BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 05/06 pp146–159
Volume 1, Number 2, June 2018
DOI: 10.26599/BDMA.2018.9020014

HPPQ: A Parallel Package Queries Processing Approach for Large-Scale Data

Meihui Shi, Derong Shen*, Tiezheng Nie, Yue Kou, and Ge Yu

Abstract: A lot of scholars have focused on developing effective techniques for package queries, and a lot of excellent approaches have been proposed. Unfortunately, most of the existing methods focus on a small volume of data. The rapid increase in data volume means that traditional methods of package queries find it difficult to meet the increasing requirements. To solve this problem, a novel optimization method of package queries (HPPQ) is proposed in this paper. First, the data is preprocessed into regions. Data preprocessing segments the dataset into multiple subsets and the centroid of the subsets is used for package queries, this effectively reduces the volume of candidate results. Furthermore, an efficient heuristic algorithm is proposed (namely IPOL-HS) based on the preprocessing results. This improves the quality of the candidate results in the iterative stage and improves the convergence rate of the heuristic algorithm. Finally, a strategy called HPR is proposed, which relies on a greedy algorithm and parallel processing to accelerate the rate of query. The experimental results show that our method can significantly reduce time consumption compared with existing methods.

Key words: package queries; heuristic algorithms; parallel processing; opposition-based learning

1 Introduction

Package query[1] is one of the hot issues in database query processing. Unlike general set-based queries, the result of package queries is a collection of data objects that satisfy constraints collectively rather than individually. It is used in, for example, vacation and travel planning[2, 3], course selection[4], team formation[5, 6], and meal planning[7]. A lot of scholars have focused on developing effective techniques for package queries and a lot of excellent approaches have been proposed. These existing algorithms are divided into several categories: exact algorithms, heuristic algorithms, and divide-and-conquer algorithms.

• Meihui Shi, Derong Shen, Tiezheng Nie, Yue Kou, and Ge Yu are with the College of Computer Science and Engineering, Northeastern University, Shenyang 110000, China. E-mail: 418316943@qq.com; shenderong@cse.neu.edu.cn; nietiezheng@cse.neu.edu.cn; kouyue@cse.neu.edu.cn; yuge@cse.neu.edu.cn.

* To whom correspondence should be addressed.
Manuscript received: 2018-01-08; accepted: 2018-01-11

Even though exact algorithms[8] can get the solutions quickly when solving problems on small-scale datasets, complex problems with multiple constraints remain a challenge. High query accuracy is not demanded in many fields, i.e., it is allowed to obtain a suboptimal solution. However, it is necessary to get the solution as fast as possible. Examples of applications follow:

Example 1. (Meal Planner). *A dietitian needs to make a meal plan for a user. They require calories in a specified range and minimum fat intake.*

In Example 1, we know that there are many data objects in the food dataset. The candidate solutions will be exponential. Although exact algorithms can obtain an optimal solution, they take a lot of time. In fact, users can accept a suboptimal solution if they do not want to wait for hours or even days.

Example 2. (Vacation and Travel Planning). *The total time required for the visit is not more than the specified time, and the total cost of spending is within the specified range visiting the largest number of attractions.*

In Example 2, users need a local travel plan during their trip. They do not need the most accurate solution, but they do need to obtain their travel plan in a relatively short period of time.

Based on the above, scholars have proposed heuristic algorithms. Heuristic algorithms[9–11] do not rely on gradient information and provide excellent performance in the problem of package queries. Compared with the exact methods, heuristic algorithms may get a suboptimal solution, but they greatly reduce the run time. However, with the rapid growth in data volume, heuristic algorithms used to solve package queries also find it difficult to meet the increasing demands of efficiency. Some scholars have used divide-and-conquer algorithms[1,7] to divide the problem into multiple sub-problems and improve query efficiency. But these use exact algorithms to solve the sub-problems, which affect the run time to a certain extent.

Therefore, it is necessary to design an efficient method of package queries for large volumes of data. In this paper, we present a method called HPPQ (Heuristic Parallel Package Queries), which is based on heuristic and divide-and-conquer strategies. It optimizes the method of package queries mainly through two aspects: improving the quality of the candidate solutions and accelerating the speed of query. To address these two aspects, we propose the IPOL-HS algorithm and the HPR strategy, respectively.

Our main contributions are summarized as follows:

- We propose an efficient heuristic algorithm, namely IPOL-HS. It is an efficient harmony search based on improved partial opposition-based learning. In the process of heuristic searching, the quality of the candidate solutions formed by each iteration is increased to accelerate convergence and reduce run time.
- To further improve the efficiency of package queries processing, a strategy called HPR is proposed, which relies on a greedy algorithm and parallel processing to accelerate query speed.
- We conducted a comprehensive set of experiments on a series of synthetic datasets and a real dataset to verify the effectiveness and efficiency of the proposed algorithms. The experimental results show that our algorithms can significantly reduce time consumption compared with existing methods.

The rest of this paper is organized as follows. Section 2 discusses related works. Section 3 presents the preliminary study on our algorithms. Section 4 proposes our package queries processing method. Section 5 shows the experimental results and Section 6 concludes.

2 Related Work

In this section, we give a brief overview of existing work related to package queries and harmony searches.

2.1 Package queries

In general, package queries processing algorithms are mainly exact, heuristic, or divide-and-conquer algorithms. Most early scholars used exact algorithms for package queries processing, such as the dynamic programming algorithm[12] and the branch-and-bound algorithm[13,14].

However, with the problems becoming increasing complex, exact algorithms do not meet the increasing demands for efficiency. Therefore, package queries algorithms based on heuristic searches have attracted much attention and research, and many variants have been proposed. Haddar et al.[15] proposed a new hybrid heuristic approach that combines the quantum particle swarm optimization technique with a local search method. Feng et al.[16] proposed an improved hybrid encoding cuckoo search algorithm with a greedy strategy. Rezoug and Boughhaci[17] presented a new hybrid self-adaptive harmony search combined with a stochastic local search algorithm, namely SAHS-SLS.

The volume of data means that both exact and heuristic algorithms need a long run time. Therefore, some scholars began to use the divide-and-conquer method. The problem is divided into multiple sub-problems to improve query efficiency. Brucato et al.[7] designed a system called PackageBuilder, and proposed two types of algorithm, DIRECT and SKETCHREFINE, to solve the problem of package queries. These algorithms are used to solve the problem of different scales of data. The DIRECT algorithm has two obvious drawbacks. It is only feasible if the data scale is small enough and the time required to obtain the results set by this exact algorithm is high. In large-scale data processing the increase in the number of candidate results is a serious burden for the database. Therefore, Brucato et al.[7] proposed SKETCHREFINE, which uses a divide-and-conquer algorithm to solve this problem. However, this algorithm does not solve the problem of high time complexity in the DIRECT algorithm. In a large-scale data environment, the

SKETCHREFINE algorithm divides the problem into multiple sub-problems, so that the amount of data to be solved by each subproblem is within a small range. However, as the volume of data grows too many sub-problems occur, and as a parallel mechanism is not used for these sub-problems they are solved sequentially, which affects the query efficiency.

In short, the existing methods for package queries have many deficiencies when dealing with large-scale data and cannot guarantee query efficiency.

2.2 Harmony search

The Harmony Search (HS) algorithm[18] is a new meta-heuristic optimization method imitating the music improvisation process whereby musicians experiment with the pitch of their instruments searching for a perfect state of harmony. As the HS algorithm has excellent global search abilities, we selected it as the basic algorithm for use in this paper. Scholars have proposed many variants to improve the performance of the traditional HS algorithm.

Mahdavi et al.[19] proposed an improved HS algorithm for solving optimization problems, namely IHS. IHS employs a novel method for generating new solution vectors that enhance the accuracy and convergence rate of the HS algorithm. Omran and Mahdavi[20] presented a new variant of HS, called global-best harmony search. Pan et al.[21] proposed a self-adaptive global best harmony search algorithm for solving continuous optimization problems. Zou et al.[22] presented a Novel Global Harmony Search (NGHS) algorithm to solve unconstrained problems. Gao et al.[23] proposed an HS hybrid optimization method and opposition-based learning. These algorithms improve different aspects of the HS algorithm. However, most are based on using a random solution as the candidate solution to calculate, which affects their performance to a certain extent.

The algorithms proposed in this paper are different from those previously proposed. First, we improve the quality of the candidate results in each iteration by using improved partial opposition-based learning, which improves the convergence rate of the heuristic algorithm. Second, we use the parallel mechanism to increase the speed of searching for the optimal solution.

3 Preliminaries

In this section, we introduce the preliminaries relevant to this paper.

3.1 Problem definition

Let D be a dataset and assume that the number of data objects in dataset D is $|D|$. Any data object in D contains multiple measurable properties.

Definition 1 (*Package queries*). Select a subset from the dataset. The total consumption of each query attribute for all data objects in the subset is subject to corresponding constraints. And the objective function should be maximized.

The mathematical model of the package queries is given below.

In Formula (1), the 0-1 decision variables x_i indicate which objects appear in the result set. A set of objects with profits $v_i > 0$ and d resources with capacities $C_j > 0$ are given. Each object i consumes an amount $c_{i,j} \geqslant 0$ of each resource j.

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^{|D|} v_i x_i, \\ &\text{subject to} && \sum_{i=1}^{|D|} c_{i,j} x_i \leqslant C_j, \\ & && x_i \in \{0,\ 1\},\ 1 \leqslant j \leqslant d \end{aligned} \tag{1}$$

3.2 Partial opposition-based learning theory

Definition 2 (*Opposite Number*). Let us consider a real number x that is defined on an interval $[a,\ b]$, i.e., $x \in [a, b]$. The opposition number $\hat{x}$ is defined as follows:

$$\hat{x} = a + b - x \tag{2}$$

Definition 3 (*Opposition-Based Learning*). Let $g(\cdot)$ be the fitness function. In each iteration of the evolutionary algorithm, both the fitness of the original solution and its opposition solution are computed simultaneously. From the original solution and the opposition solution, select the solution with higher fitness as the new candidate solution.

Similarly, we can obtain the definition of partial opposition-based learning.

Definition 4 (*Partial Opposition-Based Learning*). Let $x(x_1,\ x_2,\ ...,\ x_m)$ be a point in multidimensional space and we can obtain its partial opposition point $\hat{x^q} = (\hat{x_1^q}, \hat{x_2^q}, ..., \hat{x_m^q})$ by converting some of its elements to their opposition number. In each iteration of the evolutionary algorithm both the fitness of the original solution and its partial opposition solution are computed simultaneously. From the original solution and the partial opposition solution, select the solution with higher fitness as the new candidate solution. This

is defined as follows:

$$x = \begin{cases} x, & g(x) > g(\hat{x^q}) \\ \hat{x^q}, & \text{or} \end{cases} \tag{3}$$

The partial opposition solution is defined as follows:

$$\hat{x_i^q} = \begin{cases} a_i + b_i - x_i, & rand(0,1) > r \\ x_i, & \text{or} \end{cases} \tag{4}$$

4 Package Queries Processing

In this section, we propose our package queries processing method, namely HPPQ.

Figure 1 shows the query processing framework of the proposed method, which is divided into three modules: data preprocessing, obtaining the optimal solution of the centroid, and replacement based on parallel processing. Two aspects are used to optimize the methods of package queries on large-scale data. First, the quality of the candidate solutions and the convergence is improved with an HS. Then, a parallel processing strategy is used to obtain the global optimal solution quickly.

4.1 Preprocessing

Processing a dataset D generates $2^{|D|}$ candidate solutions, so it is necessary to reduce the number of candidate results by preprocessing the data.

Data preprocessing divides similar objects into the same sub-regions, and when dealing with package queries, the data objects in each sub-region are treated as a whole. Data preprocessing mainly has two parts: data mapping and data partitioning. First, data objects are mapped onto corresponding data points in multidimensional space using data mapping. Then, the data points are divided according to the size of the sub-regions and the regional dispersion, and the centroid of each sub-region is calculated.

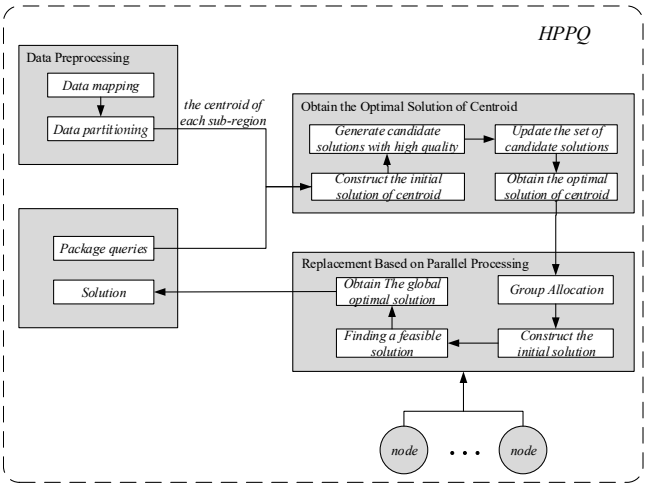


Fig. 1 Query processing framework.

Data partitioning is based on two parameters: region size and regional dispersion. The region size values affect the run time and result accuracy. If the value is too high this leads to low accuracy and if the value is too low this leads to a long query time. In the case of non-centralized data distribution in a region, the regional dispersion is limited. The region is further divided where the dispersion is greater than the threshold.

Definition 5 (*Region*). The space in which the data object is located. We use the lower and upper bounds of the space to represent the region, i.e., $\boldsymbol{R} = [R.min, R.max]$.

Definition 6 (*Region Size Threshold,* τ). It restricts the size of each region to a maximum of τ data objects.

Definition 7 (*Region Dispersion Threshold,* η). Regional dispersion is a measure of the degree of aggregation of data objects in the region. In this paper, we use the maximum distance from the regional centroid to the data object in the region to represent regional dispersion. The regional dispersion *dis* of each sub-region is calculated using the following equation:

$$dis = \max_{1 \leqslant i \leqslant |D|} \sqrt{\sum_{j=1}^{p} (\widetilde{t_j} - o_{i,j})^2} \tag{5}$$

In Eq. (5), $\widetilde{t_j}$ is the regional centroid and $o_{i,j}$ represents the j-th attribute value of the i-th data object. When preprocessing data, we do not know which attributes to use to query. So, preprocessing is based on a set of p numerical attributes that are frequently queried by users.

Different methods can be used for data partitioning. As the partitioning method based on k-dimensional quad-tree indexing results in a large number of sub-regions, our implementation is based on sequential cycle division. The iterations divide the regions into sub-regions until each sub-region satisfies the region size threshold and the regional dispersion threshold.

4.2 Obtain the optimal solution of centroid

Here, we propose a heuristic algorithm based on Improved Partial Opposition-based Learning (namely IPOL-HS) to obtain the optimal solution for the centroid.

4.2.1 Improved partial opposition-based learning

The HS algorithm is used as the basic algorithm in this paper. There have been many attempts to improve the performance of the traditional HS algorithm and,

as they are based on using a random solution as the candidate solution to calculate, they affect the performance of the algorithms to some extent. The traditional HS algorithm obtains candidate solutions by the probability generation method. A candidate solution generation algorithm based on partial opposition-based learning[11] has been proposed to improve the quality of the candidate solution, but even though it only computes the opposition values for some components of a candidate solution, the opposition-learning algorithm is improved to some extent.

In fact, in partial opposition-based learning, each component computes the opposition value with equal probability. From the original solution and partial opposition solution, select the solution with higher fitness as the new candidate solution. As it is impossible to determine which component of the original solution needs to be converted to the opposition value intelligently, there is still a possibility that a candidate solution close to the optimal solution will be deteriorated. Thus, we propose a novel method for package queries based on improved partial opposition-based learning. The current best solution in harmony memory is used as a guide as it can increase the possibility of some components with improper values changing into their opposition values and can decrease the possibility of components with proper values being converted to their opposition values.

After the data partitioning processing was complete, we used the centroid of each sub-region as the representative data object. When package queries are carried out, the maximum number of times that each representative data object appears in the result set is the size of the sub-region in which the representative data object is located. The data was preprocessed and the region of the dataset was divided into m sub-regions. After data partitioning, the data objects in each sub-region were considered as a whole. So, the centroid $\widetilde{t_i}$ can be used to represent the sub-region R_i when processing package queries. The maximum number of times that $\widetilde{t_i}$ appears in the result set is the size of the corresponding sub-region. Thus, $[a_i, b_i]$ can be converted into $[0, n_i]$. Package queries processing was performed on the preprocessed data, and the number of times that $\widetilde{t_i}$ appears in the result set can be any positive integer that does not exceed n_i. So, an m-dimensional positive integer string can be used to represent a solution of a package query.

In the basic HS algorithm, each solution is called a "harmony" and is denoted by n-dimensional real number strings. There is a Harmony Memory (HM) in the algorithm, which is used to store multiple harmonies. The algorithm's procedure mainly involves four phases: initialize HM, generate new candidate harmony, update HM, and iterative convergence. In the generate new candidate harmony phase, a new candidate harmony is improvised. The components of the new candidate solution are taken from the HM at a certain probability and formed randomly at a given probability.

In the HS search based on improved partial opposition-based learning, we generated an improved partial opposition solution in the generate new candidate harmony phase. From the original solution and improved partial opposition solution, we selected the solution with higher fitness as the new candidate solution. The improved partial opposition-based learning algorithm increases the possibility of some components with improper values changing into their opposition numbers and decreases the possibility that components with proper values are converted to their opposition numbers.

The core idea is that in the iterative optimization phase we present the current best solution in harmony memory as a guide, and the new candidate harmony will be influenced by the current best harmony by a certain probability during each iteration and absorb part of its information. So, the new candidate harmony is close to the current best harmony and mines a better harmony near the current best solution. For multidimensional spatial data points, we retain the components of the original harmony that are close to the corresponding components in the current best solution, and the components of the original harmony that are far from corresponding components in the current best solution are randomly changed. The components in the improved partial opposition solution can be defined as follows:

$$x_i^{\hat{I}q} = \begin{Bmatrix} a_i + b_i - x_i, \ |x_i - x_i^{cb}| > |\hat{x_i} - x_i^{cb}| \\ \&\&rand(0,1) < CR \\ x_i, \text{ or} \end{Bmatrix} \quad (6)$$

In Eq. (6), x_i^{cb} represents the value of the current best solution on i-th component. We determined whether the values of the components in the original harmony are close to the corresponding components in the current best solution. We reserved the component value of the original harmony when it was close to the

corresponding components in the current best solution. We calculated its opposite number with probability *CR*, when the opposite number of the component value of the original harmony was close to the corresponding components in the current best solution. The probability *CR* ensures that if the current best harmony is a local optimal solution, the possibility of falling into the local optimal can be effectively reduced by random adjustment and avoids being in the neighborhood of the local optimized solution. The current optimal solution generally has an excellent search direction, so *CR* tends to take smaller values.

Figure 2 shows an example of generating an improved partial opposition solution. The current best harmony is x^{cb}={3, 0, 3, 0, 1, 2, 5, 5, 5, 6} and the original harmony of improvisation is x^{cnew}={2, 6, 0, 2, 1, 5, 0, 4, 9, 0}. The opposition harmony obtained by opposition-based learning of the original harmony is $x^{\hat{cnew}}$={8, 2, 8, 7, 8, 3, 8, 6, 0, 9}. The values of the partial components in the improved partial opposition harmony can be determined and the values of other components are selected from the corresponding components in the original harmony with probability 1–*CR* and selected from the corresponding components in the opposition harmony with probability *CR*.

4.2.2 IPOL-HS algorithm

Figure 3 illustrates the HS execution procedure, which is based on improved partial opposition-based learning to obtain the optimal solution of the centroid. The dashed box indicates the generate new candidate harmony phase. Unlike the basic HS, an improved partial opposition solution is constructed based on the current best harmony, which accelerates the convergence of the algorithm.

The framework of our method is shown in Algorithm 1.

Using an HS based on the improved partial opposition-based learning algorithm, the optimal solution based on the centroids of the sub-regions was obtained. But this was not the eventual solution to the package query. Therefore, it was necessary to replace the centroids of the sub-regions with the real data objects in the corresponding sub-regions to get the final solution.

4.3 Replacement based on parallel processing

Each replacement program needs to be calculated and if replacing the centroids with real data objects this needs to happen sequentially. This takes a lot of run time and produces more combinations. So, in this section a strategy called HPR, based on a greedy algorithm and parallel processing, is proposed to accelerate the query speed. The HPR strategy divides into two parts: group

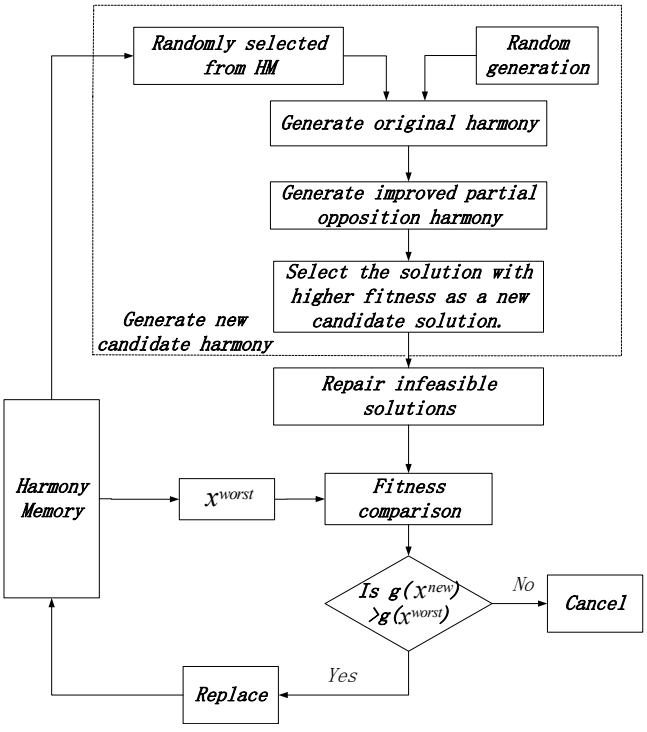


Fig. 3 Harmony search based on improved partial opposition-based learning.

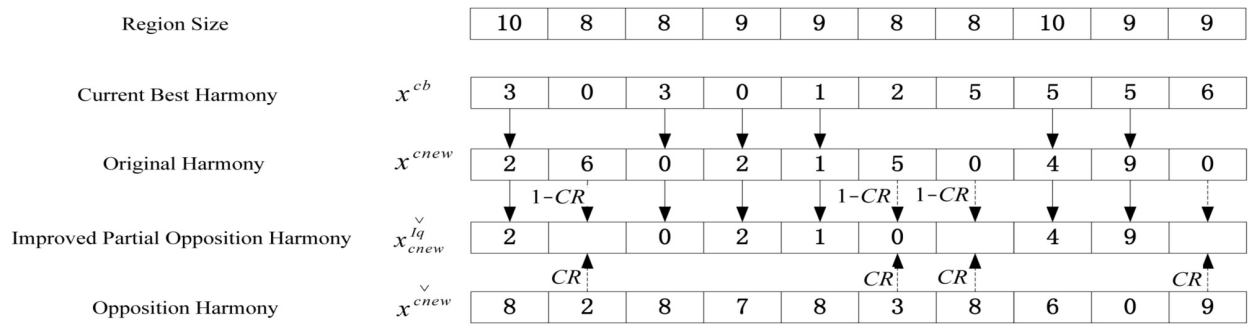


Fig. 2 The solution of improved partial opposition-based learning.

Algorithm 1 IPOL-HS

Require: harmony memory size (*HMS*), harmony memory considering rate (*HMCR*), pitch adjustment rate (*PAR*), bandwidth (*bw*), opposition construction rate (*CR*), the maximum number of generations (*T*), the maximum number of consecutive non update times(*CT*), the number of data objects in dataset (*n*)

Ensure: optimal harmony

1: Current harmony memory size (*CHMS*)←0
2: Current generation number (*GEN*)←0
3: Consecutive non update times (*t*)←0
4: **while** *CHMS* < *HMS* **do**
5: Randomly form a harmony and repair the infeasible harmony
6: *CHMS*++
7: **end while**
8: **while** (*GEN* < *T* && *t* < *CT*) **do**
9: **for** *i*=0; *i*<*n*; *i*++ **do**
10: **if** *rand*() < HMCR **then**
11: $x_i = x_i^j$, where $j \in (1, 2, ..., HMS)$
12: **if** *rand*() < *PAR* **then**
13: $x_i = x_i \pm bw$
14: **end if**
15: **else**
16: $x_i = x_{iL} + rand() \times (x_{iU} - x_{iL})$
17: **end if**
18: **end for**
19: **for** *i*=0; *i*<*n*; *i*++ **do**
20: generate the improved partial opposition number with Eq. (6)
21: **end for**
22: **if** $g(x^{cnew}) > g(x_{cnew}^{\hat{Iq}})$ **then**
23: $x^{new} = x^{cnew}$
24: **else**
25: $x^{new} = x_{cnew}^{\hat{Iq}}$
26: **end if**
27: **if** $g(x^{new}) > g(x^{worst})$ **then**
28: update *HM* and t←0
29: **else**
30: *t*++
31: **end if**
32: **end while**

allocation and parallel processing. First, we designed an allocation algorithm to balance the node load and improve parallelism. Then, using the greedy strategy and parallel processing, we effectively replaced the centroids with data objects of corresponding sub-regions to get the final solution.

4.3.1 Group allocation

Each sub-region where the centroid needs to be replaced is called a group. The set of groups, based on the improved partial opposition-based learning algorithm for HS is recorded as $G = \{g_1, g_2, ..., g_{|G|}\}$. The number of nodes in the cluster is $|N|$ and there evidently is $|G| \geq |N|$. Therefore, the $|G|$ groups need to be assigned to the $|N|$ nodes in the cluster. The allocation algorithm is based on the number of combinations generated in each group, i.e., $\binom{|g_i|}{|x_{|g_i|}^{best}|}$.

That is, the number of combinations of $|x_{|g_i|}^{best}|$ data objects is selected from the set of $|g_i|$ data objects so that, to achieve load balancing, the calculation load of each node is approximately the same.

The specific process of the group allocation algorithm is shown in Algorithm 2. The major steps are as follows.

First, eliminate sub-regions that do not require a replacement operation. That is, the centroids of these sub-regions have an element value of zero. All sub-regions with centroids that need to be replaced are composed of a set of groups and form a collection of nodes. The maximum number of combinations for each group to be replaced is calculated and arranged in descending order (line 1). The first few regions with the largest number of combinations are allocated to the nodes in the cluster in turn. Then, each time an unassigned group has the current maximum number of combinations, it is assigned to the node with the least calculation load (lines 2–7).

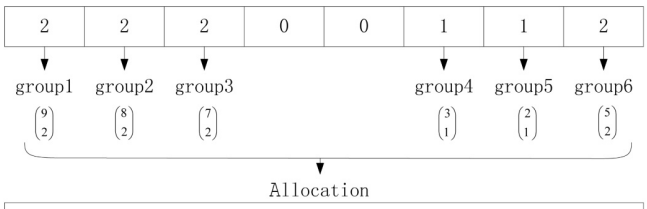
Figure 4 shows an example of how to allocate groups. Assume the optimal solution (2, 2, 2, 0, 0, 1, 1, 2) is obtained by using the partitioning results shown in Fig. 1 and the HS based on the improved partial opposition-

Algorithm 2 Group allocation algorithm

Require: the set of cluster nodes $N = \{n_1, n_2, ..., n_{|N|}\}$, the set of groups $G = \{g_1, g_2, ..., g_{|G|}\}$

Ensure: each group is assigned to a node in the cluster

1: the groups are sorted in descending order according to the number of combinations generated in the group
2: take the first $|N|$ groups, assign them to nodes in the cluster in turn
3: **while** there are unassigned groups in *G* **do**
4: get the group g with the highest number of combinations
5: find the node n_i with the least calculation load
6: assign g to n_i
7: **end while**



Node	Group	Combinations
n_1	group1	36
n_2	group2	28
n_3	group3	21
n_4	group6, group4, group5	10+3+2

Fig. 4 The allocation process of groups.

based learning algorithm. There are four nodes in the cluster. If the value of the components corresponding to the centroid of the sub-region is zero, the data objects in that sub-region will not appear in the result set, so no replacement operations are required. The remaining six groups are sorted in descending order of combinations and we get the descending sequence group1, group2, group3, group6, group4, group5. First, the first four groups are assigned to each node in turn. Next, assign group4. At this point, the node with the least number of combinations in the cluster is n_4, so assign group4 to n_4. Similarly, assign group5 to n_4.

4.3.2 Parallel processing

Inferior replacement programs for sub-problems have less likelihood of appearing in the global optimal solution, and better replacement programs have a greater likelihood of appearing in the global optimal solution. Thus, some better replacement programs can be obtained by heuristic methods and any inferior replacement programs can be eliminated directly. Although this method reduces accuracy to a certain extent, it can significantly reduce the number of replacement programs that need to be adjusted, thereby reducing the run time required for the query. By using only some of the better replacement programs to consider the interaction between replacement programs in sub-problems, the run time required for the query is further reduced.

The parallel scheme can be used to generate the optimal harmony memory for each group. However, the replacement program for each group will affect the replacement programs of the remaining groups. Therefore, before presenting the parallel processing algorithm, we first introduce a number of metrics that are used in the algorithm to coordinate the replacement programs for each group.

(1) Relative change in total value:

$$\triangle v_{ij} = v_{ij} - v_{ij_{now}} \tag{7}$$

In Eq. (7), v_{ij} represents the objective function value of the j-th replacement program in the i-th group. v_{ijnow} represents the objective function value of the j_{now}-th replacement program in the i-th group. $\triangle v_{ij}$ represents the relative change in the objective function when the replacement program of the i-th group changes from v_{ijnow} to v_{ij}.

(2) Relative change in resource consumption:

$$\triangle c_{ij} = \sum_k \{c_{ijk} - c_{ij_{now}k} \cdot r_k / |\vec{r}|\} \tag{8}$$

In Eq. (8), c_{ijk} represents the k-th resource consumption of the j-th replacement program in the i-th group, $c_{ij_{now}k}$ represents the k-th resource consumption of the j_{now}-th replacement program in the i-th group. r_k represents the current consumption of the k-th resource. $|\vec{r}|$ represents the current consumption of all resources.

(3) Scaled change of resource consumption for repairing unfeasible solutions:

$$\triangle c_{ij}^{over} = \sum_k \frac{c_{ijk} - c_{ij_{now}k}}{r_k - C_k} \tag{9}$$

In Eq. (9), $r_k - C_k$ represents the portion of the original replacement program that is exceeded on the k-th resource constraint.

(4) Scaled change of resource consumption for optimizing feasible solutions:

$$\triangle c_{ij}^{in} = \sum_k \frac{c_{ijk} - c_{ij_{now}k}}{C_k - r_k} \tag{10}$$

(5) Proportionality coefficient for repairing unfeasible solutions:

$$u_{ij}^{over} = \triangle c_{ij}^{over} / \triangle v_{ij} \tag{11}$$

(6) Proportionality coefficient for optimized feasible solutions:

$$u_{ij}^{in} = \triangle c_{ij}^{in} / \triangle v_{ij} \tag{12}$$

(7) Unfeasible coefficient:

$$f_{ijk} = r_{ijk} / C_k \tag{13}$$

In Eq. (13), if there is always $f_{ijk} < 1$ for all resource constraints, the current solution is feasible.

A parallel algorithm is proposed to efficiently replace the centroids of the sub-regions with the real data objects in the corresponding sub-region. The main premise of this is as follows: each node uses an HS based on the improved partial opposition-based learning algorithm to search the optimal harmony memory of

each group in parallel. The candidate solutions in the harmony memory are sorted in descending order of value. Therefore, the solution with the highest value in each group is in first place. The solution of the first row of each group is taken as the initial solution based on the greedy strategy. Then, the solution is upgraded or downgraded based on different metrics to find the optimal solution that satisfies the constraints.

The framework of our algorithm is shown in Algorithm 3. The major steps of the parallel processing algorithm are as follows.

Algorithm 3 Parallel processing algorithm

Require: Each harmony memory HM_i formed by corresponding group, the number of groups (NG)
Ensure: Global optimal harmony
1: $max \leftarrow 0$ $min \leftarrow 0$
2: **for** $i = 0$; i<NG; i++ **do**
3: solution $\leftarrow HM_{i0}$
4: **end for**
5: **if** the initial solution is feasible **then**
6: return solution
7: **else**
8: **while** the current solution is infeasible **do**
9: **for** $i = 0$; i<NG; i++ **do**
10: **for** each group with less valuable **do**
11: **if** $u_{ij}^{over} > max$ **then**
12: max$\leftarrow u_{ij}^{over}$
13: **end if**
14: **end for**
15: **end for**
16: Perform a replacement operation on the replacement program with proportionality coefficient for repairing infeasible solution equal to max
17: **end while**
18: **end if**
19: **while** the current solution is feasible **do**
20: **for** $i = 0$; i<NG; i++ **do**
21: **for** each group with more valuable and $f_k \leqslant 1$ **do**
22: **if** $u_{ij}^{in} < min$ **then**
23: min$\leftarrow u_{ij}^{in}$
24: **end if**
25: **end for**
26: **end for**
27: Perform a replacement operation on the replacement program with proportionality coefficient for optimized feasible solution equal to min
28: **end while**
29: **return** solution

Step 1. Greedily form the initial solution (lines 3–7). The solution in the first place of each group is taken as the initial solution. If the initial solution is feasible, the optimal result set is obtained and returned. Otherwise, Step 2 is executed.

Step 2. Calculate the metrics in parallel and reduce the value of the candidate solution to get a feasible solution (lines 8–17). Calculate u_{ij}^{over} of the replacement programs in each group, which has less value than the current replacement program in each group. Select the replacement program which is less valuable than the current replacement program in each group but has the largest global u_{ij}^{over}. Replace the current replacement program in the corresponding group with the program which has the largest global u_{ij}^{over}. If the modified solution is feasible, Step 3 is executed. Otherwise, Step 2 is executed.

Step 3. Get a better solution than the current solution (lines 19–28). For each replacement program that has more value than the current replacement program, the relative change in total resource consumption $\triangle c_{ij}$ can not be negative. This means that there is no replacement program that has a greater value and consumes less total resources than the current replacement program. Therefore, we want to improve the value of the current replacement program if it ensures that the new solution obtained by the replacement operation is still feasible. If changing the current replacement program to a more valuable replacement program can make all the unfeasible coefficients for the resources still be $f_k \leq 1$, calculate its u_{ij}^{in}. Select the replacement program which has the lowest global u_{ij}^{in}. Replace the current replacement program in the corresponding group with the program which has the lowest global u_{ij}^{in}. If there is a more valuable replacement program and it does not make the feasible solution become unfeasible, Step 3 is executed. Otherwise, stop the operation and return the current optimal solution.

5 Experiments

In this section, we provide an experimental evaluation on the performance of our method HPPQ. To test the performance of the HPR strategy proposed in this paper, we compared the sequential replacement method with the HPPQ algorithm. For the sake of convenience we called it HPQ and compared it with the existing package queries algorithms SKETCHREFINE and HS from the following aspects: (1) different sizes of real

and synthetic datasets; (2) comparative experiments on convergence; (3) the impact of varying the partitioning region size threshold on the performance of algorithms; and (4) the impact of varying the partitioning coverage on the run time of algorithms.

5.1 Dataset

The proposed algorithm in this paper was evaluated on a real-world dataset and a synthetic dataset. The real-world dataset consisted of approximately ten thousand data objects collected from a food datase (https://www.kaggle.com/bls/eating-health-module-dataset). Each data object in the dataset contained 37 numeric attributes. As the value of the partial attributes of a data object is null, data with a null attribute were filtered to form a food dataset with 23 non-null numerical attributes. As the size of the real-world data set was limited, it was necessary to create a synthetic dataset, which contained about one hundred thousand data objects. As the problem of package queries is a kind of combinatorial optimization problem, the synthetic dataset was a large-scale dataset. Statistics on the two datasets are shown in Table 1.

5.2 Comparative algorithms and evaluation criteria

We compared our algorithm with two baseline methods:

(1) SKETCHREFINE: an algorithm of package queries based on the divide-and-conquer method. A package queries problem is divided into several sub-problems, then an exact method is used to deal with sub-problems.

(2) HS: the IPOL-HS algorithm proposed in this paper is based on the HS algorithm. Therefore, the HS algorithm is used as the comparison algorithm. The convergence of the proposed algorithm is compared with the HS algorithm. The run time and accuracy rate are used to evaluate its performance.

5.3 Results and discussion

Experiment 1: Comparison of different sizes of real and synthetic datasets.

The dataset in Table 2 is extracted from the real-world dataset, and the dataset in Table 3 is extracted from the synthetic dataset. Figures 5–8 show the performance

Table 1 Statistics of datasets.

	Number of data objects	Number of attributes
Real-world dataset	10139	23
Synthetic dataset	100157	23

Table 2 Different sizes of real-world dataset.

Dataset	Number of data objects	Number of attributes
(a) R1K	1030	23
(b) R3K	3104	23
(c) R5K	5169	23
(d) R8K	8263	23
(e) R10K	10 139	23

Table 3 Different sizes of synthetic dataset.

Dataset	Number of data objects	Number of attributes
(a) S10K	10 084	23
(b) S30K	30 253	23
(c) S50K	50 425	23
(d) S80K	80 691	23
(e) S100K	100 157	23

of our methods and the SKETCHREFINE algorithm on different sizes of the real-world and synthetic datasets.

In Figs. 5 and 6, the three algorithms show high accuracy and high efficiency on small data. Even in the case of processing R1K, the accuracy of all three

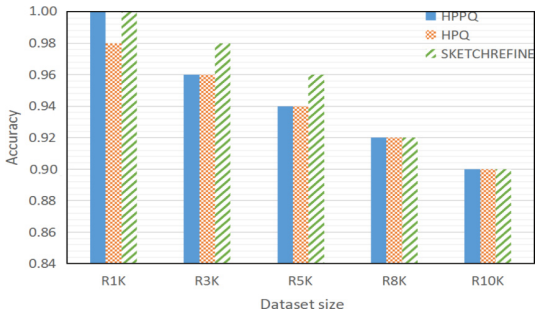


Fig. 5 Accuracy rate of different sizes of real-world dataset.

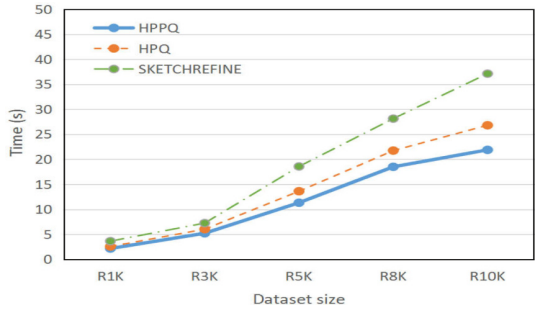


Fig. 6 Run time of different sizes of real-world dataset.

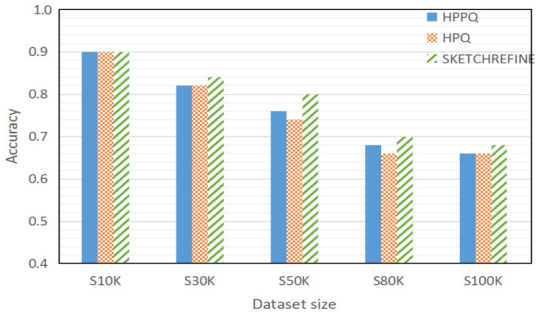


Fig. 7 Accuracy rate of different sizes of synthetic dataset.

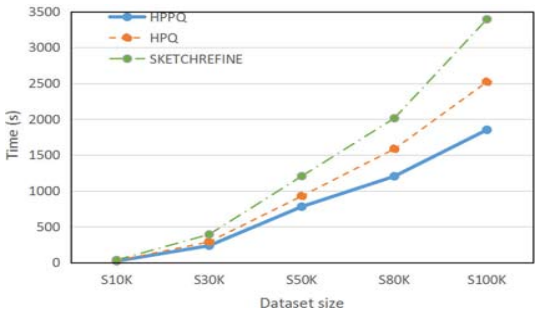


Fig. 8 Run time of different sizes of synthetic dataset.

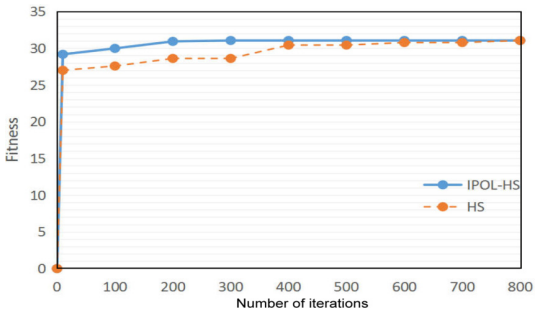


Fig. 9 Comparison of convergence on R1K.

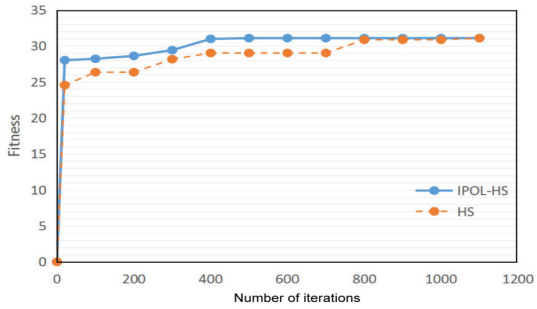


Fig. 10 Comparison of convergence on R10K.

algorithms did not reach one hundred percent. Because the centroids of the sub-regions are used to represent the data objects in the corresponding sub-regions, this, to a certain extent, reduces the accuracy of the algorithm. With an increase in data size, the accuracy of the three algorithms decreased gradually, but still achieved high accuracy. The HPQ and HPPQ algorithms may obtain suboptimal solutions because they all adopt heuristic methods.

Compared with the SKETCHREFINE algorithm, the HPQ and HPPQ algorithms use a hybrid of the divide-and-conquer and heuristic methods, so that the run time is significantly reduced. As the HPPQ algorithm had fewer replacement operations when dealing with the real-world dataset, fewer sub-problems were needed for parallel processing. Therefore, the parallel strategy introduced by the HPPQ algorithm did not significantly reduce its run time compared with the HPQ algorithm.

As can be seen from Figs. 7 and 8, this is similar to the case in the real-world datasets. The difference is that when dealing with synthetic datasets, there are more sub-problems. Compared with HPQ algorithm, the use of parallel strategy to solve the sub-problems in the HPPQ algorithm, significantly reduced the run time.

Experiment 2: Comparative experiments on convergence.

The IPOL-HS algorithm proposed in this paper is based on the HS algorithm. We compared the convergence rate of the IPOL-HS algorithm and the existing HS algorithm on different sizes of datasets and used R1K and R10K as the experimental datasets. Figures 9 and 10 show the performance of the IPOL-HS algorithm and HS algorithm on different sizes of the real-world dataset.

As can be seen from Figs. 9 and 10, for both datasets, the improved partial opposition-based learning algorithm shows excellent convergence performance. The IPOL-HS algorithm improves the quality of the candidate solution generated during the iterative process. So that each new harmony can be closer to the current best solution, which greatly accelerated the convergence rate of the algorithm.

In Fig. 9, the global optimal solution is obtained when the IPOL-HS algorithm iterates 300 times, but the HS algorithm has to iterate 800 times before it converges to the global optimal solution. In Fig. 10, using the R10K dataset, the global optimal solution is obtained when the IPOL-HS algorithm iterates 500 times, but the HS algorithm has to iterate 1100 times before it converges to the global optimal solution. Therefore, the algorithm based on the IPOL-HS algorithm is more able to search for the optimal solution compared with the HS algorithm, and the convergence advantage increases.

Experiment 3: The impact of varying the partitioning region size threshold on the performance of the algorithms.

Both the SKETCHREFINE and HPPQ algorithms adopt a data partitioning method in the data preprocessing stage. The division size depends on a threshold and this affects the performance of the two algorithms. Smaller partitions mean generating more but smaller sub-problems and larger partitions mean generating fewer but larger sub-problems. The performance of the algorithms is significantly impacted by these two cases. Therefore, we need to analyze the impact of varying partitioning region size threshold

on the performance of the algorithms. We used the real-world and synthetic datasets as the experimental datasets.

As can be seen from Figs. 11 and 12, if the region size thresholds of the two algorithms too high or too low, on both datasets, the run time of the algorithm is too long. When the region size threshold is large, the two algorithms do not achieve the effect of partitioning, which is close to direct processing. Between them, the SKETCHREFINE algorithm is similar to the exact method and can hardly obtain the global optimal solution in a limited time. The HPPQ algorithm is similar to HS and is based on an improved partial opposition-based learning algorithm. It is only a heuristic algorithm, and the run time is very long when dealing with large-scale data.

As the region size threshold decreases, the run time decreases significantly. In Figs. 11 and 12, there is an optimal number of region size thresholds in both the real-world and the synthetic datasets, which minimizes the run time. The size of the optimal threshold depends on the size of the dataset. For the real-world dataset, the optimal region size threshold for the HPPQ algorithm is 5% of the total number of data objects, and the optimal threshold for the SKETCHREFINE algorithm is 3%. For the synthetic dataset, the optimal region size threshold for the HPPQ algorithm is 1% of the total number of data objects, and the optimal threshold for the SKETCHREFINE algorithm is 0.5%.

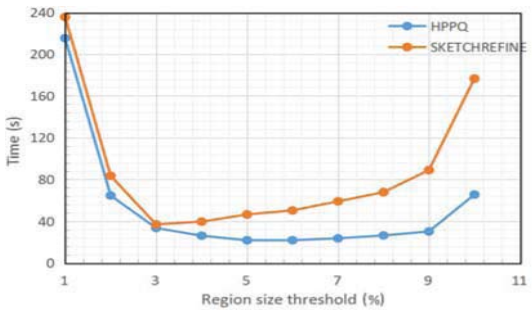


Fig. 11 Impact of threshold on real-world dataset.

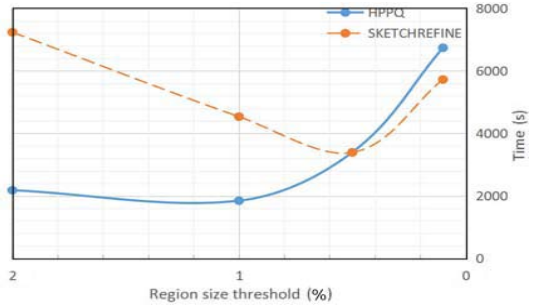


Fig. 12 Impact of threshold on synthetic dataset.

Experiment 4: Impact of varying the partitioning coverage on the run time of algorithms.

As data preprocessing is performed before query processing, the query attributes are unknown and only attributes that are queried frequently are handled as partition attributes. As a result, partitioning coverage has an important impact on the run time.

We defined the partitioning coverage as the ratio of the number of attributes contained in the partition attributes to the total number of query attributes. We used a time increase ratio to represent the change in run time. The run time for full coverage with partition attributes, i.e., a partition coverage of 1, is used as the standard run time. We used the real-world and synthetic datasets as experimental datasets to test the impact of partitioning coverage on the HPPQ algorithm.

As can be seen from Tables 4 and 5, the run time of the HPPQ algorithm decreases with an increase in partition coverage in both the real-world and synthetic datasets. When data preprocessing uses only a small number of query attributes that are queried frequently as the partition attributes, the probability of using the query attributes involved in this query in data preprocessing is reduced, and the run time increases significantly.

6 Conclusion

As a response to the requirements of large-scale data, we proposed a novel method for package queries combining heuristic methods and the divide-and-conquer strategy, namely HPPQ. The method was found to reduce the number of candidate results by data preprocessing. The IPOL-HS algorithm was then proposed to improve the convergence rate of the

Table 4 Impact of partitioning coverage on real-world dataset.

Partitioning coverage	Run time (s)	Time increase ratio
0.25	55.64	2.542
0.50	35.67	1.630
1.00	21.89	1.000
1.25	20.25	0.925

Table 5 Impact of partitioning coverage on synthetic datase.

Partitioning coverage	Run time (s)	Time increase ratio
0.250	5652	3.053
0.500	3193	1.725
1.000	1851	1.000
1.250	1697	0.917

heuristic algorithm. Then, a strategy called HPR, based on a greedy algorithm and parallel processing, was proposed to accelerate the speed of query. The experimental results show that our method can solve the problem of package queries in large-scale data and obtain the optimal solution efficiently within the range of acceptable accuracy, so satisfying the requirement of efficiency.

In the future, to further improve the convergence rate of the algorithm, the effect of different periods on the improved partial opposition operation probability should be considered. In addition, even though the method proposed in this paper reduces the run time required for package queries in the large-scale data environment to some extent, it still has the potential for further optimization. The parallel strategy mainly adopts parallel computing to reduce the run time, but there is no parallel strategy for obtaining the optimal solution of the centroid and no parallel replacement scheme for multiple sub-problems. Therefore, how to make the algorithm replace multiple sub-problems simultaneously still needs to be investigated.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Nos. 61472070 and 61672142).

References

[1] M. Brucato, J. F. Beltran, A. Abouzied, and A. Meliou, Scalable package queries in relational database systems, *Proc. VLDB Endow.*, vol. 9, no. 7, pp. 576–587, 2016.

[2] M. De Choudhury, M. Feldman, S. Amer-Yahia, N. Golbandi, R. Lempel, and C. Yu, Automatic construction of travel itineraries using social breadcrumbs, in *Proc. 21st ACM Conf. Hypertext and Hypermedia*, Toronto, Canada, 2010, pp. 35–44.

[3] M. Xie, L. V. S. Lakshmanan, and P. T. Wood, Breaking out of the box of recommendations: From items to packages, in *Proc. 4th ACM Conference on Recommender Systems*, Barcelona, Spain, 2010, pp. 151–158.

[4] A. Parameswaran, P. Venetis, and H. Garcia-Molina, Recommendation systems with complex constraints: A course recommendation perspective, *ACM Trans. Inf. Syst.*, vol. 29, no. 4, p. 20, 2011.

[5] T. Lappas, K. Liu, and E. Terzi, Finding a team of experts in social networks, in *Proc. 15th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Paris, France, 2009, pp. 467–476.

[6] A. Baykasoglu, T. Dereli, and S. Das, Project team selection using fuzzy optimization approach, *Cybern. Syst.*, vol. 38, no. 2, pp. 155–185, 2007.

[7] M. Brucato, R. Ramakrishna, A. Abouzied, and A. Meliou, PackageBuilder: From tuples to packages, *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1593–1596, 2014.

[8] Y. D. Cui, A new dynamic programming procedure for three-staged cutting patterns, *J. Global Optim.*, vol. 55, no. 2, pp. 349–357, 2013.

[9] T. Meng and Q. K. Pan, An improved fruit fly optimization algorithm for solving the multidimensional knapsack problem, *Appl. Soft Comput.*, vol. 50, pp. 79–93, 2017.

[10] Z. L. Guo, S. W. Wang, X. Z. Yue, and H. G. Yang, Global harmony search with generalized opposition-based learning, *Soft Comput.*, vol. 21, no. 8, pp. 2129–2137, 2017.

[11] R. Sarkhel, T. M. Chowdhury, M. Das, N. Das, and M. Nasipuri, A novel harmony search algorithm embedded with metaheuristic opposition based learning, *J. Intell. Fuzzy Syst.*, vol. 32, no. 4, pp. 3189–3199, 2017.

[12] M. I. Andreica, A dynamic programming framework for combinatorial optimization problems on graphs with bounded pathwidth, arXiv preprint arXiv: 0806.0840, 2008.

[13] Q. Louveaux and S. Mathieu, A combinatorial branch-and-bound algorithm for box search, *Discrete Optim.*, vol. 13, pp. 36–48, 2014.

[14] E. I. Hsu and S. A. McIlraith, Computing equivalent transformations for combinatorial optimization by branch-and-bound search, in *Proc. 3rd Annual Symposium on Combinatorial Search*, Atlanta, GA, USA, 2010.

[15] B. Haddar, M. Khemakhem, S. Hanafi, and C. Wilbaut, A hybrid quantum particle swarm optimization for the multidimensional knapsack problem, *Eng. Appl. Artif. Intell.*, vol. 55, pp. 1–13, 2016.

[16] Y. H. Feng, K. Jia, and Y. C. He, An improved hybrid encoding cuckoo search algorithm for 0-1 knapsack problems, *Comput. Intell. Neurosci.*, vol. 2014, p. 970456, 2014.

[17] A. Rezoug and D. Boughaci, A self-adaptive harmony search combined with a stochastic local search for the 0-1 multidimensional knapsack problem, *Int. J. Bio-Inspired Comput.*, vol. 8, no. 4, pp. 234–239, 2016.

[18] Z. W. Geem, J. H. Kim, and G. V. Loganathan, A new heuristic optimization algorithm: Harmony search, *Simulation*, vol. 76, no. 2, pp. 60–68, 2001.

[19] M. Mahdavi, M. Fesanghary, and E. Damangir, An improved harmony search algorithm for solving optimization problems, *Appl. Math. Comput.*, vol. 188, no. 2, pp. 1567–1579, 2007.

[20] M. G. H. Omran and M. Mahdavi, Global-best harmony search, *Appl. Math. Comput.*, vol. 198, no. 2, pp. 643–656, 2008.

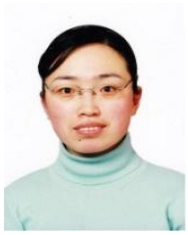
[21] Q. K. Pan, P. N. Suganthan, M. F. Tasgetiren, and J. J. Liang, A self-adaptive global best harmony search algorithm for continuous optimization problems, *Appl. Math. Comput.*, vol. 216, no. 3, pp. 830–848, 2010.

[22] D. X. Zou, L. Q. Gao, J. H. Wu, and S. Li, Novel global harmony search algorithm for unconstrained problems, *Neurocomputing*, vol. 73, nos. 16–18, pp. 3308–3318, 2010.

[23] X. Z. Gao, X. Wang, S. J. Ovaska, and K, Zenger, A hybrid optimization method of harmony search and opposition-based learning, *Eng. Optimiz.*, vol. 44, no. 8, pp. 895–914, 2012.

Meihui Shi received the BEng degree from Liaoning University in 2015, and the MEng degree from Northeastern University, China, in 2017, both in computer science. She is currently working toward the PhD degree in the College of Computer Science and Engineering, Northeastern University, China. Her research area is set-based query processing.

Derong Shen received the BEng degree in 1987, and the MEng degree in 1990, both from Jilin University. She received the PhD degree from Northeastern University in 2004. She is a professor of the College of Computer Science and Engineering, Northeastern University, China. Her research area is data integration and Web data management.



Yue Kou received the BEng degree in 2002, the MEng degree in 2005, the PhD degree in 2009, all from Northeastern University. She is an associate professor of the College of Computer Science and Engineering, Northeastern University, China. Her research area includes entity resolution and web data management.

Tiezheng Nie received the BEng degree in 2002, the MEng degree in 2005, the PhD degree in 2009, all from Northeastern University. He is an associate professor of the College of Computer Science and Engineering, Northeastern University, China. His research area includes data quality and data integration.

Ge Yu received the BEng degree in 1982, the MEng degree in 1985, both from Northeastern University. He received the PhD degree from Kyushu University, Japan, in 1996. He is a professor, the College of Computer Science and Engineering, Northeastern University, China. His research area is database and big-data management.