $\mathcal{MILP}$ approach against the well known approaches that are best so far: the improved genetic algorithm ($\mathcal{GAI}$), particle swarm optimization ($\mathcal{BPSO}$) [14] and data mining approach ($\mathcal{DM}$) [11,12] using the two problem sets $\mathcal{CMP}$ and $\mathcal{CLP}$ mentioned above. All tests are performed under Intel i7 (4 cores) processor with 8 GB RAM. The IBM CPLEX[1] 10.1 solver under Java Development Kit is used for solving the proposed model, the $\mathcal{MILP}$ model of $\mathcal{SBJISP}$. The CPLEX parameters remain at their default settings. For the GAI, the population size, crossover probability, mutation rate and number of iterations were set to 70, 0.8, 0.01 and 200 respectively yielding 14,000 maximum evaluations (200×70) [14]. For the *BPSO*, the parameters c_1, c_2, V_{max}, w_{max}, w_{min} were set to 2.0, 2.0, 6.0, 0.95 and 0.5 respectively. The population size and the number of iterations were set to 30 and 200 respectively yielding 6000 maximum evaluations (200×30) [14]. For the *DM* approach, the best minimum support on the both problem sets $\mathcal{CMP}$ and $\mathcal{CLP}$ was set to 0.14 (except for S=500 and S=800 was set to 0.20) and 0.12 respectively [14].

Both $\mathcal{MILP}$ and $\mathcal{DM}$ method runs on a single problem set, therefore is not considered to be stochastic in that sense. For stochastic algorithms $\mathcal{BPSO}$ and $\mathcal{GAI}$, the average cost of solutions over five independent runs are reported. The storage size S was systematically increased from 500 MB to 2000 MB in 100 MB increments, yielding a total of 16 different cases. Therefore, a total of 80 (16×5) runs is under consideration for algorithms $\mathcal{BPSO}$ and $\mathcal{GAI}$, and 16 (16×1) runs is for $\mathcal{DM}$ and $\mathcal{MILP}$.

Tables 2 and 3 show the number of disk page accesses needed ($\mathcal{I/O}$ costs) using the cost model presented in Section 2.1 in order to execute the query workload for the problem sets $\mathcal{CMP}$ and $\mathcal{CLP}$ respectively for sixteen different storage sizes. The column Best represents the value of the best minimum cost solution found. The column Avg represents the average cost of solutions found over five independent runs for the $\mathcal{BPSO}$ and $\mathcal{GAI}$ approaches [14]. The column Evals represents the average of the number of candidate evaluations performed for reaching the best solution for five independent runs (i.e., a measure about how fast an algorithm finds an optimal/sub-optimal solution or converges for $\mathcal{BPSO}$ and $\mathcal{GAI}$). The column Time shows the average computation time in seconds. The last row provides an overall average for the runs. In each table row, the best (i.e., the minimum) and the average querying performance result were presented in bold font for each of the algorithms considered. The last entry $\mathcal{WOI}$ represents the cost of the query workload without using any index configuration (in this case, the hash-join method was used - see Section 2.1)

[1] http://www-01.ibm.com/software/info/ilog/

Table 2. Querying performance results for the moderate size problem set $\mathcal{CMP}$.

$\mathcal{S}$	$\mathcal{BPSO}$				$\mathcal{GAI}$				$\mathcal{DM}$		$\mathcal{MILP}$	
(MB)	Best	Avg	Evals	Time	Best	Avg	Evals	Time	Best	Time	Best	Time
500	**57,221,545.8**	**57,221,545.8**	342	259	**57,221,545.8**	**57,221,545.8**	6,258	2,502	83,189,106.4	32	**57,221,545.8**	5
600	**57,221,545.8**	**57,221,545.8**	366	260	**57,221,545.8**	**57,221,545.8**	3,066	1,246	62,983,169.8	37	**57,221,545.8**	5
700	**57,221,545.8**	**57,221,545.8**	390	278	**57,221,545.8**	57,353,505.8	3,682	2,622	62,983,169.8	37	**57,221,545.8**	5
800	**57,221,545.8**	**57,221,545.8**	558	403	**57,221,545.8**	57,353,505.8	6,230	3,517	62,540,998.0	41	**57,221,545.8**	5
900	**57,221,545.8**	**57,221,545.8**	342	245	**57,221,545.8**	70,276,181.0	7,420	4,381	62,540,998.0	41	**57,221,545.8**	8
1,000	**52,571,745.5**	53,444,328.5	1,704	1,405	**52,571,745.5**	54,995,112.6	4,760	4,559	62,540,998.0	41	**52,571,745.5**	5
1,100	**52,571,745.5**	**52,571,745.5**	936	684	**52,571,745.5**	55,246,871.6	9,002	6,035	62,540,998.0	41	**52,571,745.5**	9
1,200	**52,571,745.5**	**52,571,745.5**	684	513	**52,571,745.5**	54,431,665.6	7,546	4,295	57,549,135.1	41	**52,571,745.5**	11
1,300	**52,571,745.5**	**52,571,745.5**	480	368	**52,571,745.5**	53,501,705.6	4,760	3,284	57,549,135.1	42	**52,571,745.5**	19
1,400	**52,571,745.5**	**52,571,745.5**	942	715	**52,571,745.5**	53,598,060.1	6,944	4,269	57,549,135.1	42	**52,571,745.5**	28
1,500	**52,571,745.5**	**52,571,745.5**	852	654	**52,571,745.5**	52,827,584.9	6,832	3,649	57,549,135.1	42	**52,571,745.5**	569
1,600	**52,571,745.5**	**52,571,745.5**	414	336	**52,571,745.5**	53,489,226.2	5,236	3,767	57,549,135.1	42	**52,571,745.5**	63
1,700	**51,631,175.4**	51,819,289.4	1,602	2,123	**51,631,175.4**	53,692,102.3	7,672	4,905	57,549,135.1	42	**51,631,175.4**	1,804
1,800	**39,136,645.4**	**39,136,645.4**	1,236	969	**39,136,645.4**	50,335,991.1	7,098	5,724	57,549,135.1	42	**39,136,645.4**	5
1,900	**39,136,645.4**	**39,136,645.4**	498	386	**39,136,645.4**	41,635,551.4	7,420	3,981	57,549,135.1	42	**39,136,645.4**	6
2,000	**39,136,645.4**	**39,136,645.4**	360	283	**39,136,645.4**	**39,136,645.4**	5,264	2,396	44,622,415.0	42	**39,136,645.4**	12
Avg	51,446,941.2	51,513,234.8	731.6	617.4	51,446,941.2	53,894,800.1	6,199.4	3,820.7	60,270,933.4	40.4	51,446,941.2	159.9
$\mathcal{WOI}$	251,523,278.2											

Table 3. Querying performance results for the larger size problem set $\mathcal{CLP}$.

$\mathcal{S}$	$\mathcal{BPSO}$				$\mathcal{GAI}$				$\mathcal{DM}$		$\mathcal{MILP}$	
(MB)	Best	Avg	Evals	Time	Best	Avg	Evals	Time	Best	Time	Best	Time
500	**120,162,008.1**	**120,162,008.1**	504	633	**120,162,008.1**	**120,162,008.1**	6,454	4,407	224,983,173.4	123	**120,162,008.1**	9
600	**120,162,008.1**	**120,162,008.1**	444	558	**120,162,008.1**	121,437,241.8	6,454	6,172	164,811,723.9	135	**120,162,008.1**	8
700	**120,162,008.1**	**120,162,008.1**	492	606	**120,162,008.1**	**120,162,008.1**	5,723	4,008	164,811,723.9	128	**120,162,008.1**	8
800	**120,162,008.1**	**120,162,008.1**	408	514	**120,162,008.1**	**120,162,008.1**	8,330	5,932	161,576,842.4	125	**120,162,008.1**	10
900	**120,162,008.1**	**120,162,008.1**	516	674	**120,162,008.1**	122,314,119.8	9,674	10,276	161,576,842.4	126	**120,162,008.1**	20
1,000	**120,162,008.1**	**120,162,008.1**	492	653	**120,162,008.1**	121,596,749.2	5,348	6,279	161,576,842.4	127	**120,162,008.1**	247
1,100	**120,162,008.1**	121,033,810.5	498	3,646	**120,162,008.1**	122,087,017.3	5,782	6,812	149,388,384.0	124	**120,162,008.1**	215
1,200	**112,513,169.4**	**112,513,169.4**	1,566	2,021	116,474,246.7	119,424,455.8	6,706	10,139	149,388,384.0	126	**112,513,169.4**	27
1,300	**112,513,169.4**	118,632,240.4	576	6,211	**112,513,169.4**	118,021,796.9	8,050	10,154	149,388,384.0	127	**112,513,169.4**	72
1,400	**112,513,169.4**	114,042,937.2	462	2,128	**112,513,169.4**	118,632,240.4	7,700	9,019	149,388,384.0	123	**112,513,169.4**	872
1,500	**112,513,169.4**	114,042,937.2	528	2,141	**112,513,169.4**	117,102,472.6	6,552	8,613	149,388,384.0	123	**112,513,169.4**	1,331
1,600	**112,513,169.4**	115,572,704.9	1,152	4,363	**112,513,169.4**	118,632,240.4	8,372	9,777	149,388,384.0	122	**112,513,169.4**	1,619
1,700	**112,513,169.4**	115,572,704.9	918	3,998	**112,513,169.4**	116,609,606.1	6,804	9,616	149,388,384.0	131	**112,513,169.4**	1,811
1,800	**112,513,169.4**	114,042,937.2	948	2,717	**112,513,169.4**	114,042,937.2	7,168	6,721	149,388,384.0	124	**112,513,169.4**	404
1,900	**111,013,984.0**	**111,013,984.0**	1,272	1,799	**111,013,984.0**	113,029,241.8	9,660	11,045	129,694,929.8	123	**111,013,984.0**	679
2,000	**87,926,265.2**	96,901,773.1	1,530	3,883	**87,926,265.2**	107,267,710.9	7,826	9,130	129,694,929.8	123	**87,926,265.2**	70
Avg	114,229,155.8	115,896,328.0	769.1	2,284.0	114,476,723.1	118,167,740.9	7,287.7	8,006.2	155,864,630.0	125.6	114,229,155.8	462.6
$\mathcal{WOI}$	625,192,549.9											

4.2.1. The moderate size problem set $\mathcal{CMP}$ results

The querying performance for the moderate size problem set $\mathcal{CMP}$ presented in Table 2 indicates that the $\mathcal{MILP}$ algorithm has again generated better results in general. The $\mathcal{MLIP}$ has outperformed the methods $\mathcal{BPSO}$, $\mathcal{GAI}$ and $\mathcal{DM}$ for both the best solutions found and the computation time utilized. In terms of the best solutions found, the $\mathcal{MILP}$ approach has generated the best solutions in all 16 cases (100 %), the $\mathcal{BPSO}$ algorithm has generated the best solutions in 78 out of 80 cases (97.5%) while the $\mathcal{GAI}$ method has generated the best solutions in 60 out of 80 cases (75%), the $\mathcal{DM}$ method did not generate any best solution in all 16 cases (i.e., 0 out of 80, or 0%). In terms of the average number of evaluations performed, the $\mathcal{BPSO}$ needed about 8 times (8.46 exactly) less evaluations than the $\mathcal{GAI}$, yet achieving better solution quality. In terms of the average computation time, the $\mathcal{MILP}$ was about 4 times (3.85 exactly) times faster than the $\mathcal{BPSO}$ and about 24 times (23.87 exactly) faster than the $\mathcal{GAI}$. In summary, as also indicated by the last row of Table 2, the $\mathcal{MILP}$ algorithm has shown considerably better performance than both the $\mathcal{BPSO}$ and $\mathcal{GAI}$ approaches in all aspects.

4.2.2. The larger size problem set $\mathcal{CLP}$ results

Table 3 shows the querying performance for the $\mathcal{MILP}$,$\mathcal{BPSO}$, $\mathcal{GAI}$ and $\mathcal{DM}$ approaches for the larger size problem set $\mathcal{CLP}$. Note that this class is the hardest one. The $\mathcal{MILP}$ approach has again generated better results in general. The $\mathcal{MILP}$ has outperformed three approaches $\mathcal{BPSO}$, $\mathcal{GAI}$ and $\mathcal{DM}$ for both the best solutions found and computation time utilized. In terms of the best solutions found, the $\mathcal{MILP}$ approach has generated the best solutions in all 16 cases (100%), while the $\mathcal{BPSO}$ algorithm has generated the best solutions in 68 out of 80 cases (85%) and the $\mathcal{GAI}$ method has generated the best solutions in 43 out of 80 cases (53.75%). the $\mathcal{DM}$ method has not generated any best solution in all 16 cases (i.e., 0 out of 80, or 0%). In terms of the average number of evaluations performed, the $\mathcal{BPSO}$ needed about 9 times (9.47 exactly) less evaluations than the $\mathcal{GAI}$, yet achieving better solution quality. In terms of the average computation time, the $\mathcal{MILP}$ was about 5 times (4.85 exactly) faster than the $\mathcal{BPSO}$ and about 17 times (17.30 exactly) faster than the $\mathcal{GAI}$. In summary, as also indicated by the last row of Table 3, the $\mathcal{MILP}$ algorithm has again shown considerably better performance than both the $\mathcal{BPSO}$ and $\mathcal{GAI}$ approaches in all aspects.

4.3. Performance Scalability Study

Experiments were extended (i.e., scaled up) to further analyze the effectiveness of the $\mathcal{MILP}$ approach against the $\mathcal{BPSO}$, $\mathcal{GAI}$ and $\mathcal{DM}$ approaches. The cost model was the same as the one used in the previous experiments. In the scalability study, the fact table size has been increased 30 million to 150 million tuples in 30 millions tuple increments (five different cases) and for each of the the fact table size increments, the storage size S was systematically increased from 500 MB to 2000 MB in 500 MB increments (four different cases), yielding a total of 20 different cases. Again, the average cost of solutions over five independent runs was reported for the stochastic algorithms $\mathcal{BPSO}$ and $\mathcal{GAI}$. Therefore, a total of 100 runs (20×5) was under consideration for the algorithms $\mathcal{BPSO}$ and $\mathcal{GAI}$. The querying performance of scalability experiments for the two problem sets $\mathcal{CMP}$ and $\mathcal{CLP}$ are presented in Table 4 and Table 5 respectively (see Section 4.3 for table details). The additional column $|\mathcal{F}|$ represents the size of the fact table used in millions. The $\mathcal{WOI}$ entries represents the cost of the query workload without using any index configuration for different fact table sizes.

4.3.1. Scalability results for the moderate size problem set $\mathcal{CMP}$.

The performance of scalability experiments for the moderate size problem set presented in Table 4 indicates that the $\mathcal{MILP}$ algorithm has again generated better results than the algorithms $\mathcal{BPSO}$, $\mathcal{GAI}$ and $\mathcal{DM}$. The $\mathcal{MILP}$ has outperformed the algorithms $\mathcal{BPSO}$, $\mathcal{GAI}$ and and $\mathcal{DM}$ for both the best solutions found and computation time utilized. The $\mathcal{MILP}$ has always generated the best solutions in all runs (100%). The $\mathcal{BPSO}$ has generated the best solution in 97 out of 100 cases (97 %). The $\mathcal{GAI}$ approach has generated the best solutions in 37 out of 100 runs (37%). The $\mathcal{DM}$ has not generated any best solution in all cases. In terms of the average number of evaluations performed, the $\mathcal{BPSO}$ needed about times (3.08 exactly) less evaluations than the $\mathcal{GAI}$, yet achieving better solution quality. In terms of the average computation time, the $\mathcal{MILP}$ was about 30 times (30.44 exactly) and about 94 times (93.83 exactly) faster than the $\mathcal{BPSO}$ and $\mathcal{GAI}$ respectively. In summary, as also indicated by the last row of Table 4, the $\mathcal{MILP}$ approach has again shown considerably better performance than the $\mathcal{BPSO}$, $\mathcal{GAI}$ and $\mathcal{DM}$ approaches in all aspects.

4.3.2. Scalability results for the larger size problem set $\mathcal{CLP}$.

Table 5 shows the performance for the $\mathcal{MILP}$, $\mathcal{BPSO}$, $\mathcal{GAI}$ and and $\mathcal{DM}$ approaches for the larger size problem set $\mathcal{CLP}$ in scalability. Note that this class is the hardest one (it gets even harder when a parameter size is increased). The $\mathcal{MILP}$ approach has again generated better results than the approaches $\mathcal{BPSO}$, $\mathcal{GAI}$ and $\mathcal{DM}$. The $\mathcal{MILP}$ has outperformed the algorithms $\mathcal{BPSO}$, $\mathcal{GAI}$ and $\mathcal{DM}$ for both the best solutions found and computation time utilized. The $\mathcal{MILP}$ has generated the best solution in all runs. For the $\mathcal{BPSO}$ has generated the best solutions in 92 out of 100 runs (92%), the $\mathcal{GAI}$ has reached the best solutions in 60 out of 100 runs (60%). The $\mathcal{DM}$ approach has not generated any best solution in all runs (0%). In terms of the average number of evaluations performed, the $\mathcal{BPSO}$ needed about 7 times (7.44 exactly) less evaluations than the $\mathcal{GAI}$, yet achieving better solution quality. In terms of the average computation time, the $\mathcal{MILP}$ was about 1.5 times (1.26 exactly) and about 20 times (9.43 exactly)

Table 4. Performance results for the moderate size problem set $\mathcal{CMP}$ in scalability.

$\mathcal{S}$ (MB)	$\|\mathcal{F}\|$	$\mathcal{BPSO}$ Best	$\mathcal{BPSO}$ Avg	$\mathcal{BPSO}$ Evals	$\mathcal{BPSO}$ Time	$\mathcal{GAI}$ Best	$\mathcal{GAI}$ Avg	$\mathcal{GAI}$ Evals	$\mathcal{GAI}$ Time	$\mathcal{DM}$ Best	$\mathcal{DM}$ Time	$\mathcal{MILP}$ Best	$\mathcal{MILP}$ Time
500	30	**69,255,881.6**	**69,255,881.6**	498	313	**69,255,881.6**	**69,255,881.6**	7,252	2,719	137,574,898.6	39	**69,255,881.6**	6
	60	**271,645,392.9**	**271,645,392.9**	546	323	**271,645,392.9**	274,126,815.4	5,530	1,871	501,491,611.5	37	**271,645,392.9**	7
	90	**443,986,821.1**	**443,986,821.1**	456	463	**443,986,821.1**	516,062,183.3	6,132	1,101	830,973,205.7	41	**443,986,821.1**	5
	120	**642,016,981.2**	**642,016,981.2**	396	393	**642,016,981.2**	842,175,326.7	8,190	1,489	1,180,044,925.8	40	**642,016,981.2**	5
	150	**802,514,641.6**	**802,514,641.6**	252	258	**802,514,641.6**	1,077,474,950.1	3,010	528	1,475,046,758.3	40	**802,514,641.6**	6
1,000	30	**69,255,881.6**	**69,255,881.6**	420	431	**69,255,881.6**	**69,255,881.6**	6,272	3,209	76,229,420.9	45	**69,255,881.6**	5
	60	**138,496,191.0**	**138,496,191.0**	384	356	**138,496,191.0**	149,263,042.6	7,210	2,701	246,317,728.4	44	**138,496,191.0**	3
	90	**343,011,227.5**	**343,011,227.5**	1,266	1,250	**343,011,227.5**	366,666,984.3	8,204	3,029	652,553,573.1	43	**343,011,227.5**	5
	120	**543,271,424.9**	**543,271,424.9**	366	333	**543,271,424.9**	577,806,869.7	8,036	2,940	1,002,958,761.5	37	**543,271,424.9**	6
	150	**739,965,819.0**	**739,965,819.0**	510	466	**739,965,819.0**	766,498,129.9	9,268	3,086	1,384,940,212.6	41	**739,965,819.0**	6
1,500	30	**63,625,903.3**	**63,625,903.3**	534	664	**63,625,903.3**	**63,625,903.3**	6,132	2,098	69,650,346.3	53	**63,625,903.3**	12
	60	**138,496,191.0**	**138,496,191.0**	390	464	**138,496,191.0**	**138,496,191.0**	4,844	1,497	152,442,561.1	45	**138,496,191.0**	6
	90	**207,735,149.9**	**207,735,149.9**	342	383	**207,735,149.9**	**207,735,149.9**	5,096	1,462	369,466,045.4	44	**207,735,149.9**	5
	120	**384,642,679.4**	**384,642,679.4**	876	948	457,343,478.6	523,215,167.6	4,634	1,046	868,836,152.3	44	**384,642,679.4**	6
	150	**644,733,058.4**	**644,733,058.4**	534	563	**644,733,058.4**	682,749,224.8	6,776	1,435	1,087,575,464.1	43	**644,733,058.4**	7
2,000	30	**62,484,606.9**	62,941,125.5	1,440	1,915	63,625,903.3	64,961,554.5	4,074	1,376	69,650,346.3	53	**62,484,606.9**	277
	60	**138,496,191.0**	**138,496,191.0**	318	390	**138,496,191.0**	**138,496,191.0**	5,334	1,637	151,368,563.0	53	**138,496,191.0**	5
	90	**207,735,149.9**	**207,735,149.9**	300	354	**207,735,149.9**	**207,735,149.9**	4,340	1,188	228,654,214.9	45	**207,735,149.9**	5
	120	**276,975,459.2**	**276,975,459.2**	492	564	**276,975,459.2**	**276,975,459.2**	5,208	1,447	492,616,764.5	44	**276,975,459.2**	5
	150	**480,797,856.8**	498,972,985.4	912	1,041	**480,797,856.8**	565,418,600.7	3,318	733	692,669,882.7	39	**480,797,856.8**	6
Avg		333,457,125.4	334,388,707.8	561.6	593.7	337,149,230.2	378,899,732.9	5,943.0	1,829.7	583,553,071.8	43.5	333,457,125.4	19.5
$\mathcal{WOI}$	30	304,421,457.6											
	60	608,797,384.4											
	90	913,177,336.7											
	120	1,217,555,436.0											
	150	1,521,931,807.0											

faster than the $\mathcal{BPSO}$ and $\mathcal{GAI}$ respectively. We have observed the time required to obtain an optimal solution by $\mathcal{MILP}$ in the case when S=2000 and $|\mathcal{F}|$=30 millions is higher compare to the one for the $\mathcal{BPSO}$. In this case Cplex solver utilized more time to obtain an exact solution. In summary, as also indicated by the last row of Table 5, the $\mathcal{MILP}$ algorithm has again shown considerably better performance than both the $\mathcal{BPSO}$, $\mathcal{GAI}$ and $\mathcal{DM}$ approaches in almost all aspects.

5. Conclusions

We have presented a mathematical formulation based on $\mathcal{MILP}$ to solve the single bitmap join indexes selection problem. The approach is different from the stochastic and statistical approaches that exist to solve the problem. The formulation also utilizes an internal bitmap called λ for accurately incorporating the cost of joins involved into the model (see Secton 3.1). We have used two classes of problem sets, the moderate size and the larger size to test the effectiveness of the $\mathcal{MILP}$ approach against two well-known best approaches, the improved genetic algorithm based approach, $\mathcal{GAI}$, the binary particle swarm optimization approach, $\mathcal{BPSO}$, and the data mining approach, $\mathcal{DM}$, on a fairly large data warehouse benchmark (APB-I benchmark). We have performed a scalability study to further analyze the effectiveness of the $\mathcal{MILP}$ approach against the $\mathcal{BPSO}$, the $\mathcal{GAI}$, and the $\mathcal{DM}$ approaches by systematically increasing the fact table size.

Both the general and scalability results have shown that the $\mathcal{MILP}$ approach outperforms the two currently best known algorithms, the $\mathcal{BPSO}$, the $\mathcal{GAI}$ and the $\mathcal{DM}$ in many aspects. The $\mathcal{MILP}$ was able to obtain an optimal solution for all problem sets considered ($\mathcal{CMP}$ and $\mathcal{CLP}$) in considerably smaller amount of time than the $\mathcal{BPSO}$, $\mathcal{GAI}$ and $\mathcal{DM}$.

The linear programming could also be used to solve the multiple bitmap join indexes selection problem ($\mathcal{MBJISP}$) and other problems in data warehouses, such as referential horizontal partitioning. In the future, we plan to apply linear programming to the horizontal partitioning problem.

Table 5. Performance results for the larger size problem set CLP in scalability.

S (MB)	$\|\mathcal{F}\|$	$BPSO$ Best	$BPSO$ Avg	$BPSO$ Evals	$BPSO$ Time	GAI Best	GAI Avg	GAI Evals	GAI Time	DM Best	DM Time	$MILP$ Best	$MILP$ Time
500	30	**145,433,669.2**	**145,433,669.2**	420	454	**145,433,669.2**	**145,433,669.2**	5,978	3,168	274,739,329.3	156	**145,433,669.2**	8
	60	**615,890,014.0**	**615,890,014.0**	516	628	**615,890,014.0**	875,514,914.1	5,670	3,374	1,186,121,710.9	156	**615,890,014.0**	41
	90	**989,257,487.4**	**989,257,487.4**	516	674	**989,257,487.4**	1,355,480,263.6	7,798	5,244	1,914,887,550.6	145	**989,257,487.4**	8
	120	**2,197,565,202.2**	**2,197,565,202.2**	492	719	**2,197,565,202.2**	2,310,318,283.6	7,812	3,343	2,809,114,176.9	110	**2,197,565,202.2**	8
	150	**2,746,938,290.4**	**2,746,938,290.4**	348	342	**2,746,938,290.4**	3,011,874,627.7	7,924	4,116	3,511,371,353.5	109	**2,746,938,290.4**	7
1,000	30	**145,433,669.2**	**145,433,669.2**	636	679	**145,433,669.2**	**145,433,669.2**	7,700	4,823	195,559,057.6	157	**145,433,669.2**	21
	60	**290,835,763.7**	**290,835,763.7**	480	581	**290,835,763.7**	**290,835,763.7**	6,510	2,534	549,444,314.4	158	**290,835,763.7**	8
	90	**799,457,922.1**	**799,457,922.1**	402	545	**799,457,922.1**	809,301,453.5	8,834	4,107	1,503,291,899.9	151	**799,457,922.1**	38
	120	**1,231,739,658.1**	**1,231,739,658.1**	276	271	**1,231,739,658.1**	1,438,358,301.5	9,324	5,184	2,372,190,492.7	149	**1,231,739,658.1**	29
	150	**1,648,737,358.9**	**1,648,737,358.9**	540	544	**1,648,737,358.9**	2,156,181,169.1	4,466	1,574	3,191,446,901.7	146	**1,648,737,358.9**	9
1,500	30	**136,172,020.1**	139,876,679.7	522	564	**136,172,020.1**	139,876,679.7	6,426	2,844	180,804,252.3	156	**136,172,020.1**	87
	60	**290,835,763.7**	**290,835,763.7**	402	499	**290,835,763.7**	293,061,055.8	6,398	3,156	398,916,447.2	161	**290,835,763.7**	8
	90	**436,235,022.3**	**436,235,022.3**	198	202	**436,235,022.3**	512,509,261.2	532	167	824,143,941.6	155	**436,235,022.3**	8
	120	**992,308,830.6**	1,036,483,969.0	450	437	**992,308,830.6**	1,240,722,844.3	5,250	2,103	1,997,202,397.5	150	**992,308,830.6**	31
	150	**1,479,675,778.4**	**1,479,675,778.4**	522	537	1,553,247,802.4	1,891,091,760.3	5,936	2,736	2,505,457,102.4	149	**1,479,675,778.4**	17
2,000	30	**136,172,020.1**	141,729,009.6	503	591	**136,172,020.1**	142,552,572.6	10,192	8,405	180,804,252.3	162	**136,172,020.1**	7,547
	60	**290,835,763.7**	**290,835,763.7**	228	229	**290,835,763.7**	294,304,783.2	7,350	5,482	391,077,873.8	157	**290,835,763.7**	21
	90	**436,235,022.3**	**436,235,022.3**	372	359	**436,235,022.3**	438,650,150.5	7,126	4,508	916,741,135.5	120	**436,235,022.3**	8
	120	**581,637,116.8**	**581,637,116.8**	462	456	**581,637,116.8**	**581,637,116.8**	6,762	3,701	1,098,848,926.7	153	**581,637,116.8**	8
	150	**1,200,897,793.9**	**1,200,897,793.9**	645	736	1,208,331,731.1	1,230,428,362.0	7,672	4,165	2,226,423,404.8	117	**1,200,897,793.9**	14
Avg		839,614,708.4	842,286,547.7	446.5	502.2	843,665,006.4	965,178,335.1	6,783.0	3,736.6	1,411,429,326.1	145.8	839,614,708.4	396.2
WOI	30	756,678,820.0											
	60	1,513,615,173.3											
	90	2,270,501,462.9											
	120	3,026,949,314.3											
	150	3,783,661,557.0											

References

1. Mishra, P., Eich, M.H.. Join processing in relational databases. *ACM Computing Surveys (CSUR)* 1992;**24**(1):63–113.
2. O'Neil, P., Quass, D.. Improved query performance with variant indexes. In: *ACM Sigmod Record*; vol. 26. ACM; 1997, p. 38–49.
3. Agrawal, S., Chaudhuri, S., Narasayya, V.R.. Automated selection of materialized views and indexes in sql databases. In: *VLDB*; vol. 2000. 2000, p. 496–505.
4. Zilio, D.C., Rao, J., Lightstone, S., Lohman, G., Storm, A., Garcia-Arellano, C., et al. Db2 design advisor: integrated automatic physical database design. In: *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment; 2004, p. 1087–1097.
5. O'Neil, P., Graefe, G.. Multi-table joins through bitmapped join indices. *ACM SIGMOD Record* 1995;**24**(3):8–11.
6. Kratica, J., Ljubić, I., Tošić, D.. *A genetic algorithm for the index selection problem*. Springer; 2003.
7. Comer, D.. The difficulty of optimum index selection. *ACM Transactions on Database Systems (TODS)* 1978;**3**(4):440–445.
8. Ozsu, M.T.. *Principles of Distributed Database Systems*. Upper Saddle River, NJ, USA: Prentice Hall Press; 3rd ed.; 2007. ISBN 9780130412126.
9. Chaudhuri, S., Datar, M., Narasayya, V.. Index selection for databases: A hardness study and a principled heuristic solution. *Knowledge and Data Engineering, IEEE Transactions on* 2004;**16**(11):1313–1323.
10. Caprara, A., Fischetti, M., Maio, D.. Exact and approximate algorithms for the index selection problem in physical database design. *Knowledge and Data Engineering, IEEE Transactions on* 1995;**7**(6):955–967.
11. Aouiche, K., Darmont, J., Boussaïd, O., Bentayeb, F.. Automatic selection of bitmap join indexes in data warehouses. In: *Data Warehousing and Knowledge Discovery*. Springer; 2005, p. 64–73.
12. Bellatreche, L., Missaoui, R., Necir, H., Drias, H.. A data mining approach for selecting bitmap join indices. *JCSE* 2007;**1**(2):177–194.
13. Bouchakri, R., Bellatreche, L.. On simplifying integrated physical database design. In: *Advances in Databases and Information Systems*. Springer; 2011, p. 333–346.
14. Toumi, L., Moussaoui, A., Ugur, A.. Particle swarm optimization for bitmap join indexes selection problem in data warehouses. *The Journal of Supercomputing* 2014;**68**(2):672–708.
15. Gacem, A., Boukhalfa, K.. Immune algorithm for bitmap join indexes. In: *Neural Information Processing*. Springer; 2012, p. 560–567.
16. Papadomanolakis, S., Ailamaki, A.. An integer linear programming approach to database design. In: *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*. IEEE; 2007, p. 442–449.
17. Steinbrunn, M., Moerkotte, G., Kemper, A.. Heuristic and randomized optimization for the join ordering problem. *The VLDB JournalThe International Journal on Very Large Data Bases* 1997;**6**(3):191–208.
18. Apb-1, olap benchmark, release ii, olap council,http:// www.olapcouncil.org/ Nov. 1998;.