# ASSIGNMENT PROBLEMS OF DIFFERENT-SIZED INPUTS IN MAPREDUCE

by

**Foto Afrati, Shlomi Dolev, Ephraim Korach, Shantanu Sharma, and Jeffrey D. Ullman**

Technical Report #14-05

July 2014

# Assignment Problems of Different-Sized Inputs in MapReduce

(Technical Report – Preliminary Version)

Foto Afrati* Shlomi Dolev† Ephraim Korach‡ Shantanu Sharma§ Jeffrey D. Ullman¶

**Abstract**

A MapReduce algorithm can be described by a *mapping schema*, which assigns inputs to a set of reducers, such that for each required output there exists a reducer that receives all the inputs that participate in the computation of this output. Reducers have a capacity, which limits the sets of inputs that they can be assigned. However, individual inputs may vary in terms of size. We consider, for the first time, mapping schemas where input sizes are part of the considerations and restrictions. One of the significant parameters to optimize in any MapReduce job is communication cost between the map and reduce phases. The communication cost can be optimized by minimizing the number of copies of inputs sent to the reducers. The communication cost is closely related to the number of reducers of constrained capacity that are used to accommodate appropriately the inputs, so that the requirement of how the inputs must meet in a reducer is satisfied. In this work, we consider a family of problems where it is required that each input meets with each other input in at least one reducer. We also consider a slightly different family of problems in which, each input of a set, $X$, is required to meet each input of another set, $Y$, in at least one reducer. We prove that finding an optimal mapping schema for these families of problem is NP-hard, and present a bin-packing-based approximation algorithm for finding a near optimal mapping schema.

*Keywords*: Distributed computing, mapping schema, MapReduce algorithms, reducer capacity, and reducer capacity and communication cost tradeoff

*School of Electrical and Computing Engineering, National Technical University of Athens, Greece. Email: afrati@softlab.ece.ntua.gr. Supported by the project Handling Uncertainty in Data Intensive Applications, co-financed by the European Union (European Social Fund) and Greek national funds, through the Operational Program "Education and Lifelong Learning," under the program THALES.

†Department of Computer Science, Ben-Gurion University of the Negev, Israel. Email: dolev@cs.bgu.ac.il. Partially supported by Deutsche Telekom, Rita Altura Trust Chair in Computer Sciences, Lynne and William Frankel Center for Computer Sciences, Israel Science Foundation (grant number 428/11), Cabarnit Cyber Security MAGNET Consortium, Grant from the Institute for Future Defense Technologies Research named for the Medvedi of the Technion, and Israeli Internet Association.

‡Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Israel. Email: korach@bgu.ac.il.

§Department of Computer Science, Ben-Gurion University of the Negev, Israel. Email: sharmas@cs.bgu.ac.il.

¶Department of Computer Science, Stanford University, USA. Email: ullman@cs.stanford.edu.

# 1 Introduction

MapReduce [5] is a programming system used for parallel processing of large-scale data. The given input data is processed by the *map phase* that applies a user-defined map function to produce intermediate data (of the form $\langle key, value \rangle$). Afterwards, intermediate data is processed by the *reduce phase* that applies a user-defined reduce function to keys and their associated values. The final output is provided by the reduce phase. A detailed description of MapReduce can be found in Chapter 2 of [12].

**Reducers and Reducer Capacity.** An important parameter to be considered in MapReduce algorithms is the "reducer capacity." A *reducer* is an application of the reduce function to a single $key$ and its associated list of $value$s. The *reducer capacity* is an upper bound on the sum of the sizes of the $value$s that are assigned to the reducer. For example, we may choose the reducer capacity to be the size of the main memory of the processors on which the reducers run. We always assume in this paper that all the reducers have an identical capacity, denoted by $q$.

The term *reducer capacity* is introduced, here, for the first time. There are various works in the field of MapReduce algorithms design (*e.g.*, [10, 13, 1, 7, 11, 2]); none of them considers the reducer capacity.

**Motivation and Examples.** We demonstrate a new aspect of reducer capacity in the scope of several special cases. One useful special case is where an output depends on *exactly* two inputs. We present three examples where each output depends on exactly two inputs and define two problems that are based on these examples.

*Computing common friends.* A set of inputs consists of a set of lists of friends of $m$ persons. Each pair of lists of friends corresponds to one output, common friends of the respective persons. Thus, it is mandatory that lists of friends of every two persons are compared. Specifically, a set $F = \{f_1, f_2, \ldots, f_m\}$ of $m$ lists of friends is given, and each pair of elements $\langle f_i, f_j \rangle$ corresponds to one output, common friends of persons $i$ and $j$; see Figure 1.

*The drug-interaction problem.* The drug-interaction problem is given in [13], where a set of inputs consists of 6,500 drugs, where drug $i$ holds information about the medical history of patients who had taken drug $i$. The objective is to find pairs of drugs that had particular side effects. In order to achieve the objective, it is mandatory that each pair of drugs is compared.

*Skew join of two relations $X(A, B)$ and $Y(B, C)$.* The join of relations $X(A, B)$ and $Y(B, C)$, where the joining attribute is $B$, provides the output tuples $\langle a, b, c \rangle$, where $(a, b)$ is in $A$ and $(b, c)$ is in $C$. One or both of the relations $X$ and $Y$ may have a large number of tuples with the same $B$-value. A value of the joining attribute $B$ that occurs many times is known as a *heavy hitter*. In skew join of $X(A, B)$ and $Y(B, C)$, all the tuples of both the relations with the same heavy hitter should appear together to provide the output tuples.

In Figure 2, $b_1$ is considered as a heavy hitter, hence, it is required that all the tuples of $X(A, B)$ and $Y(B, C)$ with the heavy hitter, $b_1$, should appear together to provide the desired output tuples, $\langle a, b_1, c \rangle$ ($a \in A, b_1 \in B, c \in C$), which depend on exactly two inputs. It is worth noting that all the tuples of both the relations that have a common value of the joining attribute $B$, except $b_1$, are now also required to appear together to provide the remaining output tuples.

**Problem Statement.** We define two problems where exactly two inputs are required for computing an output, as follows:

***All-to-All problem.*** In the *all-to-all* (*A2A*) problem, a set of inputs is given, and each pair of inputs corresponds to one output. Computing common friends on a social networking site, similarity-join [3, 15, 14], the drug-interaction problem, and the Hamming distance 1 problem [1] are examples of tasks for which an output depends on exactly two inputs, and the set of outputs requires us to consider each pair of inputs.

***X-to-Y problem.*** In the *X-to-Y* (*X2Y*) problem, two disjoint sets $X$ and $Y$ are given, and each pair of elements $\langle x_i, y_j \rangle$, where $x_i \in X, y_j \in Y, \forall i, j$, of the sets $X$ and $Y$ corresponds to one output. Skew join and outer product or tensor product are examples.

The *communication cost*, *i.e.*, the total amount of data transmitted from the map phase to the reduce phase, is a significant factor in the performance of a MapReduce algorithm. The communication cost comes with tradeoff in the degree of parallelism, however.

A reducer of large enough capacity can be used to accommodate all the given inputs, and provide the desired outputs. This results in the minimum communication cost but also in the minimum parallelism. Higher parallelism requires more reducers (hence, of smaller reducer capacity), and hence a larger communication cost (because the copies of the given inputs are required to be assigned to more reducers).

A substantial level of parallelism can be achieved with fewer reducers, and hence, yield a smaller communication cost. Thus, we focus on minimizing the total number of reducers, for a given reducer capacity $q$. A smaller number of reducers results in a smaller communication cost. Thus, the reducer capacity, $q$, reflects also the degree of parallelism we want, since if we want more parallelism we can explore the problem in question for smaller $q$.

**Related Work.** MapReduce was introduced by Dean and Ghemawat in 2004 [5]. Afrati et al. [1] presents a model for MapReduce algorithms where an output depends on two inputs, and shows a tradeoff between communication cost and

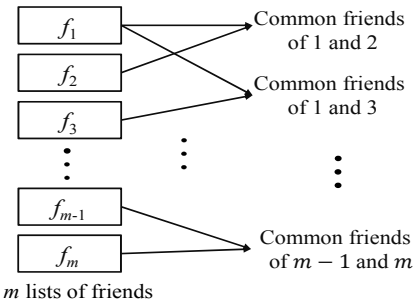



Figure 1: Computing common friends example.

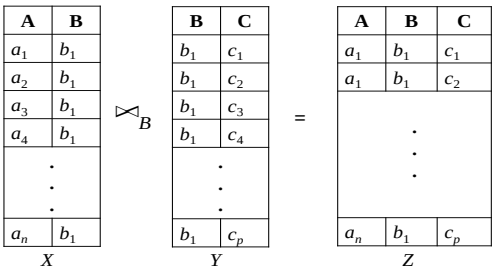



Figure 2: Skew join example for a heavy hitter, $b_1$.

parallelism. In [2], the authors consider the case where each pair of inputs produces an output and present an upper bound that meets the lower bound on communication cost as a function of the total number of inputs sent to a reducer. However, both in [1] and [2] the authors regard the reducer capacity in terms of the total number of inputs (assuming each input is of an identical size) sent to a reducer. Our setting is closely related to the settings given by Afrati et al. [1] but we allow the input sizes to be different. Thus, we consider a more realistic setting for MapReduce algorithms that can be used in various practical scenarios.

**Outline of Paper and Our Contribution.** In this paper, we provide:

- Mapping schemas for the *A2A* and the *X2Y* problems, which take into account the fact that inputs have different sizes, while all the reducers have an identical and fixed capacity (Section 2).
- A tradeoff between the reducer capacity (an upper bound on the sum of the sizes of the values that are assigned to the reducer) and the total number of reducers, which is demonstrated using similarity-join and skew join (Section 2). A tradeoff between the reducer capacity and the parallelism at the reduce phase, and a tradeoff between the reducer capacity and the communication cost (the communication cost is defined as the total amount of data that is required to move from the map to the reduce phases) is detailed in Section 2 as well.
- A proof that the *A2A mapping schema problem* for one and two reducers has a polynomial solution, and the same problem is NP-hard in the case of more than two reducers of an identical capacity (Section 3). Also, we prove that the *X2Y mapping schema problem* for one reducer has a polynomial solution, and the same problem is NP-hard in the case of more than one reducer of an identical capacity (Section 3).
- A set of heuristics, for the *A2A mapping schema problem* and the *X2Y mapping schema problem*, that is based on First-Fit Decreasing (FFD) or Best-Fit Decreasing (BFD) bin-packing algorithm, a pseudo polynomial bin-packing algorithm, and selection of a prime number $p$, and division of inputs into two sets based on their sizes (Sections 4 and 5).

# 2 Mapping Schema and Tradeoffs

Our system setting is an extension of the standard system setting [1] for MapReduce algorithms, where we consider, for the first time, inputs of different sizes. The system setting is suitable for a variety of problems where *exactly* two inputs are required to produce an output. To demonstrate the influence of the extra considerations, we need to define mapping schema and consider the communication cost tradeoff, as we elaborate next.

**Mapping Schema.** A mapping schema is an assignment of the set of inputs to some given reducers under the following two constraints:

- A reducer is assigned inputs whose sum of the sizes is less than or equal to the reducer capacity $q$.
- For each output, we must assign the corresponding inputs to at least one reducer in common.

**Tradeoffs.** The following tradeoffs appear in MapReduce algorithms and in particular in our setting:

- A tradeoff between the reducer capacity and the total number of reducers. For example, large reducer capacity allows the use of a smaller number of reducers.
- A tradeoff between the reducer capacity and the parallelism. For example, large reducer capacity results in less parallelism.
- A tradeoff between the reducer capacity and the communication cost (the communication cost is defined as the total amount of data that is required to move from the map to the reduce phase).

In the subsequent subsections, we present two types of mapping schema problems with fitting examples and explain the three tradeoffs.

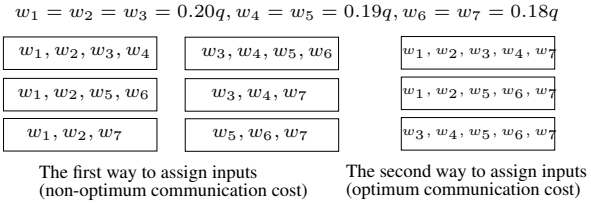



Figure 3: An example to the *A2A mapping schema problem*.

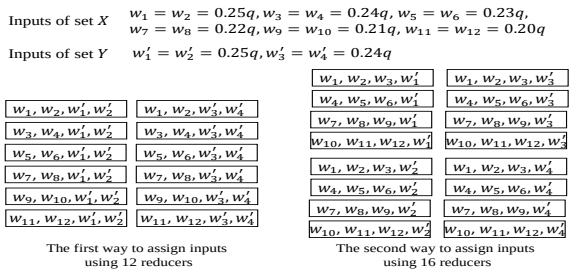



Figure 4: An example to the *X2Y mapping schema problem*.

## 2.1 The *A2A Mapping Schema Problem*

The *A2A mapping schema problem* is defined in terms of a set of *inputs*, a size for each input, a set of reducers, and a mapping from outputs to sets of inputs. An instance of the *A2A mapping schema problem* consists of a set of $m$ inputs whose input size set is $W = \{w_1, w_2, \ldots, w_m\}$ and a set of $z$ identical reducers of capacity $q$. A solution to the *A2A mapping schema problem* assigns every pair of inputs to at least one reducer in common, without exceeding $q$ at any reducer.

*Example.* We are given a set of seven inputs $I = \{i_1, i_2, \ldots, i_7\}$ whose size set is $W = \{0.20q, 0.20q, 0.20q, 0.19q, 0.19q, 0.18q, 0.18q\}$ and reducers of capacity $q$. In Figure 3, we show two different ways that we can assign the inputs to reducers. The best we can do to minimize the communication cost is to use three reducers. However, there is less parallelism at the reduce phase as compared to when we use six reducers. Observe that when we use six reducers, then all reducers have a lighter load, since each reducer may have capacity less than $0.8q$.

**Similarity-join.** Similarity-join is an example of the *A2A mapping schema problem* that can be used to find the similarity between any two inputs, *e.g.*, Web pages or documents. A set of $m$ inputs (*e.g.*, Web pages) $WP = \{wp_1, wp_2, \ldots, wp_m\}$, a similarity function $sim(x, y)$, and a similarity threshold $t$ are given, and each pair of inputs $\langle wp_x, wp_y \rangle$ corresponds to one output such that $sim(wp_x, wp_y) \geq t$. It is necessary to compare all-pairs of inputs when the similarity measure is sufficiently complex that shortcuts like locality-sensitive hashing are not available. Therefore, it is mandatory that every two inputs (Web pages) of the given input set ($WP$) are compared. The similarity-join is useful in various applications, mentioned in [3], *e.g.*, near-duplicate document detection, collaborative filtering, and query refinement for Web search.

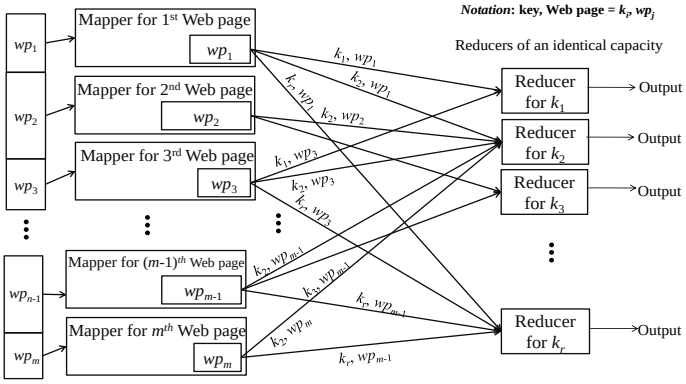



Figure 5: Similarity-join example.

In Figure 5, an example of similarity-join is given as it is applied to Web pages. We are given a set of $m$ Web pages, where the *A2A mapping schema problem* finds the similarity between every two Web pages. A mapper (a mapper is an application of the map function to a single input) would take only a single Web page, and a reducer produces pairs of every two Web pages and their similarity score.

**Explanation of tradeoffs.** The tradeoffs can also be explained with the help of the above mentioned example. Consider that $m$ Web pages are of $w_1, w_2, \ldots, w_m$ sizes. A single reducer of capacity $q = w_1 + w_2 + \ldots + w_m$ is able to find the similarity between every pair of Web pages. The use of only one reducer results in no parallelism at the reduce phase. But at the same time, the use of a single reducer yields the minimum possible communication cost. On the other hand, in case $q$ is small but is still greater than or equal to $w_i + w_j$, for any $i$ and $j$, then more reducers are required, and a higher level of parallelism is obtained. But, at the same time, the communication cost is higher, since every input is communicated to $m - 1$ reducers.

## 2.2 The *X2Y Mapping Schema Problem*

The *X2Y mapping schema problem* is defined in terms of two disjoint sets $X$ and $Y$ of *inputs*, a size for each input, a set of reducers, and a mapping from outputs to sets of inputs. An instance of the *X2Y mapping schema problem* consists of two disjoint sets $X$ and $Y$ and a set of identical reducers of capacity $q$. The inputs of the set $X$ are of sizes $w_1, w_2, \ldots, w_m$, and the inputs of the set $Y$ are of sizes $w'_1, w'_2, \ldots, w'_n$. A solution to the *X2Y mapping schema problem* assigns every two

inputs, the first from one set, $X$, and the second from the other set, $Y$, to at least one reducer in common, without exceeding $q$ at any reducer.

*Example.* We are given two sets $X$ of 12 inputs and $Y$ of 4 inputs, see Figure 4, and reducers of capacity $q$. We show that we can assign each input of the set $X$ with each input of the set $Y$ in two ways. In order to minimize the communication cost, the best way is to use 12 reducers. Note that we cannot obtain a solution for the given inputs using less than 12 reducers. However, the use of 12 reducers results in less parallelism at the reduce phase as compared to when we use 16 reducers.

**Skew join and explanation of tradeoffs.** Skew join (see Figure 2) of two relations $X(A, B)$ and $Y(B, C)$ for a heavy hitter is an example of the *X2Y mapping schema problem*. We explain the tradeoffs using the example of skew join. We assume that both the relations $X(A, B)$ and $Y(B, C)$ have only a single heavy hitter, say $b_1$. Note that we do not consider tuples with no heavy hitter.

A reducer of capacity $q$ that is sufficient to hold all the tuples of $X(A, B)$ and $Y(B, C)$ with the heavy hitter results in the minimum communication cost. However, due to a single reducer, there is no parallelism at the reduce phase. In addition, a single reducer takes a long time to produce all the desired output tuples of the heavy hitter.

In order to decrease the time (or when $q$ is small but still enough to hold only two tuples, the first from $X(A, B)$ and the second from $Y(B, C)$, which have the common $B$-value), we can restrict reducers in a way that they can hold many tuples, but not all the tuples with the heavy-hitter-value. In this case, we use more reducers, which result in a higher level of parallelism at the reduce phase. But, there is a higher communication cost, since each tuple with the heavy hitter must be sent to more than one reducer.

# 3 Intractability of Finding a Mapping Schema

In this section, we will show that the *A2A* and the *X2Y* mapping schema problems do not possess a polynomial solution. In other words, we will show that the assignment of *two required inputs* to the minimum number of identical-capacity reducers to find solutions to the *A2A* and the *X2Y* mapping schema problems cannot be achieved in polynomial time.

## 3.1 NP-hardness of the *A2A Mapping Schema Problem*

A set of inputs $I = \{i_1, i_2, \ldots, i_m\}$ whose input size set is $W = \{w_1, w_2, \ldots, w_m\}$ and a set of identical reducers $R = \{r_1, r_2, \ldots, r_z\}$, are an input instance to the *A2A mapping schema problem*. The *A2A mapping schema problem* is a decision problem that asks whether or not there exists a mapping schema for the given input instance such that every input, $i_x$, is assigned with every other input, $i_y$, to at least one reducer in common. An answer to the *A2A mapping schema problem* will be "yes," if for each pair of inputs ($\langle i_x, i_y \rangle$), there is at least one reducer that holds them.

In this section, we prove that the *A2A mapping schema problem* is NP-hard in the case of $z > 2$ identical reducers. In addition, we prove that the *A2A mapping schema problem* has a polynomial solution to one and two reducers.

If there is only one reducer, then the answer is "yes" if and only if the sum of the input sizes $\sum_{i=1}^{m} w_i$ is at most $q$. On the other hand, if $q < \sum_{i=1}^{m} w_i$, then the answer is "no." In case of two reducers, if a single reducer is not able to accommodate all the given inputs, then there must be at least one input that is assigned to only one of the reducers, and hence, this input is not paired with all the other inputs. In that case, the answer is "no." Therefore, we achieve a polynomial solution to the *A2A mapping schema problem* for one and two identical-capacity reducers.

We now consider the case of $z > 2$ and prove that the *A2A mapping schema problem* for $z > 2$ reducers is at least as hard as the partition problem.

**Throrem 1** *The problem of finding whether a mapping schema of $m$ inputs of different input sizes exists, where every two inputs are assigned to at least one of $z \geq 3$ identical-capacity reducers, is NP-hard.*

**Proof.** The proof is by a reduction from the partition problem [6] that is a known NP-complete problem. The partition problem is defined as follows: given a set $I = \{i_1, i_2, \ldots, i_m\}$ of $m$ positive integer numbers, it is required to find two disjoint subsets, $S_1 \subset I$ and $S_2 \subset I$, so that the sum of numbers in $S_1$ is equal to the sum of numbers in $S_2$, $S_1 \cap S_2 = \emptyset$, and $S_1 \cup S_2 = I$.

We are given $m$ inputs whose input size set is $W = \{w_1, w_2, \ldots, w_m\}$, and the sum of the sizes is $s = \Sigma_{1 \leq i \leq m} w_i$. We add $z - 3$ additional inputs, $ai_1, ai_2, \ldots, ai_{z-3}$, each of size $\frac{s}{2}$. We call these new $z - 3$ ($ai_1, ai_2, \ldots, ai_{z-3}$) inputs the *medium inputs*. In addition, we add one more additional input, $ai'$, of size $\frac{(z-2)s}{2}$ that we call the *big input*. Further, we assume that the reducer capacity is $\frac{(z-1)s}{2}$.

The proof proceeds in two steps: first, we prove that in case the $m$ original inputs can be partitioned, then all the inputs can be assigned to the $z$ reducers such that every two inputs are assigned to at least one

reducer, and, second, we prove that in case the mapping schema for all the inputs over the $z$ reducers is successful, then there are two disjoint subsets $S_1$ and $S_2$ of the $m$ original inputs that satisfy the partition requirements, and we can assume that, since if that sum is not divisible by 2, then the answer to the partition problem is surely "no," so the reduction of the partition problem to the *A2A mapping schema problem* is trivial.

We first show that if there are two disjoint subsets $S_1$ and $S_2$ of equal size of the $m$ original inputs, then there must exist a solution to the *A2A mapping schema problem*. Recall that any of the reducers can hold a set of inputs whose sum of the sizes is at most $\frac{(z-1)s}{2}$, and the sum of the sizes of the new $z-3$ medium inputs is exactly $\frac{(z-3)s}{2}$. Hence, a set of the $m$ original inputs $(i_1, i_2, \ldots, i_m)$ and a set of the $z-3$ medium inputs can be assigned to a single reducer (out of the $z$ reducers), and this assignment uses $s + \frac{(z-3)s}{2}$ capacity, which is exactly the capacity of any reducer. Further, the big input, $ai'$, of size $\frac{(z-2)s}{2}$ can share the same reducer with only a medium input $ai_i$. Thus, the big input, $ai'$, and all the medium inputs are assigned to $z-3$ reducers (out of the remaining $z-1$ reducers). In addition, the remaining two reducers can be used for the following assignment: the first reducer is assigned the set $S_1$ and the big input, $ai'$, and the second reducer is assigned the set $S_2$ and the big input, $ai'$. The above assignment is a solution to the *A2A mapping schema problem* for the given $m$ original inputs, the $z-3$ medium inputs, and the big input using $z$ reducers, see Figure 6.

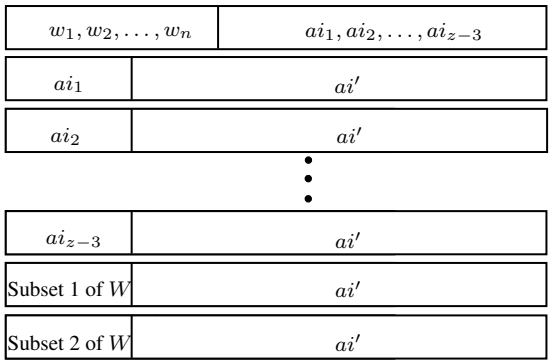



Figure 6: Proof of NP-hardness of the *A2A mapping schema problem* for $z > 2$ identical-capacity reducers, Theorem 1.

Now, we show that a solution to the *A2A mapping schema problem* — for all the inputs over the $z$ reducers — results in a partition of the $m$ original inputs into two equal-sized blocks. We also show that in a solution to the *A2A mapping schema problem*, each of the $m$ original inputs and every medium input, $ai_i$, are assigned to exactly two reducers, and the big input, $ai'$, is assigned to exactly $z-1$ reducers. Recall that the total sum of the sizes is $s + \frac{(z-3)s}{2} + \frac{(z-2)s}{2} = \frac{(2z-3)s}{2}$.

Due to the reducer capacity of a single reducer, all the inputs cannot be assigned to a single reducer; only a subset of the inputs, whose sum of the sizes is at most $\frac{(z-1)s}{2}$, can be assigned to one reducer. Thus, each input is assigned to at least two reducers in order to be coupled with all the other inputs.

Moreover, the big input, $ai'$, can share the same single reducer with only a subset, $S'$, whose sum of the sizes is at most $\frac{s}{2}$. Hence, the big input, $ai'$, is required to be assigned to at least $z-3$ reducers in order to be paired with the medium inputs $ai_i$. Furthermore, the big input, $ai'$, can share the same reducer with a subset of the $m$ original inputs whose sum of the sizes is at most $\frac{s}{2}$. This fact means that the big input, $ai'$, must be assigned to two more reducers. On the other hand, all the medium inputs can share the same reducer with the original $m$ inputs. Thus, here, the total reducer capacity occupied by all the inputs is $2 \times \Sigma_{1 \leq i \leq m}\, w_i + 2 \times \frac{(z-3)s}{2} + (z-1) \times \frac{(z-2)s}{2} = 2s + (z-3)s + \frac{(z-1)(z-2)s}{2} = \frac{(z-1)zs}{2}$, which is exactly the total capacity of all the $z$ reducers. Thus, each of the $m$ original inputs and each medium input $ai_i$ cannot be assigned more than twice, and hence, each is assigned exactly twice. In addition, the big input, $ai'$, is assigned to exactly $z-1$ reducers. This fact also shows that all the reducers are entirely filled with distinct inputs. Thus, a solution to the *A2A mapping schema problem* yields partitions of the $m$ original inputs to $S_1$ and $S_2$ blocks, where the sum of the input sizes of any block is exactly $\frac{s}{2}$. Therefore, if there is a polynomial-time algorithm to construct the mapping schema, where every input is required to be paired with every other input, then the mapping schema finds the partitions of the $m$ original inputs in polynomial time. ■

## 3.2 NP-hardness of the *X2Y Mapping Schema Problem*

Two sets of inputs, $X = \{i_1, i_2, \ldots, i_m\}$ whose input size set is $W_x = \{w_1, w_2, \ldots, w_m\}$ and $Y = \{i'_1, i'_2, \ldots, i'_n\}$ whose input size set is $W_y = \{w'_1, w'_2, \ldots, w'_n\}$, and a set of identical reducers $R = \{r_1, r_2, \ldots, r_z\}$ are an input instance to the *X2Y mapping schema problem*. The *X2Y mapping schema problem* is a decision problem that asks whether or not there exists a mapping schema for the given input instance such that each input of the set $X$ is assigned with each input of the set $Y$ to at least one reducer in common. An answer to the *X2Y mapping schema problem* will be "yes," if for each pair of inputs, the first from $X$ and the second from $Y$, there is at least one reducer that has both those inputs.

The *X2Y mapping schema problem* has a polynomial solution for the case of a single reducer. If there is only one reducer, then the answer is "yes" if and only if the sum of the input sizes $\sum_{i=1}^{m} w_i + \sum_{i=1}^{n} w'_i$ is at most $q$. On the other hand, if $q < \sum_{i=1}^{m} w_i + \sum_{i=1}^{n} w'_i$, then the answer is "no." Next, we will prove that the *X2Y mapping schema problem* is an NP-hard problem for $z > 1$ identical reducers.

**Throrem 2** *The problem of finding whether a mapping schema of $m$ and $n$ inputs of different input sizes that belongs to set $X$ and set $Y$, respectively, exists, where every two inputs, the first from $X$ and the second from $Y$, are assigned to at least one of $z \geq 2$ identical-capacity reducers, is NP-hard.*

**Proof.** The proof is by a reduction from the partition problem [6] that is a known NP-complete problem. We are given a set of inputs $I = \{i_1, i_2, \ldots, i_m\}$ whose input size set is $W = \{w_1, w_2, \ldots, w_m\}$, and the sum of the sizes is $s = \Sigma_{1 \leq i \leq m} w_i$. We add $z - 2$ additional inputs, $ai_1, ai_2, \ldots, ai_{z-2}$, each of size $\frac{s}{2}$. We call these new $z - 2$ $(ai_1, ai_2, \ldots, ai_{z-2})$ inputs the *big inputs*. In addition, we add one more additional input, $ai'$, of size 1 that we call the *small input*. Further, we assume that the reducer capacity is $1 + \frac{s}{2}$. Now, the set $I$ holds $m + z - 1$ inputs.

For the *X2Y mapping schema problem*, we consider $m$ original inputs and the $z-2$ big inputs as a set $X$, and the small input as a set $Y$. A solution to the *X2Y mapping schema problem* assigns each of the $m$ original inputs and each big input (of the set $X$) with the small input of the set $Y$.

The proof proceeds in two steps: first, we prove that in case the $m$ original inputs can be partitioned, then all the $m$ original inputs, the $z - 2$ big inputs, and the small input can be assigned to the $z$ reducers such that they satisfy the *X2Y mapping schema problem*, and, second, in case the *X2Y mapping schema problem* is successful, then there are two disjoint subsets, $S_1$ and $S_2$, of the $m$ original inputs that satisfy the partition requirements.

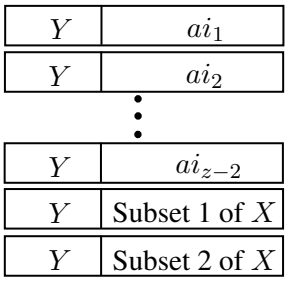


Figure 7: Proof of NP-hardness of the *X2Y mapping schema problem* for $z > 1$ identical-capacity reducers, Theorem 2.

We first show that if there are two disjoint subsets $S_1$ and $S_2$ of equal size of the $m$ original inputs, then there must exist a solution to the *X2Y mapping schema problem*. Recall that any of the reducers can hold a set of inputs whose sum of sizes is at most $1 + \frac{s}{2}$, and the sum of the sizes of the new $z - 2$ big inputs is exactly $\frac{s}{2}$. Hence, the small input, $ai'$, of size 1 and each big input, $ai_i$, can be assigned to $z-2$ reducers (out of the $z$ reducers), and this assignment uses $1 + \frac{s}{2}$ capacity, which is exactly the capacity of any reducer. In addition, the remaining two reducers can be used for the following assignment: the first remaining reducer is assigned the set $S_1$ and the small input, $ai'$, and the second remaining reducer is assigned the remaining original inputs, $S_2$, and the small input, $ai'$. The above assignment is a solution to the *X2Y mapping schema problem* (for the given $m + z - 2$ inputs of the set $X$ and the one input of the set $Y$ using $z$ reducers, see Figure 7).

Now, we prove the second claim that a solution to the *X2Y mapping schema problem* results in a partition of the $m$ original inputs into two equal-sized blocks. Recall that the total sum of the sizes is $s + \frac{(z-2)s}{2} + 1 = \frac{z \times s}{2} + 1$.

Due to the reducer capacity of a single reducer, all the inputs cannot be assigned to a single reducer; only a subset of the inputs, whose sum of the sizes is at most $1 + \frac{s}{2}$, can be assigned to a single reducer. We show that the small input, $ai'$, must be assigned to all the $z$ reducers. The small input, $ai'$, of size one can share the same single reducer with only a subset, $S'$, whose sum of the sizes is at most $\frac{s}{2}$. Hence, the small input, $ai'$, is required to be assigned to $z - 2$ reducers (out of $z$ reducers) in order to be paired with all the big inputs $ai_i$. and the remaining two reducers in order to be paired with all the $m$ original inputs. This fact results in that a solution to the *A2A mapping schema problem* yields partitions of the $m$ original inputs to $S_1$ and $S_2$ blocks, where the sum of the input sizes of any block is exactly $\frac{s}{2}$. Therefore, if there is a polynomial-time algorithm to construct the mapping schema, where every input of one set is required to be paired with every other input of another set, then the mapping schema finds the partitions of the $m$ original inputs in polynomial time. ■

# 4 Heuristics for the *A2A Mapping Schema Problem*

Since the *A2A Mapping Schema Problem* is NP-hard, in polynomial time we cannot assign each pair of inputs to the minimum number of reducers, which results in the optimum communication between the map phase and the reduce phase. In this section, we propose several heuristics for the *A2A mapping schema problem* that are based on bin-packing algorithms, selection of a prime number $p$, and division of inputs into two sets based on their sizes. In addition, the proposed heuristics assume that a *fixed* reducer capacity $q$ is given. The heuristics have two cases depending on the sizes of the inputs, as follows:

1. Input sizes are upper bounded by $\frac{q}{2}$,
2. One input is of size, say $w_i$, greater than $\frac{q}{2}$, but less than $q$, and all the other inputs have size less than or equal to $q - w_i$,

The idea of the heuristics is: if the two largest inputs are greater than the given reducer capacity $q$, then there is no solution to the *A2A mapping schema problem* because these two inputs cannot be assigned to a single reducer in common. We analyze our heuristics on three criteria, as follows:

1. *Minimum number of reducers*, $r(m, q)$. The minimum number of reducers of capacity $q$ that can solve the *A2A* (and *X2Y*) mapping schema problem(s) for the given inputs with certain size restrictions.
2. *Replication of individual inputs*. Inputs of different sizes need to be replicated at different numbers of reducers. We therefore need to consider the minimum number of reducers to which each individual input is sent.
3. *The total communication cost*, $c$. The total communication cost is defined to be the sum of all the bits that are required to transfer from the map phase to the reduce phase.

## 4.1 All the inputs sizes are upper bounded by $\frac{q}{2}$

We first consider the case where all the input sizes are at most $\frac{q}{2}$ size. We consider the following three cases in this section:

1. All the inputs are potentially of different sizes but at most size $\frac{q}{2}$,
2. All the inputs sizes are equal. Particularity, all the inputs are of size $\frac{q}{k}$, where $k > 1$,
3. Input sizes are classified as *big* and *small* inputs, where there are two options to consider the size of the big and the small inputs, as follows:
   (a) The big inputs have size between $\frac{q}{3}$ and $\frac{q}{2}$, and the small inputs have size less than or equal to $\frac{q}{3}$.
   (b) The big inputs have size between $\frac{q}{4}$ and $\frac{q}{2}$, and the small inputs have size less than or equal to $\frac{q}{4}$.

### 4.1.1 Different-sized inputs but at most size $\frac{q}{2}$

We first provide a heuristic for inputs of potentially different sizes, where the largest input can have at most size $\frac{q}{2}$. The heuristic uses a bin-packing algorithm to place the given $m$ inputs into bins of size $\frac{q}{2}$. Before going into details of the heuristic, we look at the lower bound on the replication of an input $i$ (of size $w_i$), the total number of reducers, and the total communication cost between the map phase and the reduce phase. The bounds are given in Table 1.

**Throrem 3 (Replication of individual inputs)** *For a set of $m$ inputs and a given reducer capacity $q$, an input $i$ of size $w_i < q$ is required to be sent to at least $\lceil \frac{s-w_i}{q-w_i} \rceil$ reducers for a solution to the A2A mapping schema problem, where $s$ is the sum of all the input sizes.*

**Proof.** Consider an input $i$ of size $w_i$. The input $i$ can share a reducer with inputs whose sum of the sizes is at most $q - w_i$. In order to assign the input $i$ with all the remaining $m - 1$ inputs, it is required to assign subsets of the $m - 1$ inputs, each subset with sum of sizes at most size $q - w_i$. Such an assignment results in at least $\lceil \frac{s-w_i}{q-w_i} \rceil$ subsets of the $m - 1$ inputs. Thus, the input $i$ is required to be sent to at least $\lceil \frac{s-w_i}{q-w_i} \rceil$ reducers to be paired with all the remaining $m - 1$ inputs. ■

**Throrem 4 (The total communication cost and number of reducers)** *For a set of $m$ inputs and a given reducer capacity $q$, the total communication cost and the total number of reducers, for the A2A mapping schema problem, are at least $\frac{s^2}{q}$ and $\frac{s^2}{q^2}$, respectively, where $s$ is the sum of all the input sizes.*

**Proof.** Since an input $i$ is replicated to at least $\lceil \frac{s-w_i}{q-w_i} \rceil$ reducers, the communication cost for the input $i$ is $w_i \times \lceil \frac{s-w_i}{q-w_i} \rceil$. Hence, the total communication cost for all the inputs will be at least $\sum_{i=1}^{m} w_i \frac{s-w_i}{q-w_i}$. Since $s \geq q$, we can conclude $\frac{s-w_i}{q-w_i} \geq \frac{s}{q}$. Thus, the total communication cost is at least $\sum_{i=1}^{m} w_i \frac{s}{q} = \frac{s^2}{q}$.

Since the total communication cost, the total number of bits to be assigned at reducers, is at least $\frac{s^2}{q}$, and a reducer can hold inputs whose sum of the sizes is at most $q$, the total number of reducers must be at least $\frac{s^2}{q^2}$. ■

**Bin-packing-based Heuristic.** First-Fit Decreasing (FFD) and Best-Fit Decreasing (BFD) [4] are most notable bin-packing algorithms. We use the FFD or BFD bin-packing algorithm to place the given $m$ inputs to bins of size $\frac{q}{2}$. Assume that FFD or BFD bin-packing algorithm needs $x$ bins to place $m$ inputs, and now, each of such bins is considered as a single input of size $\frac{q}{2}$. Since the reducer capacity is $q$, any two bins can be assigned to a single reducer. Hence, the heuristic uses at most $r(m, q) = \frac{x(x-1)}{2}$ reducers; see Figure 8. There also exists a pseudo polynomial bin-packing algorithm, suggested by Karger and Scott [9], that can place the $m$ inputs in as few bins as possible of size $\frac{q}{2}$.

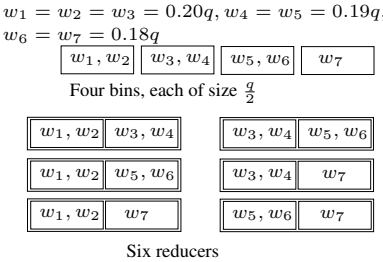



Figure 8: Bin-packing-based heuristic.

*Total required reducers.* FFD and BFD bin-packing algorithms provide an $\frac{11}{9} \cdot \text{OPT}$ approximation ratio [8], *i.e.*, if any optimum bin-packing algorithm needs OPT bins to place ($m$) inputs in the bins of a given size (say, $\frac{q}{2}$), then FFD and BFD bin-packing algorithms always use at most $\frac{11}{9} \cdot \text{OPT}$

| Cases | Solutions | Sections | Theorems | Replication of individual inputs | Reducers | Communication cost |
|---|---|---|---|---|---|---|
| The **lower bounds** for the *A2A mapping schema problem* | | | | | | |
| Different-sized inputs | | 4.1.1 | 3 and 4 | $\frac{s}{q}$ | $\frac{s^2}{q^2}$ | $\frac{s^2}{q}$ |
| Equal-sized inputs | | 4.1.2 | 6 and 7 | $\lceil \frac{m-1}{q-1} \rceil$ | $\lfloor \frac{m}{q} \rfloor \lceil \frac{m-1}{q-1} \rceil$ | $m\lceil \frac{m-1}{q-1} \rceil$ |
| The **lower bounds** for the *X2Y mapping schema problem* | | | | | | |
| Different-sized inputs | | 5 | 14 and 15 | $\frac{sum_x}{q}$, $\frac{sum_y}{q}$ | $\frac{2 \cdot sum_x \cdot sum_y}{q^2}$ | $\frac{2 \cdot sum_x \cdot sum_y}{q}$ |
| The **upper bounds** for the *A2A mapping schema problem* | | | | | | |
| Different-sized inputs | Bin-packing-based heuristic | 4.1.1 | 5 | $\frac{4s}{q}$ | $\frac{8s^2}{q^2}$ | $\frac{4s^2}{q}$ |
| Equal-Sized inputs | Algorithm 1 | 4.1.2 | 9 | $\lceil \frac{2m}{q-1} \rceil - 1$ | $(\lceil \frac{2m}{(q-1)} \rceil)^2/2$ | $m\big(\lceil \frac{2m}{(q-1)} \rceil - 1\big)$ |
| | Algorithm 2 | 4.1.2 | 11 | $\lceil \frac{2m}{q} \rceil - 1$ | $(\lceil \frac{2m}{q} \rceil)^2/2$ | $m\big(\lceil \frac{2m}{q} \rceil - 1\big)$ |
| An input of size $> \frac{q}{2}$ | Bin-packing-based heuristic | 4.2 | 13 | $m-1$ | $m-1+\frac{8s^2}{q^2}$ | $(m-1)\cdot q + \frac{4s^2}{q}$ |
| The **upper bounds** for the *X2Y mapping schema problem* | | | | | | |
| Different-sized inputs, $q = 2b$ | Bin-packing-based heuristic | 5 | 16 | $\frac{2 \cdot sum_x}{b}$, $\frac{2 \cdot sum_y}{b}$ | $\frac{4 \cdot sum_x \cdot sum_y}{b^2}$ | $\frac{4 \cdot sum_x \cdot sum_y}{b}$ |

Table 1: The bounds for heuristics for the *A2A* and the *X2Y* mapping schema problems.

bins of the same size (to place the given $m$ inputs). Since we require at most $\frac{x(x-1)}{2}$ reducers for a solution to the *A2A mapping schema problem*, the heuristic requires at most $r(m,q) = \frac{(\frac{11}{9} \cdot \text{OPT})^2}{2}$ reducers.

Note that, here in this case, OPT does not indicate the optimum number of reducers to assign $m$ inputs that satisfy the *A2A mapping schema problem*; OPT indicates the optimum number of bins of size $\frac{q}{2}$ that are required to place $m$ inputs.

The following theorem gives the upper bounds that this heuristic achieves on the replication of an inputs, the total communication cost and the number of reducers.

**Throrem 5 (Upper bounds from the heuristic)** *The above heuristic using a bin size $b = \frac{q}{2}$ where $q$ is the reducer capacity achieves the following three upper bounds: The total number of reducers, the replication of individual inputs, and the total communication cost, for the A2A mapping schema problem, are at most $\frac{8s^2}{q^2}$, at most $4\frac{s}{q}$, and at most $4\frac{s^2}{q}$, respectively, where $s$ is the sum of all the input sizes.*

**Proof.** A bin $i$ can hold inputs whose sum of the sizes is at most $b$. Since the total sum of the sizes is $s$, it is required to divide the inputs into at least $\frac{s}{b}$ bins. Since the FFD or BFD bin-packing algorithm ensures that all the bins (except only one bin) are at least half-full, each bin of size $\frac{q}{2}$ has at least inputs whose sum of the sizes is at least $\frac{q}{4}$. Thus, all the inputs can be placed in at most $\frac{s}{q/4}$ bins of size $\frac{q}{2}$. Since each bin is considered as a single input, we can assign every two bins at a reducer, and hence, we require at most $\frac{8s^2}{q^2}$ reducers. Since each bin is replicated to at most $4\frac{s}{q}$ reducers, the replication of individual inputs is at most $4\frac{s}{q}$ and the total communication cost is at most $\sum_{1 \le i \le m} w_i \times 4\frac{s}{q} = 4\frac{s^2}{q}$. ■

### 4.1.2 Equal-Sized Inputs

In this section, we provide four algorithms for $m$ equal-sized ($\frac{q}{k}$, where $k > 1$) inputs to assign each pair of inputs at reducers of capacity $q$. Equivalently, we can take the reducer capacity to be $q$ and the inputs to be of unit size. In what follows, we will continue to use $q$ as the reducer capacity and assume all inputs are of unit size. The four algorithms can be summarized as follows:

1. *2-step algorithms* (Algorithm 1 and Algorithm 2) handle the case of $m$ unit-sized inputs and odd-even values of the reducer capacity $q$. Algorithms 1 and 2 assume that $q$ is an odd or an even number, respectively.
2. In order to assign $m = q^2$ unit-sized inputs to reducers of capacity $q$, where $q$ is a prime number, an approach is suggested by Afrati and Ullman in [2]. (We refer to this approach as the *AU method*.)
   (a) We provide an extension to the *AU method* that handles $m = p^2 + p \cdot l + l$ unit-sized inputs and assigns them to reducers of capacity $p + l = q$, where $p$ is a near most prime number to $q$. We call it the *first extension to the AU method* (Algorithm 3).
   (b) We also provide another extension to the *AU method* that handles $m = q^l$ unit-sized inputs and assigns them to reducers of capacity $q$, where $q$ is a prime number and $l > 2$. We call it the *second extension to the AU method* (Algorithm 4).

**Aside.** Algorithms 1, 2, 3, and 4 have an advantage over the bin-packing-based heuristic (Section 4.1.1). When inputs of almost identical sizes are given, the bin-packing-based heuristic uses more reducers as compared to Algorithms 1, 2, 3, and 4. For example, we are given a set of seven inputs $I = \{i_1, i_2, \ldots, i_7\}$ whose size set is $W = \{0.20q, 0.20q, 0.20q, 0.19q, 0.19q, 0.18q, 0.18q\}$. In this case, the bin-packing-based heuristic uses at least six reducers while we can assign them to three reducers, see Figure 9.

Our goal to use Algorithms 1, 2, 3, and 4 is to minimize the communication cost between the map and reduce phases for a given number of unit-sized inputs and the reducer capacity $q$. Before going into details of algorithms for $q > 2$, we look at the lower bound on the total communication cost between the map and reduce phases. The case of $m$ inputs of size one and reducers of capacity two is trivial. In this case, we can assign every pair of inputs to a single reducer, which results in $r(m, q) = \frac{m(m-1)}{2}$ reducers, and moreover, it is clearly impossible to use fewer reducers.

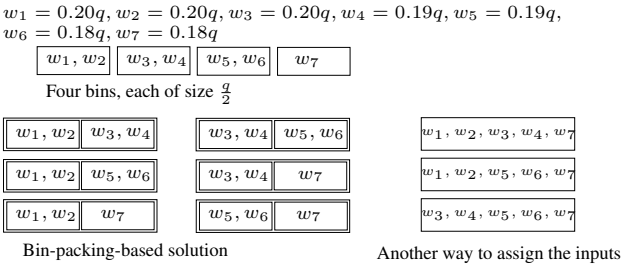



Figure 9: Comparison between the bin-packing-based heuristic and the proposed approach for equal-sized inputs.

**Throrem 6 (Replication of individual inputs)** *For a given reducer capacity $q > 1$ and a set of $m$ inputs of size one, an input $i$ required to be sent to at least $\lceil \frac{m-1}{q-1} \rceil$ reducers for a solution to the A2A mapping schema problem.*

**Proof.** Consider an input $i$. The input $i$ can share a reducer with only $q - 1$ inputs. In order to assign the input $i$ with all the remaining $m - 1$ inputs, it is required to create disjoint subsets of the remaining $m - 1$ inputs such that each subset can hold at most $q - 1$ inputs. In this manner, there are at least $\lceil \frac{m-1}{q-1} \rceil$ subsets of $m - 1$ inputs. Hence, the input $i$ is required to be sent to at least $\lceil \frac{m-1}{q-1} \rceil$ reducers. ■

**Throrem 7 (The total communication cost and number of reducers)** *For a given reducer capacity $q > 1$ and a set of $m$ inputs of size one, the total communication cost and the total number of reducers, for the A2A mapping schema problem, are at least $m\lceil \frac{m-1}{q-1} \rceil$ and at least $\lfloor \frac{m}{q} \rfloor \lceil \frac{m-1}{q-1} \rceil$, respectively.*

**Proof.** Since an input $i$ is required to be sent to at least $\lceil \frac{m-1}{q-1} \rceil$ reducers, the sum of the number of copies of $(m)$ inputs sent to reducers is at least $m\lceil \frac{m-1}{q-1} \rceil$, which result in at least $m\lceil \frac{m-1}{q-1} \rceil$ communication cost.

There are at least $m\lceil \frac{m-1}{q-1} \rceil$ total number of copies of $(m)$ inputs to be sent to reducers and a reducer can hold at most $q$ inputs; hence, at least $\lfloor \frac{m}{q} \rfloor \lceil \frac{m-1}{q-1} \rceil$ reducers are required. ■

**Algorithm 1: 2-step algorithm when the reducer capacity $q$ is an odd number**. For the sake of understanding and presentation, we first present two examples, where $q = 3$ (that is, a reducer can hold at most three unit-sized inputs) see Figure 10, and $q = 5$ (that is, a reducer can hold at most five unit-sized inputs) see Figure 11.

Following the example given for $q = 5$ (in Figure 11), we present our algorithm (see Algorithm 1) that handles any odd value of $q$. (However, we will present an efficient algorithm (Algorithm 4) to handle the case when $m = q^l$, where $l > 2$ and $q$ is a prime number). The algorithm consists of five steps as follows:

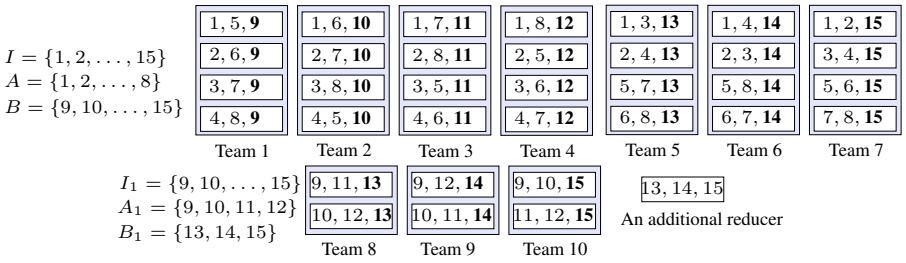
1. Divide $m$ inputs into two sets $A$ and $B$ of size $y = \lfloor \frac{q}{2} \rfloor (\lfloor \frac{2m}{q+1} \rfloor + 1)$ and $x = m - y$, respectively.
2. Group the $y$ inputs into $u = \lceil \frac{y}{q - \lceil q/2 \rceil} \rceil$ disjoint groups, where each group holds $\lceil \frac{q-1}{2} \rceil$ inputs. (We consider each of the $u$ $(= \lceil \frac{y}{q - \lceil q/2 \rceil} \rceil)$ disjoint groups as a single input that we call the *derived input*. By making $u$ disjoint groups[1] (or derived inputs) of $y$ inputs of the set $A$, we turn the case of any odd value of $q$ to a case where a reducer can hold only three inputs, the first two inputs are pairs of the derived inputs and the third input is from the set $B$.)
3. Organize $(u - 1) \times \lceil \frac{u}{2} \rceil$ reducers, each of capacity $q$, in the form of $u - 1$ teams of $\lceil \frac{u}{2} \rceil$ reducers in each team. Assign every two groups to one of $(u - 1) \times \lceil \frac{u}{2} \rceil$ reducers. To do so, we will prove the following Lemma 1.

   **Lemma 1** *Each pair of $u = 2^i$, $i > 0$, inputs can be assigned to $2^i - 1$ teams of $2^{i-1}$ reducers in each team, where the reducer capacity is $q$ and the size of each input is $\lceil \frac{q-1}{2} \rceil$.*

[1] We suppose that $u$ is a power of 2. In case $u$ is not a power of 2 and $u > q$, we add dummy inputs each of size $\lceil \frac{q-1}{2} \rceil$ so that $u$ becomes a power of 2. Consider that we require $d$ dummy inputs. If groups of inputs of the set $B$ each of size $\lceil \frac{q-1}{2} \rceil$ are less than equal to $d$ dummy inputs, then we use inputs of the set $B$ in place of dummy inputs, and the set $B$ will be empty.

Algorithm 1, with $q = 3$, divides $m$ unit-sized inputs into two disjoint sets $A$ and $B$ of $y$ and $x \leq y - 1$ inputs respectively. (The selection of the value of $y$ will be described later. But, note that we prefer $y$ to be a power of 2, and if $y \neq 2^i$ and $y > 4$, $i > 2$, then we add unit-sized dummy inputs so that $y$ is a power of 2.) When $q = 3$, we organize $(y-1) \times \lceil \frac{y}{2} \rceil$ reducers (each of capacity three) in the form of $y - 1$ *teams* of $\lceil \frac{y}{2} \rceil$ *players* (or reducers) in each team, and these reducers are used to assign each input of the set $A$ with all the remaining inputs of the sets $A$ and $B$. Note that a team must have each of the inputs of the set $A$ occurring exactly once among the reducers of that team, and this fact will be clear soon.

There are $(y-1) \times \lceil \frac{y}{2} \rceil$ pairs of inputs of the set $A$ (each of size two) and there are the same number of reducers (each of capacity three); hence, it is possible to assign one pair to each reducer, and these two inputs become two of the three inputs allowed to each reducer. Once, we assign every pair of inputs of the set $A$ to $(y-1) \times \lceil \frac{y}{2} \rceil$ reducers, then we assign $i^{th}$ input of the set $B$ to all the $\lceil \frac{y}{2} \rceil$ reducers of $i^{th}$ team. Further, we follow a similar procedure on inputs of the set $B$ to assign each pair of the remaining $x$ inputs.




**Example.** We are given 15 inputs ($I = \{1, 2, \ldots, 15\}$) of size one and $q = 3$. We create two sets, namely $A$ of $y = 8$ inputs and $B$ of $x = 7$ inputs, and arrange $(y-1) \times \lceil \frac{y}{2} \rceil = 28$ reducers in the form of 7 *teams* of 4 *players* (or reducers) in each team. These 7 teams assign each input of the set $A$ with all the remaining inputs of the set $A$ and the set $B$. We pair every two inputs of the set $A$ and assign them to exactly one of 28 reducers. (All these pairs of the inputs of the set $A$ are created and assigned using lines 11, 13, and 14 of Algorithm 1.) Once every pair of $y = 8$ inputs of the set $A$ is assigned to exactly one of 28 reducers, then we assign $i^{th}$ input of the set $B$ to all the four reducers of $i^{th}$ team, see Team 1 to Team 7. Of course, the third input in $i^{th}$ team is $i^{th}$ input of the set $B$.

Now note that the first four teams pair inputs 1-4 with inputs 5-8. The first team (Team 1) has pairs $\{1, 5\}$, $\{2, 6\}$, $\{3, 7\}$, and $\{4, 8\}$. Team 2-4 has pairs by rotation of the 5-8 inputs. Teams 5 and 6 handle pairs of 1-2 with 3-4 and 5-6 with 7-8, respectively, in the same way, and the last team has pairs $\{1, 2\}, \{3, 4\}, \ldots, \{7, 8\}$.

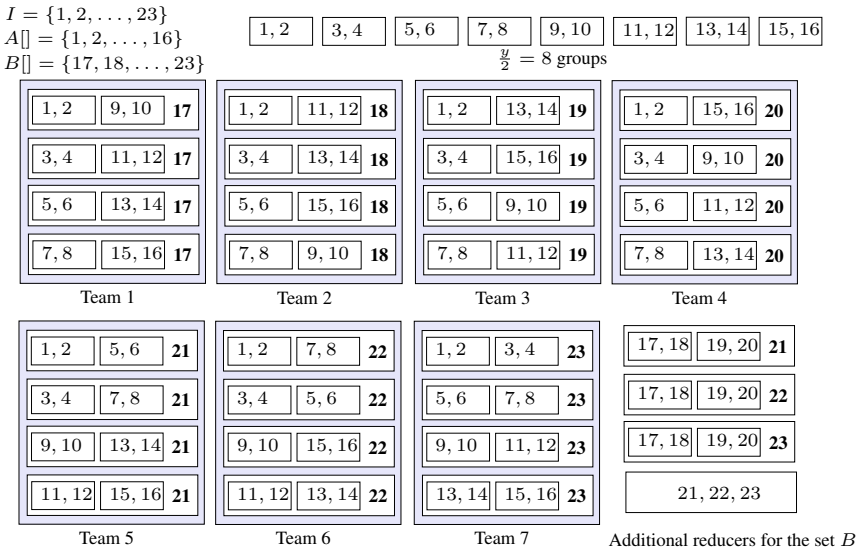
Next, we implement the same procedure on 7 inputs of the set $B$. We create two sets, say $A_1 = \{9, 10, 11, 12\}$ of $y_1 = 4$ inputs and $B_1 = \{13, 14, 15\}$ of $x_1 = 3$. Then, we arrange $(y_1 - 1) \times \lceil \frac{y_1}{2} \rceil = 6$ reducers in the form of 3 *teams* of 2 reducers in each team. We assign each pair of inputs of the set $A_1$ to these 6 reducers, and then $i^{th}$ input of the set $B_1$ to all the two reducers of a team, see Team 8 to Team 10. Further, we assign the remaining inputs of the set $B_1$ to a single reducer. The assignment of inputs to Teams 8-10 follows the same procedure as we did for Teams 1-7.

We have three claims, as follows: (*i*) each input of the set $A$ appears exactly once in each team, (*ii*) the set $B$ holds $x \leq y - 1$ inputs when $q = 3$, and (*iii*) the given algorithm assigns each pair of inputs to at least one reducer. We will prove these claims in algorithm correctness.

Figure 10: 2-step algorithm (Algorithm 1) for the reducer capacity $q = 3$ and $m = 15$.

Algorithm 1, with $q = 5$, divides $m$ unit-sized inputs into two disjoint sets $A$ and $B$ of $y$ and $x \leq y - 1$ inputs respectively (as we did for the case of $q = 3$, Figure 10). When $q = 5$, a reducer is assigned at most five inputs, where the first four inputs belong to the set $A$, and the last (fifth) input belongs to the set $B$. We first make $u = \lceil \frac{y}{2} \rceil$ disjoint groups, where each group holds $\frac{q-1}{2} = 2$ inputs of the set $A$, and each of the $u$ groups is considered as a single input that we call the *derived input*. Here, a team also has each of the derived inputs of the set A occurring exactly once among the reducers of that team (as in the case of $q = 3$).

We organize $(u - 1) \times \frac{u}{2}$ reducers in the form of $u - 1$ *teams* of $\frac{u}{2}$ reducers in each team. There are the same number ($(u-1) \times \lceil \frac{u}{2} \rceil$) of pairs of the derived inputs of the set $A$ (each of size four) and reducers (each of capacity five); hence, it is possible to assign one pair to each reducer, and this pair of two inputs become four of the five inputs allowed to each reducer. Then, we assign $i^{th}$ input of the set $B$ to all the reducers of $i^{th}$ team. In this manner, all the inputs of the set $A$ are paired with each of the $m$ inputs. Further, we follow a similar procedure on the set $B$ to assign each pair of the remaining $x$ inputs.




**Example.** We are given 23 unit-sized inputs ($I = \{1, 2, \ldots, 23\}$) and $q = 5$. We create two sets, namely $A$ of $y = 16$ inputs and $B$ of $x = 7$ inputs. Then, we make $u = \lceil \frac{y}{2} \rceil = 8$ disjoint groups, where each group holds $\frac{q-1}{2} = \frac{5-1}{2} = 2$ inputs of the set $A$, as follows: $\{1, 2\}$, $\{3, 4\}$, ..., $\{15, 16\}$. Now, each of the groups is considered as a single input, called the *derived input*.

We organize $(u - 1) \times \frac{u}{2} = 28$ reducers in the form of 7 *teams* of 4 reducers in each team. These 28 reducers are used to assign each pair of the derived inputs (as we assigned 8 inputs of the set $A$ in Figure 10). Then, we assign $i^{th}$ input of the set $B$ to all the four reducers of $i^{th}$ team; see Team 1 to Team 7. Next, we follow a similar procedure on $x = 7$ inputs of the set $B$. Since the total number of inputs of the set $B$ are very less, they can be assigned trivially.

Now note that the first four teams pair inputs 1-8 with inputs 9-16. The first team (Team 1) has pairs of derived inputs $\{\{1, 2\}, \{9, 10\}\}$, $\{\{3, 4\}, \{11, 12\}\}$, $\{\{5, 6\}, \{13, 14\}\}$, and $\{\{7, 8\}, \{15, 16\}\}$. Team 2-4 has pairs by rotation of the $\{9, 10\}$-$\{15, 16\}$ inputs. Teams 5 and 6 handle pairs of $\{1, 2\}$-$\{3, 4\}$ with $\{5, 6\}$-$\{7, 8\}$ and $\{9, 10\}$-$\{11, 12\}$ with $\{13, 14\}$-$\{15, 16\}$. The last team has pairs $\{\{1, 2\}, \{3, 4\}\}, \{\{5, 6\}, \{7, 8\}\} \ldots, \{\{13, 14\}, \{15, 16\}\}$.

In the case of $q = 5$, we have three claims as we have in the case $q = 3$. We will prove these claims in algorithm correctness.

Figure 11: 2-step algorithm (Algorithm 1) when the reducer capacity $q = 5$ and $m = 23$.

**Algorithm 1:** 2-step algorithm for an odd value of the reducer capacity $q$

**Inputs:** $m$: total number of unit-sized inputs, $q$: the reducer capacity.
**Variables:**
$A$: A set $A$, where the total inputs in the set $A$ is $y = \lfloor \frac{q}{2} \rfloor (\lfloor \frac{2m}{q+1} \rfloor + 1)$
$B$: A set $B$, where the total inputs in the $B$ is $x = m - (\lceil \frac{y}{q-\lceil q/2 \rceil} \rceil - 1)$
$Team[i, j]$ : represents teams of reducers, where index $i$ indicates $i^{th}$ team and index $j$ indicates $j^{th}$ reducer in $i^{th}$ team. Consider $u = \lceil \frac{y}{q-\lceil q/2 \rceil} \rceil$. There are $u - 1$ teams of $v = \lceil \frac{u}{2} \rceil$ reducers in each team.
$groupA[]$ : represents disjoint groups of inputs of the set $A$, where $groupA[i]$ indicates $i^{th}$ group of $\lceil \frac{q-1}{2} \rceil$ inputs of the set $A$.

**Function** $create\_group(y)$ **begin**
  **for** $i \leftarrow 1$ **to** $u$ **do** $groupA[i] \leftarrow \langle i, i+1 \ldots, i + \frac{q-1}{2} - 1 \rangle, i \leftarrow i + \frac{q-1}{2}$ ;
  $2\_step\_odd\_q(1, u)$, $Last\_Team(groupA[])$, $Assign\_input\_from\_B(Team[])$
**Function** $2\_step\_odd\_q(lower, upper)$ **begin**
  **if** $\lfloor \frac{upper-lower}{2} \rfloor < 1$ **then return**;
  **else**
    $mid \leftarrow \lceil \frac{upper-lower}{2} \rceil$, $Assignment(lower, mid, upper)$
    $2\_step\_odd\_q(lower, mid)$, $2\_step\_odd\_q(mid + 1, upper)$
**Function** $Assignment(lower, mid, upper)$ **begin**
  **while** $mid > 1$ **do**
    **foreach** $(a, t) \in [lower, lower + mid - 1] \times [0, mid - 1]$ **do**
    $Team[(u - 2 \cdot mid + 1) + t, a - \lfloor \frac{a-1}{mid} \rfloor \cdot \frac{mid}{2}] \leftarrow \langle groupA[a], groupA[value\_b(a, t, mid, upper)] \rangle$ ;
**Function** $value\_b(a, t, mid, upper)$ **begin**
  **if** $a + t + mid < upper + 1$ **then return** $(a + t + mid)$ ;
  **else if** $a + t + mid > upper$ **then return** $(a + t)$ ;
**Function** $Last\_Team(lower, mid, upper)$ **begin**
  **foreach** $i \in [1, v]$ **do** $Team[u - 1, i] \leftarrow groupA[2 \times i - 1], groupA[2 \times i]$ ;
**Function** $Assign\_input\_from\_B(Team[])$ **begin**
  **foreach** $(i, j) \in [1, u - 1] \times [1, v]$ **do** $Team[i, j] \leftarrow B[i]$ ;

**Proof.** The proof is by induction on $i$.
**Basis case.** For $i = 1$, we have $u = 2$ inputs, and we can assign them to a team of one reducer of capacity $q$. Hence, Lemma 1 holds for $(i = 1)$ two inputs.
**Inductive step.** Assume that the inductive hypothesis — there is a solution for $u = 2^{i-1}$ inputs, where all-pairs of $u = 2^{i-1}$ inputs are assigned to $2^{i-1} - 1$ teams of $2^{i-2}$ reducers in each team and have the team property (each team has one occurrence of each input, which we will prove in algorithm correctness) — is true. Now, we can build a solution for $u = 2^i$ inputs, as follows:

(a) Divide $u = 2^i$ inputs into two groups of $2^{i-1}$ inputs in each group,
(b) Recursively create teams for each of the two groups,
(c) Create some of the teams for the $2^i$ inputs by combining the $j^{th}$ team from the first group with the $j^{th}$ team from the second group. Since by the inductive hypothesis we have a solution for $u = 2^{i-1}$ inputs, we can assign inputs of these two groups to $2 \cdot (2^{i-1} - 1)$ teams of $2^{i-2}$ reducers in each team. And, by combining $j^{th}$, where $j = 1, 2, \ldots, (2^{i-1} - 1)$, teams of each group, there are $2^{i-1} - 1$ teams of $2^{i-1}$ reducers in each team; see Teams 5-7 for 8 inputs in Figure 10.
(d) Create $2^{i-1}$ additional teams that pair the inputs from the first group with inputs from the second group. In each team, the $j^{th}$ input from the first group is assigned to the $j^{th}$ reducer. In the first team, the $j^{th}$ input from the second group is also assigned to the $j^{th}$ reducer. In subsequent teams, the assignments from the second group rotate, so in the $t^{th}$ team, the $j^{th}$ input from the second group is assigned to reducer $k + j - (2^{i-1} - 1)(modulo 2^{i-1})$; see Teams 1-4 for 8 inputs in Figure 10.

By steps (c) and (d), there are total $2^{i-1} - 1 + 2^{i-1} = 2^i - 1$ teams of $2^{i-1}$ reducers in each team, and these teams holds each pair of the $u = 2^i$ inputs. ■

4. Once every pair of the derived inputs are assigned, then assign $i^{th}$ input of the set $B$ to all the reducers of $i^{th}$ team.
5. Apply (the above mentioned) steps 1-4 on the set $B$ until there is a solution to the *A2A mapping schema problem* for the $x$ inputs.

*Algorithm description.* Algorithm 1 provides a solution to the *A2A mapping schema problem* for unit-sized inputs when $q$ is an odd number. First, we divide $m$ inputs into two sets $A$ and $B$. Then, we make $u = \left\lceil \frac{y}{q-\lceil q/2 \rceil} \right\rceil$ disjoint groups of $y$ inputs of the set $A$ such that each group holds $\frac{q-1}{2}$ inputs, lines 1, 2. (Now, each of the groups is considered as a single input that we call the *derived input*.) We do not show the addition of dummy inputs and assume that $u$ is a power of 2. Function $2\_step\_odd\_q(lower, upper)$ recursively divides the derived inputs into two halves, line 4. Function $Assignment(lower, mid, upper)$ (line 9) pairs every two derived inputs and assigns them to the respective reducers (line 11). Each reducer of the last team is assigned using function $Last\_Team(groupA[])$, lines 15, 16.

Note that functions $2\_step\_odd\_q(lower, upper)$, $Assignment(lower, mid, upper)$, and $value\_b(lower, t, mid, upper)$ take two common parameters, namely $lower$ and $upper$ where $lower$ is the first derived input and $upper$ is the last derived input (*i.e.*, $u^{th}$ group) at the time of the first call to functions, line 3. Once all-pairs of the derived inputs are assigned to reducers, line 11, function $Assign\_input\_from\_B(Team[])$ assigns $i^{th}$ input of the set $B$ to all the $\left\lceil \frac{u}{2} \right\rceil$ reducers of $i^{th}$ team, lines 17, 18. After that, algorithm is invoked over inputs of the set $B$ to assign each pair of the remaining inputs of the set $B$ to reducers until every pair to the remaining inputs is assigned to reducers.

*Algorithm correctness.* The algorithm correctness proves that every pair of inputs is assigned to reducers. Specifically, we prove that all those pairs of inputs, $\langle i, j \rangle$ and $\langle i', j' \rangle$, of the set $A$ are assigned to a team whose $i \neq i'$ and $j \neq j'$ (Claim 1). Then that all the inputs of the set $A$ appear exactly once in each team (Claim 2). We then prove that the set $B$ holds $x \leq y-1$ inputs, when $q = 3$ (Claim 3). At last we conclude in Theorem 8 that Algorithm 1 assigns each pair of inputs to reducers.

Note that we are proving all the above mentioned claims for $q = 3$; the cases for $q > 3$ can be generalized trivially where we make $u = \left\lceil \frac{y}{q-\lceil q/2 \rceil} \right\rceil$ derived inputs from $y$ inputs of the set $A$ (and assign in a manner that all the inputs of the $A$ are paired with all the remaining $m-1$ inputs).

**Claim 1** *Pairs of inputs $\langle i, j \rangle$ and $\langle i', j' \rangle$, where $i = i'$ or $j = j'$, of the set $A$ are assigned to different teams.*

**Proof.** First, consider $i = i'$ and $j \neq j'$, where $\langle i, j \rangle$ and $\langle i', j' \rangle$ must be assigned to two different teams. If $j \neq j'$, then both the $j$ values may have an identical value of $lower$ and $mid$ but they must have two different values of $t$ (see lines 13, 14 of Algorithm 1), where $j = lower + t + mid$ or $j = lower + t$. Thus, for two different values of $j$ , we use two different values of $t$, say $t_1$ and $t_2$, that results in an assignment of $\langle i, j \rangle$ and $\langle i', j' \rangle$ to two different teams $t_1$ and $t_2$, (note that teams are also selected based on the value of $t$, $(y - 2 \cdot mid + 1) + t$, see line 11 of Algorithm 1, where for $q = 3$, we have $u = y$). Suppose now that $i \neq i'$ and $j = j'$, where $\langle i, j \rangle$ and $\langle i', j' \rangle$ must be assigned to two different teams. In this case, we also have two different values of $t$, and hence, two different $t$ values assign $\langle i, j \rangle$ and $\langle i', j' \rangle$ to two different teams ($(y - 2 \cdot mid + 1) + t$, line 11 of Algorithm 1).

Hence, it is clear that pairs $\langle i, j \rangle$ and $\langle i', j' \rangle$, where $i \neq i'$ and $j \neq j'$, are assigned to a team. ■

**Claim 2** *All the inputs of the set $A$ appear exactly once in each team.*

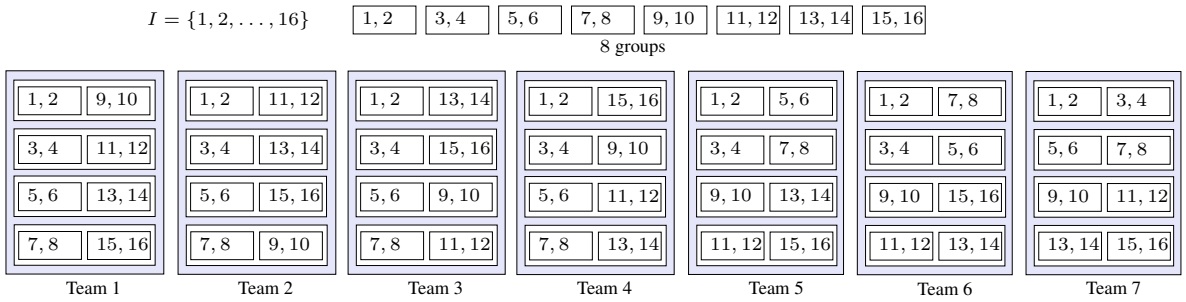
**Proof.** There are the same number of pairs of inputs of the set $A$ and the total number of reducers ($(y-1)\left\lceil \frac{y}{2} \right\rceil$) that can provide a solution to the *A2A mapping schema problem* for the $y$ inputs of the set $A$. Recall that $(y-1)\left\lceil \frac{y}{2} \right\rceil$ reducers are arranged in the form of $(y-1)$ teams of $\left\lceil \frac{y}{2} \right\rceil$ reducers in each team, when $q = 3$. Note that if there is a input pair $\langle i, j \rangle$ in team $t$, then the team $t$ cannot hold any pair that has either $i$ or $j$ in the remaining $\left\lceil \frac{y}{2} \right\rceil - 1$ reducers. For the given $y$ inputs of the set $A$, there are at most $\left\lceil \frac{y}{2} \right\rceil$ disjoint pairs $\langle i_1, j_1 \rangle$, $\langle i_2, j_2 \rangle$, ..., $\langle i_{\lceil y/2 \rceil}, j_{\lceil y/2 \rceil} \rangle$ such that $i_1 \neq i_2 \neq \ldots \neq i_{\lceil y/2 \rceil} \neq j_1 \neq j_2 \neq \ldots \neq j_{\lceil y/2 \rceil}$. Hence, all $y$ inputs of the set $A$ are assigned to a team, where no input is assigned twice in a team. ■

**Claim 3** *When the reducer capacity $q = 3$, the set $B$ holds at most $x \leq y - 1$ inputs.*

**Proof.** Since a pair of inputs of the set $A$ requires at most $q-1$ capacity of a reducer and each team holds all the inputs of the set $A$, an input from the set $B$ can be assigned to all the reducers of the team. In this manner, all the inputs of the set $A$ are also paired with an input of the set $B$. Since there are $y-1$ teams and each team is assigned an input of the set $B$, the set $B$ can hold at most $x \leq y - 1$ inputs. ■

**Throrem 8** *Algorithm 1 assigns each pair of the given $m$ inputs to reducers.*

Algorithm 2, where $q$ is an even number, makes $u = \lceil \frac{2m}{q} \rceil$ disjoint groups, where each group holds $\frac{q}{2}$ inputs. Each of the $u$ groups is considered as a single input that we call the *derived input*. We organize $(u-1) \times \frac{u}{2}$ reducers in the form of $u-1$ *teams* of $\frac{u}{2}$ *players* (or reducers) in each team. These reducers are used to assign each pair of the derived inputs. (Here, a team has each of $m$ inputs, which are occurring only once among reducers of the team, as in case of $q = 5$ see Teams 1-7 in Figure 11; we will not explain this assignment here again.) There are $(u-1) \times \frac{u}{2}$ pairs of the derived inputs (each of size four), and there are $(u-1) \times \frac{u}{2}$ reducers (each of capacity four). Hence, it is possible to assign one pair of derived inputs to one of the reducers.




**Example.** We are given 16 inputs ($I = \{1, 2, \ldots, 16\}$) of size one and $q = 4$. We make $u = \lceil \frac{m}{2} \rceil = 8$ disjoint groups, where each group holds $\frac{q}{2} = 2$ inputs, as follows: $\{1, 2\}$, $\{3, 4\}$, …, $\{15, 16\}$, and now each group becomes a *derived input*. We organize $(u-1) \times \frac{u}{2} = 28$ reducers in the form of 7 *teams* of 4 reducers in each team. These seven teams of four reducers in each team are used to assign each pair of the derived inputs, and there is a solution to the *A2A mapping schema problem* for 16 unit-sized inputs, when $q = 4$.
When $q = 4$, we have two claims as follows: (*i*) each of the given input appears exactly once in each team, and (*ii*) the given algorithm assigns each pair of inputs to at least one reducer. We will prove these claims in the algorithm correctness.

Figure 12: 2-step algorithm (Algorithm 2) when the reducer capacity $q = 4$.

**Proof.** We have $(y-1)\lceil \frac{y}{2} \rceil$ pairs of inputs of the set $A$ of size $q-1$, and there are the same number of reducers; hence, each reducer can hold one input pair. Further, the remaining capacity of all the reducers of each team can be used to assign an input of $B$. Hence, all the inputs of $A$ are paired with every other input and every input of $B$ (as we proved in Claims 2 and 3). Following the fact that the inputs of the set $A$ are paired with all the $m$ inputs, the inputs of the set $B$ is also paired by following a similar procedure on them. Thus, Algorithm 1 assigns each pair of the given $m$ inputs to reducers. ■

**Throrem 9** *Algorithm 1 requires at most* $(\lceil \frac{2m}{(q-1)} \rceil)^2/2$ *reducers and results in at most* $m\big(\lceil \frac{2m}{(q-1)} \rceil - 1\big)$ *communication cost.*

**Proof.** There are at most $x = \lceil \frac{2m}{q-1} \rceil$ groups (derived inputs) of the given $m$ inputs. In order to assign each pair of the derived inputs, each derived input is required to assign to at most $x-1$ reducers. This fact results in at most $m\big(\lceil \frac{2m}{(q-1)} \rceil - 1\big)$ communication cost, and there are at most $(\lceil \frac{2m}{(q-1)} \rceil)^2/2$ pairs of the derived inputs that require at most $(\lceil \frac{2m}{(q-1)} \rceil)^2/2$ reducers. ■

**Algorithm 2: 2-step algorithm when the reducer capacity $q$ is an even number**. We present our algorithm (see Algorithm 2) that handles any even value of $q$. For the sake of understanding and presentation, we first present an example where $q = 4$, namely the case in which a reducer can hold at most four unit-sized inputs, as demonstrated in Figure 12. Note that unlike the algorithm for odd values of $q$ (Algorithm 1) the algorithm for even values of $q$ (Algorithm 2) does not divide the $m$ inputs into two sets. The algorithm consists of two steps, as follows:

1. Group the given $m$ inputs into $u = \lceil \frac{2m}{q} \rceil$ disjoint groups,
2. Organize $(u-1) \times \frac{u}{2}$ reducers, each of capacity $q$, in the form of $u-1$ teams of $\frac{u}{2}$ reducers in each team. Assign every two groups to one of $(u-1) \times \frac{u}{2}$ reducers. We use Lemma 1 for the assignment of every two groups.

Note that we consider each of the $u$ $(= \lceil \frac{2m}{q} \rceil)$ groups as a single input that we call the *derived input*. By making $u$ disjoint groups of the $m$ inputs, we turn the case of any even value of $q$ to a case when $q = 2$ (*i.e.*, a reducer can hold only two unit-sized inputs) and assign every two derived inputs to reducers. In this manner, each input of the set $A$ is assigned with all the remaining $m-1$ inputs.

**Algorithm 2:** 2-step algorithm for an even value of the reducer capacity $q$

**Inputs:** $m$: total number of unit-sized inputs, $q$: the reducer capacity.
**Variables:**
$Team[i, j]$ : represents teams of reducers, where index $i$ indicates $i^{th}$ team and index $j$ indicates $j^{th}$ reducer in $i^{th}$ team. Consider $u = \frac{2m}{q}$. There are $u - 1$ teams of $\lceil \frac{u}{2} \rceil$ reducers in each team.
$groupA[]$ : represents disjoint groups of inputs of the set $A$, where $groupA[i]$ indicates $i^{th}$ group of $\lceil \frac{q}{2} \rceil$ inputs of the set $A$.

1 **Function** $create\_group(m)$ **begin**
2 **for** $i \leftarrow 1$ **to** $u$ **do** $groupA[i] \leftarrow \langle i, i+1 \ldots, i+\frac{q}{2}-1 \rangle, i \leftarrow i+\frac{q}{2}$ ;
3 $2\_step\_even\_q(1, u)$, $Last\_Team(1, \lceil \frac{u-1}{2} \rceil, u)$
4 **Function** $2\_step\_even\_q(lower, upper)$ **begin**
5 **if** $\lfloor \frac{upper-lower}{2} \rfloor < 1$ **then return**;
6 **else**
7 $mid \leftarrow \lceil \frac{upper-lower}{2} \rceil$, $Assignment(lower, mid, upper)$
8 $2\_step\_even\_q(lower, mid)$, $2\_step\_even\_q(mid+1, upper)$

*Algorithm description.* Algorithm 2 provides a solution to the *A2A mapping schema problem* for unit-sized inputs when $q$ is an even number. Recall that Algorithm 1 and Algorithm 2 are almost similar except Algorithm 2 does not create two sets $A$ and $B$. We first make $u = \lceil \frac{2m}{q} \rceil$ disjoint groups of the given $m$ inputs such that each group holds $\frac{q}{2}$ inputs (lines 2), (and consider each of the groups as a single input, the *derived input*). Function $2\_step\_even\_q(lower, upper)$ recursively divides the derived inputs into two halves, lines 4 and 7. Function $Assignment(lower, mid, upper)$ (line 7) is a similar function as given for Algorithm 1 (see line 9 of Algorithm 1) and makes pairs of every two derived inputs. Function $Last\_Team(group[])$ (lines 3) assigns inputs to the last team, *i.e.*, team $u - 1$. Note that function $Last\_Team(group[])$ is same as given for Algorithm 1 (see line 15 of Algorithm 1).

*Algorithm correctness.* We show that every pair of inputs is assigned to reducers. Specifically, Algorithm 2 satisfies two claims, as follows:

**Claim 4** *Pairs of derived inputs $\langle i, j \rangle$ and $\langle i', j' \rangle$, where $i \neq i'$ or $j \neq j'$, are assigned to a team.*

**Claim 5** *All the given $m$ inputs appear exactly once in each team.*

We do not prove Claims 4 and 5. Note that Claim 4 follows Claims 1, where Claims 1 shows that all the pairs of inputs of the set $A$ (in case $q = 3$) and all the pairs of derived inputs of the set $A$ (in case $q > 3$) $\langle i, j \rangle$ and $\langle i', j' \rangle$, where $i \neq i'$ or $j \neq j'$ are assigned to a team. Also, Claim 5 follows Claim 2, where Claim 2 shows that all the inputs of the set $A$ appear in each team only once, while in case of Algorithm 2 the set $A$ is considered as a set of $m$ inputs.

**Throrem 10** *Algorithm 2 assigns each pair of the given $m$ inputs to reducers.*

**Proof.** Since there are the same number of pairs of the derived inputs and the total number of reducers are $((u-1) \times \frac{u}{2})$, it is possible to assign one pair to each reducer that results in all-pairs of the $m$ inputs. ■

**Throrem 11** *Algorithm 2 requires at most $(\lceil \frac{2m}{q} \rceil)^2/2$ reducers and results in at most $m\big(\lceil \frac{2m}{q} \rceil - 1\big)$ communication cost.*

**Proof.** We are creating $\lceil \frac{2m}{q} \rceil$ derived inputs (or groups) of the $m$ inputs. In order to assign a derived input, $i$, with all the derived input, $i$ is required to assign to at most $\lceil \frac{2m}{q} \rceil - 1$ reducers. Hence, there are at most $m\big(\lceil \frac{2m}{q} \rceil - 1\big)$ copies of the $m$ inputs, which results in at most $m\big(\lceil \frac{2m}{q} \rceil - 1\big)$ communication cost. In addition, there are at most $(\lceil \frac{2m}{q} \rceil)^2/2$ pairs of the derived inputs that are assigned to at most $(\lceil \frac{2m}{q} \rceil)^2/2$ reducers, which are organized in the form of $\lceil \frac{2m}{q} - 1 \rceil$ teams of $\lceil \frac{m}{q} \rceil$ reducers in each team. ■

**The *AU method*.** An algorithm to provide a mapping schema for the reducer capacity $p$, where $p$ is a prime number, and $m = p^2$ inputs is suggested by Afrati and Ullman in [2]. We call this approach the *AU method* that meets the lower bounds on the communication cost (Theorem 7 ).

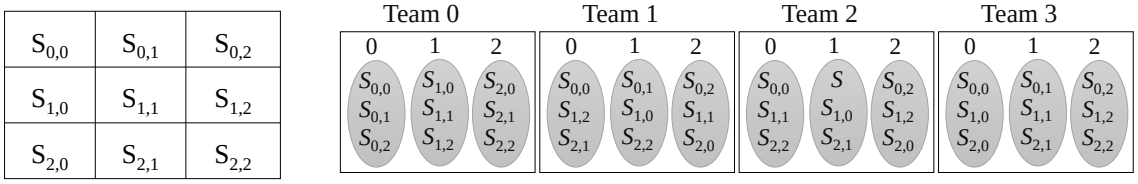



Figure 13: The *AU method* for the reducer capacity $p = 3$ and $m = 9$.

For the sake of completeness, we provide an overview of the *AU method*. Interested readers may refer to [2]. We divide the $m$ inputs into $p^2$ equal-sized *subsets* (each with $m/p^2$ inputs) that are arranged in a $P = p \times p$ square. The subsets in row $i$ and column $j$ are represented by $S_{i,j}$, where $0 \leq i < p$ and $0 \leq i < p$.

According to Theorem 7, at least $\left\lfloor \frac{m}{p} \right\rfloor \left\lceil \frac{m-1}{p-1} \right\rceil$ reducers are required for a solution to the *A2A mapping schema problem* when $m$ equal-sized inputs are given. Hence, in case of $m = p^2$, we require at least $p(p+1)$ reducers. We now organize $p(p+1)$ reducers in the form of $p+1$ *teams* of $p$ *players* (or reducers) in each team. Note that sum of sizes of the inputs in each row and column of the $P$ square is exactly $p$.

The teams are arranged from 0 to $p$, and the reducers are arranged from 0 to $p-1$. We first arrange inputs to the team $p$. Since the sum of the sizes in each column of the $P$ square is $p$, we place one column of the $P$ square to one reducer of the team $p$. Now we place the inputs to the remaining teams. We use modulo operation for the assignment of each subset to each team. The subset $S_{i,j}$ is assigned to a reducer $r$ of each team $t$, $0 \leq t < p$, such that $(i + tj) modulo\, p = r$. An example for $p = 3$ and $m = 9$ is given in Figure 13.

*Algorithm correctness.* Algorithm correctness is given in [2].

*Total required reducers.* The *AU method* uses $p(p+1)$ reducers, which are organized in the form of $p+1$ teams of $p$ reducers in each team, and the total communication cost is $p^2(p+1)$.

Next, we extend the *AU method* to handle more than $p^2$ inputs, when $p$ is a prime number, Algorithms 3 and 4.

**Algorithm 3: The *First Extension of the AU method*.** We extend the *AU method* by increasing the reducer capacity and the total number of inputs, see Algorithm 3. Consider that the *AU method* assigns $p^2$ unit-sized inputs to reducers of capacity $p$, where $p$ is a prime number. We add $l(p+1)$ inputs and increase the reducer capacity to $p + l$ $(= q)$.

In other words, $m$ unit-sized inputs and the reducer capacity $q$ are given. We select a prime number, say $p$, that is near most to $q$ such that $p + l = q$ and $p^2 + l(p+1) \leq m$. Also, we divide the $m$ inputs into two disjoint sets $A$ and $B$, where $A$ holds at most $p^2$ inputs and $B$ holds at most $l(p+1)$ inputs.

For the sake of presentation and understanding of Algorithm 3, we provide an example for $m = 13$ and $q = 4$, see Figure 14. Algorithm 3 consists of six steps, where $m$ inputs and the reducer capacity $q$ are inputs to Algorithm 3, as follows:

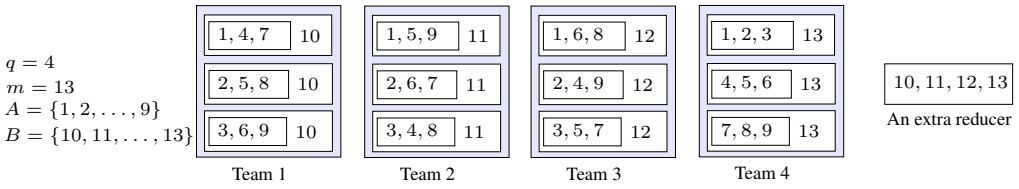
1. Divide the given $m$ inputs into two disjoint sets $A$ of $y = p^2$ inputs and $B$ of $x = m - y$ inputs, where $p$ is a near most prime number to $q$ such that $p + l = q$ and $p^2 + l(p+1) \leq m$.
2. Perform the *AU method* on the inputs of the set $A$ by placing $y$ inputs to $p+1$ teams of $p$ bins in each team, where the size of each bin is $p$.
3. Organize $p(p+1)$ reducers, each of capacity $q$, in the form of $p+1$ teams of $p$ reducers in each teams, and assign $j^{th}$ bin of $i^{th}$ team of bins to $j^{th}$ reducer of $i^{th}$ team of reducers.
4. Group the $x$ inputs of the set $B$ into $u = \left\lceil \frac{x}{q-p} \right\rceil$ disjoint groups.
5. Assign $i^{th}$ group to all the reducers of $i^{th}$ team.
6. Use Algorithm 1 or Algorithm 2 to make each pair of inputs of the set $B$, depending on the case of the value of $q$, which is either an odd or an even number, respectively.

Note that when we perform the above mentioned step 3, we assign each pair of inputs of the set $A$ to $p(p+1)$ reducers, and such an assignment uses $p$ capacity of each reducer. Now, each of $p(p+1)$ reducers has $q - p$ remaining capacity that is used to assign $i^{th}$ group of inputs of the set $B$. In this manner, all the inputs of the set $A$ are assigned with all the $m$ inputs.

*Algorithm description.* Algorithm 3 assigns inputs of the set $A$ to $p(p+1)$ bins (line 2), and then assign the bins to $p+1$ teams of $p$ reducers in each team (line 3). We make $u$ disjoint groups of the inputs of set $B$ such that each group holds $\left\lceil \frac{x}{q-p} \right\rceil$ unit-sized inputs (line 4) and assign $i^{th}$ group to all the reducers of $i^{th}$ team, line 5. Further, we assign each pair of inputs of the set $B$ using Algorithm 1 (line 6) or Algorithm 2 (line 7).

*Algorithm correctness.* The correctness shows that all-pairs of inputs are assigned to reducers. Specifically, we show that each pair of inputs of the set $A$ is assigned to $p(p+1)$ reducers that use only $p$ capacity of each reducer (Claims 6 and 7).

The *first extension of the AU method* (Algorithm 3) for $q = 4$ and $m = 13$ selects $p = 3$ as a near most prime number to $q = 4$. We divide $m = 13$ inputs into two sets $A$ of $y = p^2$ inputs and $B$ of $x = m - p^2$ inputs. We implement the *AU method* on $y$ inputs by placing them to $p + 1$ teams of $p$ bins (each of size $p$) in each team. Then, $j^{th}$ bin of $i^{th}$ team of bins is assigned to $j^{th}$ reducer of $i^{th}$ team of reducers, where we have $p + 1$ teams of $p$ reducers (each of capacity $q$) in each teams. Note that any two bins cannot be assigned to a single reducer, and a team must have each of the inputs of the set $A$ occurring exactly once among the reducers of that team (by following the *AU method*). Now, $q - p$ capacity of each reducer is left. We then make $u = \lceil \frac{x}{q-p} \rceil$ disjoint groups of inputs of the set $B$, and assign $i^{th}$ group to all the reducer of $i^{th}$ team. Further, we implement Algorithms 1 or 2 on the inputs of the set $B$.




**Example.** In case of $q = 4$ and $m = 13$, we implement the *AU method* on 9 inputs of the set $A$ and assign them to 12 bins each of size 3. We then assign each bin to exactly one the $p(p + 1) = 12$ reducers, which are arranged in the form of 4 teams of 4 reducers each of capacity $q = 4$. The remaining unit capacity of all the reducers of a team is used to assign an inputs of the set $B$. Further, we assign all the 4 inputs of the set $B$ to an additional reducer.
In case of $q = 4$ and $9 < m < 14$, we have three claims as follows: (*i*) each of the inputs of the set $A$ appears exactly once in each team, (*ii*) $x \leq m - 9$, and (*iii*) the given algorithm assigns each pair of inputs to at least one reducer. We will prove these claims in the algorithm correctness.

Figure 14: The *first extension of the AU method* (Algorithm 3) for the reducer capacity $q = 4$ and $9 < m < 14$.

Then prove that the set $B$ holds $x \leq m - p^2$ inputs. At last we conclude that Algorithm 3 assigns each pair of inputs to reducers.

**Claim 6** *All the inputs of the set $A$ are assigned to $p(p + 1)$ reducers (each of capacity $q$), and the assignment of the inputs of the set $A$ uses only $p$ capacity of each reducer.*

**Claim 7** *All the inputs of the set $A$ appear in each team exactly once.*

We are not proving Claims 6 and 7 here. Claims 6 and 7 follow the correctness of the *AU method*; hence, all the inputs of the set $A$ are placed to $p + 1$ teams of $p$ bins (each of size $q$) in each team, and the assignment of each such bin only uses $p$ capacity of each reducer. Further two bins cannot be assigned to a reducer because $2 \times p > q$. Claim 7 also follows the correctness of the *AU method*, and hence, all the inputs of the set $A$ appear only once in each team.

**Claim 8** *When the reducer capacity is $q$, the set $B$ holds $x \leq m - p^2$ inputs, where $p$ is a near most prime number to $q$.*

**Proof.** There are $p + 1$ teams of $p$ reducers in each team, and inputs of the set $A$ use $q - p$ capacity of each reducers. Hence, each reducer can hold $q - p$ additional unit-sized inputs. Since inputs of the set $A$ appear in each team (Claim 7), an assignment of $q - p$ additional unit-sized inputs to all the reducers of a team provides pairs of all the inputs of the set $A$ with additional inputs. In this manner, $p + 1$ teams, which hold $p^2$ inputs of the set $A$, can hold at most $(p + 1) \times (q - p)$ additional inputs. Since $p^2 < m \leq p^2 + (p + 1) \times (q - p)$, the set $B$ can hold $x \leq m - p^2$ inputs. ■

**Throrem 12** *Algorithm 3 assigns each pair of inputs to reducers.*

We are not proving Theorem 12 here. The proof of Theorem 12 considers the fact that all the inputs of the set $A$ are paired with each other using the *AU method*, and they are also paired with all the remaining inputs of the set $B$. Further, inputs of the set $B$ will be paired with each other by using Algorithm 1 or 2 (Theorems 8 or 10).

*Total required reducers.* When $l = q - p$ equals to one, we can assign all the $p^2$ inputs of the set $A$ to $p(p + 1)$ reducers of capacity $p + 1$. In this case, an input of the set $B$, where the set $B$ holds at most $p + 1$ inputs, can be assigned to all the reducers of a team. Since there are the same number of teams and inputs in the set $B$, we can assign $i^{th}$ input of the set $B$ to all the reducers of $i^{th}$ teams. In addition, we need one more reducers to assign all the inputs of the set $B$. Thus, we use at least $p(p + 1) + 1$ reducers and the total communication cost is at least $(p^2 + p + 1)q$.

**Algorithm 3:** The *first extension to the AU method*.

**Inputs:** $m$: total number of unit-sized inputs, $q$: the reducer capacity.
**Variables:**
$A$: A set $A$, where the total inputs in the set $A$ is $y = p^2$, where $p$ is a near most prime number to $q$ such that $p + l = q$
$B$: A set $B$, where the total inputs in the $B$ is $x \leq m - y$
$Bin[i, j]$ : represents teams of bin, where index $i$ indicates $i^{th}$ bin and index $j$ indicates $j^{th}$ reducer in $i^{th}$ team. There are $p + 1$ teams of $p$ bins (each of size $p$), in each team.
$Team[i, j]$ : represents teams of reducers, where index $i$ indicates $i^{th}$ team and index $j$ indicates $j^{th}$ reducer in $i^{th}$ team. There are $p + 1$ teams of $p$ reducers (each of capacity $q$) in each team.
$groupB[]$ : represents disjoint groups of inputs of the set $B$, where $groupB[i]$ indicates $i^{th}$ group of $\lceil \frac{x}{q-p} \rceil$ inputs of the set $B$.

**Function** $Assignment(A, B)$ **begin**
  $Bin[] \leftarrow$ The *AU method*$(y, p)$
  **foreach** $(i, j) \in [1, p + 1] \times [1, p]$ **do** $Team[i, j] \leftarrow Bin[i, j]$ ;
  **for** $i \leftarrow 1$ **to** $x$ **do** $groupB[i] \leftarrow \langle i, i + 1 \ldots, i + \frac{x}{q-p} \rangle, i \leftarrow i + \frac{x}{q-p}$ ;
  **foreach** $(i, j) \in [1, p + 1] \times [1, p]$ **do** $Team[i, j] \leftarrow groupB[i]$ ;
  **if** $q$ *is an odd number* **then** Algorithm $1(x, q)$ ;
  **else if** $q$ *is an even number* **then** Algorithm $2(x, q)$ ;

In case of $l > 1$, a single reducer cannot be used to assign all the inputs of the set $B$. Since Algorithm 3 is based on the *AU method*, Algorithm 2, and Algorithm 1, we always use at most $p(p+1) + z$ reducers, where $z$ reducers are used to assign each pair of inputs of the set $B$ based on Algorithms 1 or 2. Thus, the communication cost is at most $qp(p + 1) + z'$, where $z'$ is the maximum communication cost required by Algorithm 1 or 2.

**Algorithm 4: The *Second Extension of the AU method*.** The second extension to the *AU method* (Algorithm 4) handles a case when $m = q^l$, where $l > 2$ and $q$ is a prime number. Recall that the *AU method* assigns only $m = q^2$ inputs, where $q$ is a prime number, to $q(q + 1)$ reducers. We present Algorithm 4 for $m = q^l$, $l > 2$, inputs and the reducer capacity $q$, where $q$ is a prime number. Nevertheless, $m$ inputs that are less than but close to $q^l$ can also be handled by Algorithm 4 by adding dummy inputs such that $m = q^l$, $l > 2$. Algorithm 4 consists of two phases. The first phase is the creation of a *bottom up tree*. A simple example for the bottom-up tree's creation is given for $q = 3$ and $m = 3^4$ in Figure 15.

The second phase is the creation of a *assignment tree*. The assignment tree is created in top-down fashion. Our objective is to assign each pair of inputs to a reducer, where inputs are arranged in the *input columns* of the bottom-up tree. If we can assign each pair of *input columns* (of the bottom-up tree) in the form of $(q \times q)$-sized matrices, then the implementation of the *AU method* on each such matrices results in an assignment of every pair of inputs to reducers. Hence, we try to make pairs of all the *input column*s by creating a tree called the *assignment tree*. The assignment tree for $m = 3^4$ and $q = 3$ is presented in Figure 16.

The assignment tree uses the root node of the bottom up tree, and we implement the *AU method* on the root node that results in $q(q + 1)$ child nodes at level two. Each child node is a $q \times q$ matrix, and the columns of all the $q(q + 1)$ matrices provide all-pairs of the cell values of the root node matrix. At level $i$, the assignment tree has $(q(q + 1))^{i-1}$ nodes, see Figure 17. The height of the assignment tree is $l$, where $(l - 1)^{th}$ level has all-pairs of *input column*s and $l^{th}$ level has a solution to the *A2A mapping schema problem* for $m$ inputs.

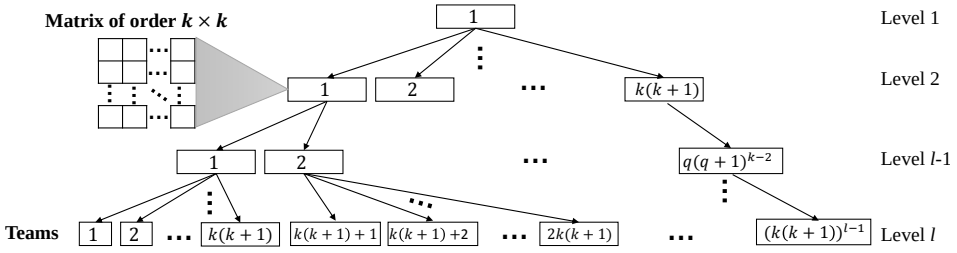
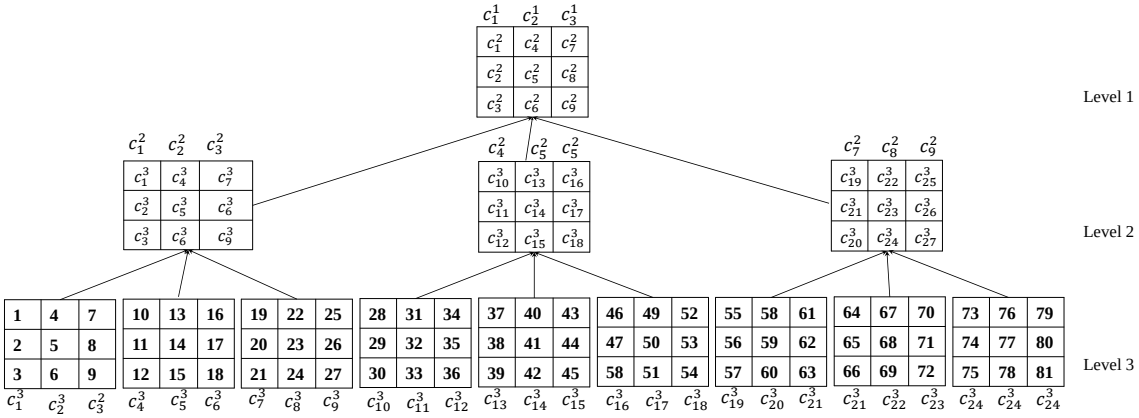


Figure 17: An assignment tree created using Algorithm 4.

*Algorithm correctness.* Algorithm 4 satisfies the following Lemma 2:

**Lemma 2** *The height of the assignment tree is $l$, and $l^{th}$ level of the assignment tree assigns each pairs of inputs to reducers.*

We create a bottom-up tree for $m = q^l$, $l > 2$, and $q = 3$. The height of the bottom up tree is $l-1$, and the last $(l-1)^{th}$ level has $m$ inputs in the form of $\frac{m}{q^2}$ matrices of size $q \times q$. Note that we have $\frac{m}{q}$ columns at the last level, which holds $m$ inputs; and these $\frac{m}{q}$ columns are called the *input columns*. We create the tree in bottom-up fashion, where $(l-2)^{th}$ level has $\frac{m}{q^3}$ matrices, whose each cell value refers to a *input column* of $(l-1)^{th}$ level. We use a notation to refer a column of $i^{th}$ level by $c^i_j$, where $j$ is column index. Note that each column, $c^i_j$, at level $i$ holds $q$ columns $(c^{i+1}_{(j-1)q+1}, c^{i+1}_{(j-1)q+2}, \ldots c^{i+1}_{jq})$ of $(i+1)^{th}$ level. In general, there are $\frac{m}{q^{l-i+2}}$ matrices at level $i$, whose each cell value, $c^i_j$, refers to a column, $c^{i+1}_j$, of $(i+1)^{th}$ level.




**Example.** We are given $3^4$ unit-sized inputs and $q = 3$. The bottom-up tree has height 3. The last level ($(l-1)^{th} = 3^{rd}$) has 81 inputs in the form of $\frac{m}{q^2} = 9$ matrices of size $3 \times 3$. Note that we have $\frac{m}{q} = 24$ columns at $3^{rd}$ level; called the *input columns*. The $l - 2 = 2^{ed}$ level has $\frac{m}{q^3} = 3$ matrices, whose each column, $c^2_j$, refers to $q = 3$ columns $(c^3_{(j-1)q+1}, c^3_{(j-1)q+2}, \ldots c^3_{jq})$ of $3^{rd}$ level. Further, the root node is at level 1, whose each column, $c^1_j$, refers to $q = 3$ columns $(c^2_{(j-1)q+1}, c^2_{(j-1)q+2}, \ldots c^2_{jq})$ of $2^{ed}$ level.

Figure 15: The *second extension of the AU method* (Algorithm 4): Phase 1 – Creation of the bottom-up tree.

**Algorithm 4:** The *second extension to the AU method*.

**Inputs:** $m$: total number of unit-sized inputs.
$q$: the reducer capacity.

**Function** $Assignment(m)$ **begin**
  $bottom\text{-}up\_tree(m)$, $assignment\_tree(root\_node\_of\_bottom\text{-}up\_tree)$

*Total required reducers.* For a given $m = q^l$, $l > 2$, the assignment tree has height $l$, and $l^{th}$ level has $(q(q+1))^{l-1}$ child nodes that are an assignment of each pairs of inputs. Hence, the approach uses $(q(q+1))^{l-1}$ reducers, and the total communication cost is $q \times (q(q+1))^{l-1}$.

### 4.1.3 A hybrid approach

In the previous sections, we provide heuristics for different-sized and equal-sized inputs. The hybrid approach considers both different-sized and equal-sized inputs together. The objective of the hybrid approach to place inputs to two different-sized bins, and then, consider each of the bins as a single input.

Specifically, the hybrid approach uses the previously given heuristic (bin-packing-based heuristic) and Algorithms 1, 2, 3, 4. We divide the given $m$ inputs into two disjoint sets according to their input size, and then, use the bin-packing-based heuristic and Algorithms 1, 2, 3, or 4 depending on the size of inputs. We present two criteria to divide the $m$ inputs into two sets, and based on these sets, we present Algorithm 5.

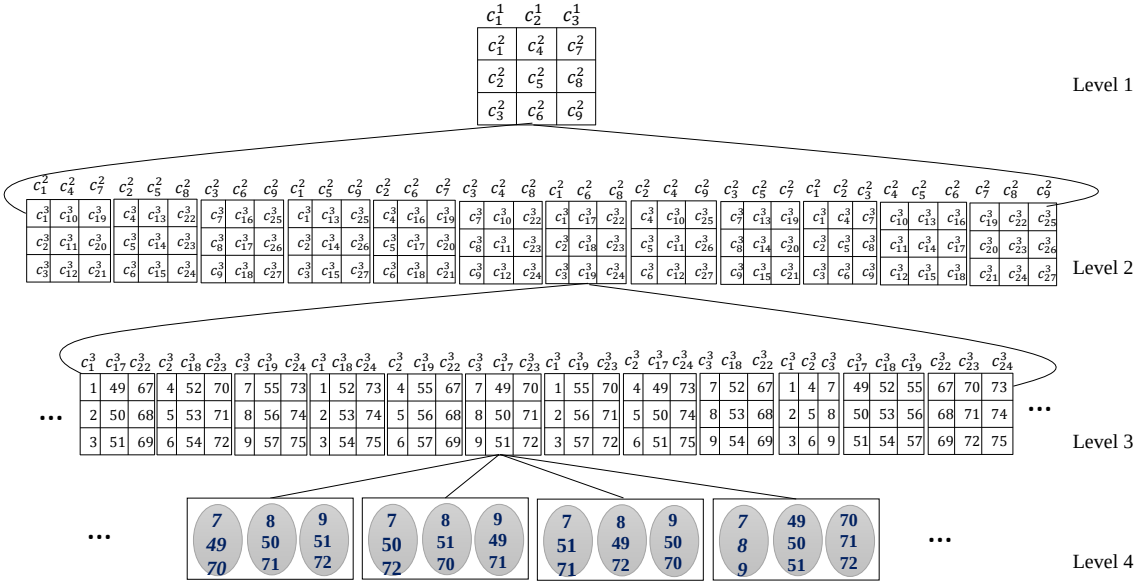
**Algorithm 5.** We divide $m$ inputs into two sets $A$ that holds the input $i$ of size $\frac{q}{3} < w_i \leq \frac{q}{2}$, and $B$ holds all the inputs of sizes less than or equal to $\frac{q}{3}$. Algorithm 5 consists of four steps, as follows:

1. Use the bin-packing-based heuristic to place all the inputs of:

The root node of the bottom-up tree becomes the root node the assignment tree. Recall that the root node of the bottom-up tree is a $q \times q$ matrix. First, consider the root node to understand the working of the *AU method* to create the assignment tree. Consider that each cell value of the root node matrix is of size one, and we have $(q+1)$ teams of $q$ bins (of size $q$) in each team. Our objective to use the *AU method* on the root node matrix is to assign each pair of cell values ($\langle c_x^2, c_y^2 \rangle$) in $q(q+1)$ bins that results in an assignment of every pair of cell values $\langle c_x^2, c_y^2 \rangle$ at a bin.

Now, we create matrices by using these bins (the bins created by the *AU method*'s implementation on the root node) that are holding the indices of columns of the second level ($c_x^2$) of the bottom-up tree. We take each bin and its $q$ indices $c_j^2, c_{j+1}^2, \ldots c_{j+q}^2$. We replace each $c_j^2$ with $q$ columns as: $c_{(j-1)q+1}^3, c_{(j-1)q+2}^3, \ldots c_{jq}^3$ that results in $q(q+1)$ matrices of size $q \times q$, and these $q(q+1)$ matrices become child nodes of the root node. Now, we consider each such matrix separately and perform a similar operation as we did for the root node.

In this manner, the *AU method* creates $(q(q+1))^{i-1}$ child nodes (that are matrices of size $q \times q$) at $i^{th}$ level of the assignment tree, and they create $(q(q+1))^i$ child nodes (matrices of size $q \times q$) at $(i+1)^{th}$ level of the assignment tree. Recall that there are $\frac{m}{q}$ *input columns* at $(l-1)^{th}$ level of the bottom-up tree that hold the original $m$ inputs. The implementation of the *AU method* on each node ($q \times q$-sized) matrix of $(l-2)^{th}$ level of the assignment tree assigns each pair of *input columns* at $(l-1)^{th}$ level of the assignment tree. Further the *AU method*'s implementation on each matrix of $(l-1)^{th}$ level assigns every pairs of the original inputs to $q^l \times (q+1)^{l-1}$ reducers at $l^{th}$ level, which have reducers in the form of $(q(q+1))^{l-1}$ teams of $q$ reducers in each team.




**Example.** We are given $m = 3^4$ unit-sized inputs and the reducer capacity $q = 3$. We take the root node of the bottom-up tree (Figure 15) that becomes the root node of the assignment tree. We implement the *AU method* on the root node and assign each pair of cell values ($c_j^2, 1 \leq j \leq 9$) to a bin of size $q$. Each cell value of the bins ($c_j^2$) is then placed by $q = 3$ columns $c_{(j-1)q+1}^3, c_{(j-1)q+2}^3, \ldots c_{jq}^3$ that results in an assignment of each pair of columns of the second level of the bottom-up tree. For clarity, we are not showing bins. For the next $3^{rd}$ level, we again implement the *AU method* on all 12 matrices at $2^{nd}$ level and get 144 matrices at the third level. The matrices at $3^{rd}$ level are pairs of each *input columns* (of the bottom-up tree). The *AU method*'s implementation on each matrix of $3^{rd}$ level assigns each pair of original inputs to reducers. For clarity, we are only showing all the matrixes and teams at levels 3 and 4, respectively.

Figure 16: The *second extension of the AU method* (Algorithm 4): Phase 2 – Creation of the assignment tree.

(a) the set $A$ to bins of size $\frac{q}{2}$, and each such bin is considered as a single input of size $\frac{q}{2}$ that we call the *big input*. Consider that $x$ big inputs are obtained.
(b) the set $B$ twice, first to bins of size $\frac{q}{2}$, where each bin is considered as a single input of size $\frac{q}{2}$ that we call the *medium input*, and second, to bins of size $\frac{q}{3}$, where each bin is also considered as a single input of size $\frac{q}{3}$ that we call the *small input*. Consider that $y$ medium and $z$ small inputs are obtained.

2. Use $\frac{x(x-1)}{2}$ reducers to assign each pair of big inputs.
3. Use $x \times y$ reducers to assign each big input with each medium input.
4. Use the *AU method*, Algorithm 1, 3, or 4 on the $z$ small inputs, depending on the case, to assign each pair of small inputs.

We present an example to illustrate Algorithm 5 in Figure 18. Note that the use of $\frac{x(x-1)}{2}$ reducers assigns each pair of original inputs whose size between $\frac{q}{3}$ and $\frac{q}{2}$. Also by using $x \times y$ reducers, we assign each big input (or original inputs whose size is between $\frac{q}{3}$ and $\frac{q}{2}$) with each original input whose size is less than $\frac{q}{3}$. Further, the *AU method*, Algorithm 1, 3, or Algorithm 4 assigns each pair of original inputs whose size is less than or equal to $\frac{q}{3}$.

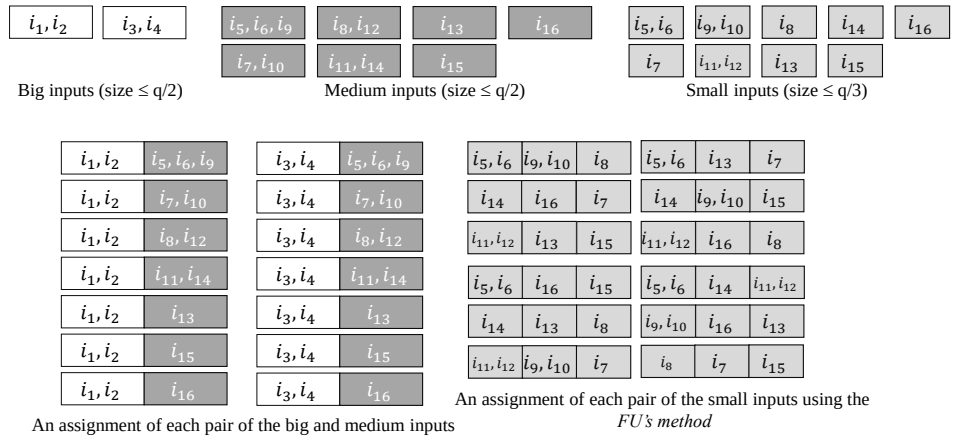



Figure 18: An example to show the working of Algorithm 5. We are given 15 inputs, where inputs $i_1$ to $i_4$ are of sizes greater than $\frac{q}{3}$, and all the other inputs are of sizes less than or equal to $\frac{q}{3}$.

*Algorithm correctness.* The algorithm correctness shows that every pair of inputs is assigned to reducers. Specifically, the algorithm correctness shows that each pair of the big inputs is assigned to reducers, each of the big inputs is assigned to reducers with each of the medium inputs, and each pair of the small inputs is assigned to reducers.

The another criteria is to divide $m$ inputs into two sets $A$ that holds the input $i$ of size $\frac{q}{4} < w_i \leq \frac{q}{2}$, and $B$ holds all the inputs of sizes less than or equal to $\frac{q}{4}$. Following the division of inputs into two sets, we follow a similar approach as Algorithm 5.

## 4.2 A big input of size greater than $\frac{q}{2}$

In this section, we consider the case of an input of size $w_i$, $\frac{q}{2} < w_i < q$; we call such an input as a *big input*. Note that if there are two big inputs, then they cannot be assigned to a single reducer, and hence, there is no solution to the *A2A mapping schema problem*. We assume $m$ inputs of different sizes are given. There is a big input and all the remaining $m-1$ inputs, which we call the *small inputs*, have at most size $q - w_i$. We consider the following three cases in this section:

1. The big input has size $w_i$, where $\frac{q}{2} < w_i \leq \frac{2q}{3}$,
2. The big input has size $w_i$, where $\frac{2q}{3} < w_i \leq \frac{3q}{4}$,
3. The big input has size $w_i$, where $\frac{3q}{4} < w_i < q$.

The communication cost is dominated by the big input. We consider three different cases of the big input to provide efficient algorithms in terms of the communication cost, where the first two cases can assign inputs to almost optimum number of reducers, which results in almost minimum communication cost. We use the previously given heuristic (bin-packing-based heuristic) and Algorithms 1-5 to provide a solution to the *A2A mapping schema problem* for the case of a big input.

A *simple solution* is to use FFD or BFD bin-packing algorithm to place the small inputs to bins of size $q - w_i$. Now, we consider each of the bins as a single input of size $q - w_i$. Let $x$ bins are used. We assign each of the $x$ bins to one reducer with a copy of the big input. Further, we assign the small inputs to bins of size $\frac{q}{2}$, and consider each of such bins as a single input of size $\frac{q}{2}$. Now, we can assign each pair of bins (each of size $\frac{q}{2}$) to reducers. In this manner, each pair of inputs is assigned to reducers.

**The big input of size $\frac{q}{2} < w_i \leq \frac{2q}{3}$.** In this case, we assume that the small inputs have at most $\frac{q}{3}$ size. We use First-Fit Decreasing (FFD) or Best-Fit Decreasing (BFD) bin-packing algorithm, the *AU method*, and Algorithms 3, 4 (Section 4.1.2).

We proceed as follows:

1. First assign the big input with the small inputs.
   (a) Use FFD or BFD bin-packing algorithm to place the small inputs to bins of size $\frac{q}{3}$. Now, we consider each of the bins as a single input of size $\frac{q}{3}$.
   (b) Consider that $x$ bins are used. Assign each of the bins to one reducer with a copy of the big input.
2. Depending on the total number of bins, we use the *AU method*, and Algorithms 3, 4 to assign each pair of the small inputs to reducers.

An example is given in Figure 19, where we place the small inputs to 9 bins of size $\frac{q}{3}$ and assign each of the bins to one reducer with a copy of the big input. Further, we implement the *AU method* on 9 bins to assign each pair of the small inputs.

**The big input of size $\frac{2q}{3} < w_i \leq \frac{3q}{4}$.** In this case, we assume that the small inputs have at most $\frac{q}{4}$ size. We use FFD or BFD bin-packing algorithm and Algorithms 2, 6 (Sections 4.1.2, 4.1.3). We proceed as follows:

1. First assign the big input with the small inputs.
   (a) Use FFD or BFD bin-packing algorithm to place the small inputs to bins of size $\frac{q}{4}$.
   (b) Consider that $x$ bins are used. Assign each of the bins to one reducer with a copy of the big input.
2. Depending on the total number of bins, we use Algorithm 2 to assign each pair of small inputs.

**The big input of size $\frac{3q}{4} < w_i < q$.** In this case, we assume that the small inputs have at most $q - w_i$ size. In this case, we use FFD or BFD bin-packing algorithm and place the small inputs to bins of size $q - w_i$. We then place each of the bins to one reducer with a copy of the big input. Note that, we have not assigned each pair of small inputs. In order to assign each pair of small inputs, we use the bin-packing-based heuristic (Section 4.1.1) or Algorithms 1-5 depending on size of the small inputs.

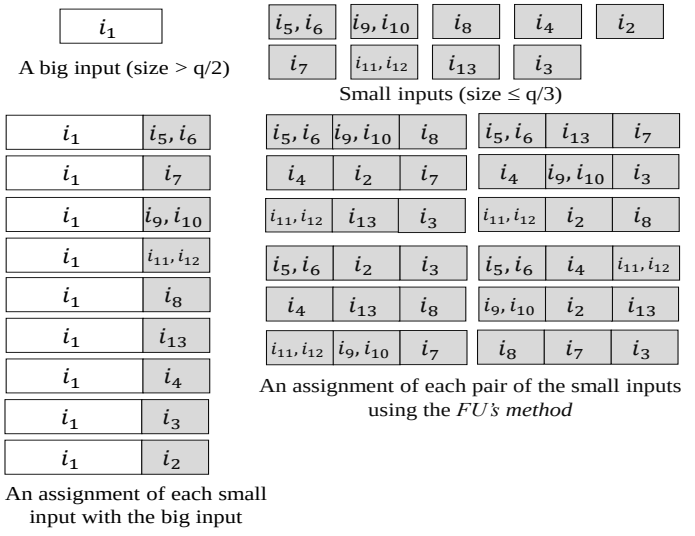



Figure 19: An example to show an assignment of a big input of size $\frac{q}{2} < w_i \leq \frac{2q}{3}$ with all the remaining inputs of sizes less than or equal to $\frac{q}{3}$.

**Throrem 13 (Upper bounds from heuristic)** *For a set of $m$ inputs where a big input, $i$, of size $\frac{q}{2} < w_i < q$ and for a given reducer capacity $q$, $q < s' < s$, an input is replicated to at most $m - 1$ reducers for the A2A mapping schema problem, and the total number of reducers and the total communication cost are at most $m - 1 + \frac{8s^2}{q^2}$ and $(m-1)q + \frac{4s^2}{q}$, respectively, where $s'$ is the sum of all the input sizes except the size of the big input and $s$ is the sum of all the input sizes.*

**Proof.** The big input $i$ can share a reducer with inputs whose sum of the sizes is at most $q - w_i$. In order to assign the input $i$ with all the remaining $m-1$ small inputs, it is required to assign a subset of $m-1$ inputs whose sum of the sizes is at most $q - w_i$. If all the small inputs are of size almost $q - w_i$, then a reducer can hold the big input and one of the small inputs. Hence, the big input is required to be sent to at most $m - 1$ reducers that results in at most $(m-1)q$ communication cost.

Also, each pair of all the small inputs is assigned to reducers (by first placing them to bins of size $\frac{q}{2}$ using FFD or BFD bin-packing algorithm). The assignment of all the small inputs results in at most $\frac{8s'^2}{q^2} < \frac{8s^2}{q^2}$ reducers and at most $\frac{4s'^2}{q} < \frac{4s^2}{q}$ communication cost (Theorem 5). Thus, the total number of reducers are at most $m - 1 + \frac{8s^2}{q^2}$ and the total communication cost is at most $(m-1)q + \frac{4s^2}{q}$. ■

# 5 A Heuristic for the *X2Y Mapping Schema Problem*

We propose a heuristic for the *X2Y mapping schema problem* that is based on bin-packing algorithms. The proposed heuristic assumes a fixed reducer capacity $q$. Two sets, $X$ of $m$ inputs and $Y$ of $n$ inputs, are given. We assume that the sum of input sizes of the sets $X$, denoted by $sum_x$, and $Y$, denoted by $sum_y$, is greater than $q$. We analyze the heuristic on criteria given in Section 4. We look at the lower bounds in Theorems 14 and 15, and Theorem 16 gives an upper bound from a heuristic. The bounds are given in Table 1.

**Throrem 14 (Replication of individual inputs)** *For a set $X$ of $m$ inputs, a set $Y$ of $n$ inputs, and a given reducer capacity $q$, an input $i$ of the set $X$ is required to be sent to at least $\frac{sum_y}{q}$ reducers and an input $j$ of the set $Y$ is required to be sent to at least $\frac{sum_x}{q}$ reducers for a solution to the X2Y mapping schema problem.*

**Proof.** Consider the largest input, say input $i$, of size $w_i$ of the set $X$. The input $i$ can share a reducer with inputs of the set $Y$ whose sum of the sizes is at most $q - w_i$. In order to assign the input $i$ with all the $n$ inputs of the set $Y$, it is required to assign a subset of $n$ inputs whose sum of the sizes is at most size $q - w_i$. Such an assignment results in at least $\lceil \frac{sum_y}{q-w_i} \rceil$ subsets of the $n$ inputs of the set $Y$. We can also conclude $\lceil \frac{sum_y}{q-w_i} \rceil > \frac{sum_y}{q}$. Thus, the input $i$ of the set $X$ is required to be sent to at least $\frac{sum_y}{q}$ reducers to be paired with all the $n$ inputs of the set $Y$.

The argument for $\frac{sum_x}{q}$ is symmetrical. ■

**Throrem 15 (The total communication cost and number of reducers)** *For a set $X$ of $m$ inputs, a set $Y$ of $n$ inputs, and a given reducer capacity $q$, the total communication cost and the total number of reducers, for the X2Y mapping schema problem, are at least $\frac{2 \cdot sum_x \cdot sum_y}{q}$ and $\frac{2 \cdot sum_x \cdot sum_y}{q^2}$, respectively.*

**Proof.** Since an input $i$ of the set $X$ and an input $j$ of the set $Y$ are replicated to at least $\frac{sum_y}{q}$ and $\frac{sum_x}{q}$ reducers, respectively, the communication cost for the inputs $i$ and $j$ are $w_i \times \frac{sum_y}{q}$ and $w_j \times \frac{sum_x}{q}$, respectively. Hence, the total communication cost will be at least $\sum_{i=1}^{m} w_i \frac{sum_y}{q} + \sum_{j=1}^{n} w_j \frac{sum_x}{q} = \frac{2 \cdot sum_x \cdot sum_y}{q}$.

Since the total number of bits to be assigned at reducers is at least $\frac{2 \cdot sum_x \cdot sum_y}{q}$ and a reducer can hold inputs whose sum of the sizes is at most $q$, the total number of reducers must be at least $\frac{2 \cdot sum_x \cdot sum_y}{q^2}$. ■

**Bin-packing-based heuristic for the *X2Y mapping schema problem.*** A solution to the *X2Y mapping schema problem* for different-sized inputs can be achieved using bin-packing algorithms. Let a fixed reducer capacity $q$, two sets $X$ of $m$ inputs, and $Y$ of $n$ inputs are given. The heuristic will not work when a set holds an input of size $w_i$ and the another set holds an input of size greater than $q - w_i$, because these inputs cannot be assigned to a single reducer in common. Let the size of the largest input, $i$, of the set $X$ is $w_i$; hence, all the inputs of the set $Y$ have at most size $q - w_i$. We place inputs of the set $X$ to bins of size $w_i$, and let $x$ bins are used to place $m$ inputs. Also, we place inputs of the set $Y$ to bins of size $q - w_i$, and let $y$ bins are used to place $n$ inputs. Now, we consider each of the bins as a single input, and a solution to the *X2Y mapping schema problem* is obtained by assigning each of the $x$ bins with each of the $y$ bins to reducers. In this manner, we require $x \cdot y$ reducers.

**Throrem 16 (Upper bounds from the heuristic)** *For a bin size $b$, a given reducer capacity $q = 2b$, and with each input of sets $X$ and $Y$ being of size at most $b$, the total number of reducers, the replication of individual input of the set $X$ (resp. $Y$), and the total communication cost, for the X2Y mapping schema problem, are at most $\frac{4 \cdot sum_x \cdot sum_y}{b^2}$, at most $\frac{2 \cdot sum_y}{b}$ (resp. at most $\frac{2 \cdot sum_x}{b}$), and at most $\frac{4 \cdot sum_x \cdot sum_y}{b}$, respectively.*

**Proof.** A bin $i$ can hold inputs whose sum of the sizes is at most $b$. Hence, it is required to divide inputs of the sets $X$ and $Y$ into at least $\frac{sum_x}{b}$ and $\frac{sum_y}{b}$ bins, respectively. Since the FFD or BFD bin-packing algorithm ensures that all the bins (except only one bin) are at least half-full, each bin of size $b$ has at least inputs whose sum of the sizes is at least $\frac{b}{2}$. Thus, all the inputs of the sets $X$ and $Y$ can be placed in at most $\frac{sum_x}{b/2}$ and $\frac{sum_y}{b/2}$ bins of size $b$, respectively.

Let $x$ (=$\frac{2 \cdot sum_x}{b}$) and $y$ (=$\frac{2 \cdot sum_y}{b}$) bins are used to place inputs of the sets $X$ and $Y$, respectively. Since each bin is considered as a single input, we can assign each of the $x$ bins with each of the $y$ bins at reducers, and hence, we require at most $\frac{4 \cdot sum_x \cdot sum_y}{b^2}$ reducers. Since each bin that is containing inputs of the set $X$ (resp. $Y$) is replicated to at most $\frac{2 \cdot sum_y}{b}$ (resp. at most $\frac{2 \cdot sum_x}{b}$) reducers, the replication of individual inputs of the set $X$ (resp. $Y$) is at most $\frac{2 \cdot sum_y}{b}$ (resp. at most $\frac{2 \cdot sum_x}{b}$) and the total communication cost is at most $\sum_{1 \le i \le m} w_i \times \frac{2 \cdot sum_y}{b} + \sum_{1 \le j \le n} w_j \times \frac{2 \cdot sum_x}{b} = \frac{4 \cdot sum_x \cdot sum_y}{b}$. ■

# 6 Conclusion

Two new important practical aspects in the context of MapReduce, namely different-sized inputs and the reducer capacity, are introduced for the first time. The capacity of a reducer is defined in terms of the reducer's memory size. We note that processing time is typically proportional to the memory capacity. All reducers have an identical capacity, and any reducer cannot hold inputs whose input sizes are more than the reducer capacity. We demonstrated the importance of the capacity aspect by considering two common mapping schema problems of MapReduce, *A2A mapping schema problem* – every two inputs are required to be assigned to at least one common reducer – *X2Y mapping schema problem* – every two inputs, the first input from a set $X$ and the second input from a set $Y$ – is required to be assigned to at least one common reducer.

Unfortunately, it turned out that finding solutions to the *A2A* and the *X2Y* mapping schema problems that use the minimum number of reducers is not possible in polynomial time. On the positive side, we present near optimal heuristics for the *A2A* and the *X2Y* mapping schema problems.

# References


[1] F. N. Afrati, A. D. Sarma, S. Salihoglu, and J. D. Ullman. Upper and lower bounds on the cost of a map-reduce computation. *PVLDB*, 6(4):277–288, 2013.

[2] F. N. Afrati and J. D. Ullman. Matching bounds for the all-pairs MapReduce problem. In *IDEAS*, pages 3–4, 2013.

[3] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proceedings of the 16th international conference on World Wide Web*, pages 131–140, 2007.

[4] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for NP-hard problems. chapter Approximation algorithms for bin packing: a survey, pages 46–93. PWS Publishing Co., 1997.

[5] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[7] M. T. Goodrich. Simulating parallel algorithms in the MapReduce framework with applications to parallel computational geometry. *CoRR*, abs/1004.4708, 2010.

[8] D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

[9] D. R. Karger and J. Scott. Efficient algorithms for fixed-precision instances of bin packing and euclidean tsp. In *APPROX-RANDOM*, pages 104–117, 2008.

[10] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *SODA*, pages 938–948, 2010.

[11] A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, and E. Upfal. Space-round tradeoffs for MapReduce computations. In *ICS*, pages 235–244, 2012.

[12] A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*. Cambridge Unversity Press, 2011.

[13] J. D. Ullman. Designing good MapReduce algorithms. *ACM Crossroads*, 19(1):30–34, 2012.

[14] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using MapReduce. In *SIGMOD Conference*, pages 495–506, 2010.

[15] C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In *Proceedings of the 17th international conference on World Wide Web*, pages 131–140, 2008.