Spring 2020

A Systematic Approach to Compute All Cyclic Quorum Sets

Yiming Bian
Iowa State University

A Systematic Approach to Compute All Cyclic Quorum Sets

Yiming Bian, *Member, IEEE,* and Arun K. Somani, *Life Fellow, IEEE*

***Abstract*— Use of quorum sets and cyclic quorum sets have proved to be a very useful method to achieve efficient initial data placement and data communication in distributed computation and communication systems. For example, in all-pairs data interaction problems, cyclic quorum sets can be used to avoid communication completely after initial data placement. Searching for all possible cyclic quorum sets for a given number of objects, *P*, is a task that requires massive computations. This is known to be a hard problem and no time complexity reduction method has been found thus far. In this paper, we try to optimize the search process by avoiding the search space where it is not possible to find a feasible cyclic quorum base set. By studying all possible cyclic quorum sets for given *P*, we develop insight into the properties of all quorum sets that helps us to reduce the total number of computations significantly compared to that adopting the naïve exhaustive search. We notice that as *P* grows, better performance could be achieved.**

Index Terms: **All-Pairs Problem, Searching All Cyclic Quorum Sets, Feasible Quorum Base Sets, Search Space Optimization.**

I. Introduction

A good number of applications are using a large amount of data in the current wave of data-based decision-making processes. One of the most significant big data problems arises when many data elements need to interact with each other to derive meaningful conclusions. Such interactions among all data elements occur frequently in many applications, such as deduplication detection, n-body problem, etc., It is known as all-pair interaction problem, and solutions requires managing a large amount of data and their processing.

Since all-pair interaction problems in the real world are often so large-scaled that it is almost impossible to solve them on a single machine due to its disadvantages such as limited computing ability and restricted storage space. Therefore, a distributed system with multiple nodes offer a promising solution to meeting the challenge. Now the focus turns to the data allocation strategy within a distributed system because it makes little sense to have multiple nodes and store all the relevant data at every node. The question that arises then is how to allocate and process the data efficiently? Driscoll et al. [1] proposed a communication-optimal n-body algorithm in 2013. Later in 2018, Kleinheksel et al. [2] applied quorum theory to manage data distribution and achieve all-to-all interactions in to solve an animal gene problem in a systematic manner. Specifically, their solution uses cyclic quorum sets, which solves data distribution and communication problem in a very elegant manner.

In the previous work by Wai-Shing Luk et al.[3], for distributed system with the number of nodes from 4 to 111, one feasible cyclic quorum base set have been computed for each case. The idea of quorums first introduced in Maekawa's algorithm [4]. Without an exception, all researchers mentioned in their papers that searching for those solutions cost them a huge amount of time because so far, the only algorithm to tackle this problem is to deploy an exhaustive search.

Without getting into any detail of exhaustive search, this is a daunting task because everyone knows its notoriety of poor performance. Unfortunately, we are not going to propose a fancy new lower-complexity algorithm that boosts the time complexity to linear, quadratic, or even near. Our idea however is to explore and avoid some search spaces because there would never be a feasible cyclic quorum base set in those spaces. Moreover, we are curious about how many feasible cyclic quorum base sets could be found for a given size of a distributed system. Our approach not only decreases the time to find the first feasible base set but also make it possible to find all feasible base sets in a reasonable amount of time. With our results, we believe that new research using the cyclic quorum sets will thrive. For example, it will enable us to answer questions like is there a cyclic quorum base set that delivers better performance compared to any other feasible cyclic quorum base sets? How these cyclic quorum sets affect the fault-tolerance property of the distributed system due to the presence of duplicated data among them? Many such other questions can be thought of.

I.1 Contributions

We make the following contributions.

- We drastically decrease the amount of computations required during exhaustive search.
- We explore, characterize, and prove some useful properties of special cyclic quorum sets.
- We develop systematic approach to search for all cyclic quorum sets.
- We verify the optimality and correctness of previously published cyclic quorum sets for $N = 4$ to 111.

I.2 Outline of the paper

The outline of our idea is first we prove all feasible sets are equivalent to a kind of set that has a specific pattern. Then only search for that kind of sets for feasible ones instead of every set in the space in order to cut the total number of searches. The rest of the paper is organized as follows.

Authors are with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IS 50011
e-mail: ybian@ iastate.edu, arun@iastate.edu

Section II and section III provide prerequisite knowledge about all-pairs problem and cyclic quorum sets. In section IV we introduce the idea of set equivalence and feasibility check measure. Then three optimization strategies are proposed with proofs and an additional optimization for some special cases is also given. Section V is the analysis of both feasibility check function and whole search algorithm. In Section VI, we present part of the results as examples to show the great reduction to number of searches with our optimization strategies. In section VII, we provide the conclusion and expectation on future work.

II. All-Pairs Problem

II.1 General All-Pairs Problem Definition

There are multiple approaches proposed to compute all-pairs problem in a distributed manner. The easiest but possibly the worst way is giving all data to every node if every node has enough memory to store all the data. Then all nodes are assigned to compute specific computations out of the following loop that represents the pseudocode for a general all-pairs algorithm:

```
Given: one-dimensional array $D$
for $i \leftarrow 0$ to $length(D)-1$ do
    for $j \leftarrow i+1$ to $length(D)-1$ do
        Compute Interaction of $(d_i, d_j)$
    end for
end for
```

(d_0, d_1) (d_0, d_2) (d_0, d_3)
(d_1, d_2) (d_1, d_3)
(d_2, d_3)

Fig.1. All-pairs of four data elements.

To avoid shared computing resources being abused like this, several researchers [2] have developed frameworks to solve all-pairs classification of algorithms and show performance improvement. We discuss some of them in the following.

Given a set $D=\{d_0, d_1, d_2, \ldots, d_{m-1}\}$ with size m, the interactions among all elements form the job set. For example, for $D=\{d_0, d_1, d_2, d_3\}$, the job set is shown in Fig.1. The size of job set is equivalent to the total number of interactions among m elements, which is $\binom{m}{2} = \frac{m(m-1)}{2}$.

II.2 Distributed All-Paris Problem

An alternative way to address the distributed all-pairs problem is to divide a given data set D into P mutually exclusive subsets and process in a distributed system with P nodes. Thus, $D=D_0+D_1+\ldots+D_{P-1}$ and subsets satisfy the following properties:

$$\begin{cases} D = D_0 \cup D_1 \cup \cdots \cup D_{p-1} \\ \forall i \neq j,\ D_i \cap D_j = \emptyset \end{cases}$$

After the partition, each node in the distributed system is assigned a subset. To realize interaction computations, there should be communication among nodes. A simple way to accomplish this is adopting round robin strategy [5] to systematically move data around from each node to other nodes so that full computations can be accomplished.

III. Cyclic Quorum Sets

To avoid massive communication as is inherent in round robin communication above, fortunately, for such problems it is possible to initially assign several data subsets by using cyclic quorum set to a node and perform all interactions without any further communication [2].

Suppose there is a distributed system with P nodes and the data set D is evenly divided into P subsets $D_0, D_1, \ldots, D_{p-1}$. Each node is assigned n subsets and all these subsets form a quorum base set, which is denoted by S_i *(for $i = 0, 1, \ldots, P-1$)*. The basic property of a cyclic quorum set is that when we know one base set, we can derive all other base sets. The job for each node is to perform interactions among the data elements of all assigned subsets. After every node finishes its job, the cyclic quorum set is optimal if the union of all outcomes cover all possible interactions among all subsets, possibly with some or without any redundant computations [2]. In some special cases, it is possible that all nodes' jobs are mutually exclusive. In these cases, cyclic quorum sets form a minimal optimal set as there is no redundancy in computation.

An optimal cyclic quorum set has four properties: i) Any two base sets have non-empty intersection; ii) Each base set performs equal work; iii) Each base set has equal workload responsibility [4]; and iv) Together they satisfy the all-pairs computation needed. Maekawa [4] proposed the second and third properties for quorums used for distributed algorithms and proved the lower bound of the size of a base set is $\sqrt{P}$. Kleinheksel et al. [2] proposed the fourth property to apply cyclic quorum sets to address all-pairs problem.

Take $P=7$ as an example. Original data set D is uniformly divided into seven subsets, namely $D_0, D_1, \ldots, D_6$. Our goal is to accomplish all twenty-one interactions, like in the handshake model [6], shown in Fig. 2.

(D_0, D_1) (D_0, D_2) (D_0, D_3) (D_0, D_4) (D_0, D_5) (D_0, D_6)
(D_1, D_2) (D_1, D_3) (D_1, D_4) (D_1, D_5) (D_1, D_6)
(D_2, D_3) (D_2, D_4) (D_2, D_5) (D_2, D_6)
(D_3, D_4) (D_3, D_5) (D_3, D_6)
(D_4, D_5) (D_4, D_6)
(D_5, D_6)

Fig.2. All the interactions of subsets with P=7

$S_0=\{D_0, D_1, D_2\}$	$S_0=\{D_0, D_1, D_3\}$
$S_1=\{D_1, D_2, D_3\}$	$S_1=\{D_1, D_2, D_4\}$
$S_2=\{D_2, D_3, D_4\}$	$S_2=\{D_2, D_3, D_5\}$
$S_3=\{D_3, D_4, D_5\}$	$S_3=\{D_3, D_4, D_6\}$
$S_4=\{D_4, D_5, D_6\}$	$S_4=\{D_4, D_5, D_0\}$
$S_5=\{D_5, D_6, D_0\}$	$S_5=\{D_5, D_6, D_1\}$
$S_6=\{D_6, D_0, D_1\}$	$S_6=\{D_6, D_0, D_2\}$
Fig.3a. P=7, n=3	Fig.3b. P=7, n=3

The minimum base set size needed is $n=3$ ($\sqrt{P}$). The seven data subsets (D_i) can be allocated to each node in a cyclic manner as shown in Fig. 3a and 3b in two different ways. The group of sets in Fig. 3a is not optimal because the intersection between S_0 and S_3 is empty. Also, the interaction between D_0

and D_3 is not performed by any node, thus all-pairs property is missing.

Fig. 3b on the right shown a little variation of distribution and in this case, all properties are satisfied. Hence, Fig. 3b represents an optimal cyclic quorum set.

To avoid confusion, an optimal cyclic quorum set is a group of P sets that satisfy all four properties. Each of them is called a base set because the remaining P-1 sets can be derived from it and each of them has a size of n. Also, in the rest of the paper, cyclic quorum sets will be referred to as CQS.

In the case for $P=7$, we have $n=3$, where the lower bound is reached. However, the lower bound is not always achieved for an optimal CQS. A simple example can be constructed for $P=4$ and the lower bound of n in this case is 2.

For $n=2$, two data subsets (D_i) can be allocated to each node in a cyclic manner and form a CQS as shown in Fig. 4a. In fact, no matter how to distribute data subsets, at most four out of six interactions are performed, which violates the all-pairs property. Hence, this set is not optimal with any possible assignments under the condition $n=2$, proving that the lower bound is not achieved.

$S_0 = \{D_0, D_1\}$	$S_0 = \{D_0, D_1, D_2\}$
$S_1 = \{D_1, D_2\}$	$S_1 = \{D_1, D_2, D_3\}$
$S_2 = \{D_2, D_3\}$	$S_2 = \{D_2, D_3, D_0\}$
$S_3 = \{D_3, D_0\}$	$S_3 = \{D_3, D_0, D_1\}$

Fig.4a. P=4, n=2 Fig.4b. P=4, n=3

If given $n=3$, three data subsets (D_i) are allocated to each node in a cyclic manner and form a CQS as shown in Fig. 4b. This distribution not only accomplishes all-pair interactions, but computes each of them twice, providing a redundancy level of two [7]. Although such computation redundancy may impact the performance of system, this CQS satisfy all four properties, hence, we say it is optimal even when the lower bound is not achieved.

IV. Computing All Cyclic Quorum Sets

In distributed system, quorum sets are used in many ways such as distributed consensus development problem. The above discussion suggests that applying CQS can perfectly solve the distributed all-pairs problem. Therefore, searching for CQS is important. This gives rise to several questions such as how to search for an optimal CQS? Is a given quorum set optimal? How many feasible base sets exist to generate an optimal CQS? Is one CQS better than other CQSs? What is the relationship among different CQSs? If we know one CQS, can we derive other CQSs? The previous research found one base set for a given value of P. Is there an algorithm that can find all such base sets? Since one CQS may be preferred over the others for a variety of reason, we may be interested in finding all such base sets and study their properties. Our goal in this paper is to try to answer most of these questions.

For simplicity, in the rest paper, all data subsets in CQSs are replaced by their indices. Now the elements of a base set are numbers instead of using the name of the data subsets. For example, $S_0=\{D_0, D_1, D_3\}$ is written as $S_0=\{0, 1, 3\}$ where the numbers are indices of the data set.

IV.*1 Equivalent sets*

As mentioned in the previous section, a CQS contains P quorum base sets and the whole set could be derived from any one of them. Therefore, our interest in the search is to find only one of the base sets. The rest of them are also base sets, but they are redundant, and they are equivalent. Also, any permutation of these sets is equivalent. For example, in Fig. 3b, S_0, S_1, S_2, S_3, S_4, S_5 and S_6 are equivalent. Apart from this, if we are given a set $\{0, 4, 5\}$, it is also equivalent to anyone of them because it is a cyclic permutation of S_4.

Definition 1. ***Standard form***: If elements of a base set are in ascending order, we say that it is in standard form.

Definition 2. ***Interest set***: If a base set is in standard form and the starting two elements are $\{0, 1\}$, then we call it an interest set.

Theorem 1. Given a feasible set, there exist an equivalent set that is an interest set.

Proof. We can generate the cyclic quorum set using the given set. Since the given set is feasible, the cyclic quorum set is optimal, which satisfies all-pairs requirement. Therefore, the interaction between 0 and 1 must exist in a base set. Let's say, S_0. Then S_0 is equivalent to the given set. Put S_0 in standard form then we get an interest set equivalent to the given feasible set. Hence, this theorem holds.

□

IV.*2 Feasibility Check*

Before searching, the first problem we must address is how to decide if a CQS is optimal or not. We call this the feasibility check problem. In fact, we only check one of the base sets instead of the whole CQS. In [2], authors introduced a theorem that states the property about *(P, n)-difference set* to check the feasibility.

The (P, n)-difference set is defined as follow: for a given set $A=\{a_0 .., a_{n-1}\}$ and value of P, all integers 0, ..., $(P-1)$ must be constructed by the differences of all possible pairs of elements in set A modulus P. For example, given $P=7$, $n=3$, and a base set $A=\{0, 1, 2\}$, the values of differences between all pairs modulus P are shown in Fig. 5a. It is an invalid difference set because 3 and 4 are missing. In Fig. 5b, it shows the values of differences modulus P corresponding to $A=\{0, 1, 3\}$ and all integers from 0 to 6 are present. Therefore, $A=\{0, 1, 3\}$ is a valid $(7,3)$-difference set.

(ai-aj) *mod* 7		ai		
		0	1	2
aj	0	0	1	2
	1	6	0	1
	2	5	6	0

Fig.5a. P=7, n=3

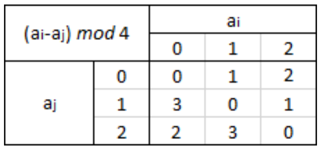
(ai-aj) *mod* 7		ai		
		0	1	3
aj	0	0	1	3
	1	6	0	2
	3	4	5	0

Fig.5b. P=7, n=3

When a quorum base set has a valid (P,n)-difference set, it is feasible to generate an optimal CQS.

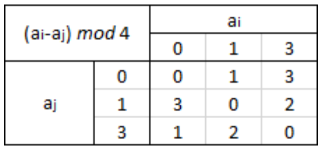
Now, we are going to show that it is possible that feasible base set is not unique.

While in the case where $P=4$, $n=3$, the *(4,3)-difference set* of the base set $\{0, 1, 2\}$ is shown in Fig.6a. And the *(4,3)-difference set* of $\{0, 1, 3\}$ is shown in Fig.6b. As these are two valid difference sets, both base sets are feasible to generate an optimal cyclic quorum set.



(ai-aj) *mod* 4		ai		
		0	1	2
aj	0	0	1	2
	1	3	0	1
	2	2	3	0

Fig.6a. P=4, n=3



(ai-aj) *mod* 4		ai		
		0	1	3
aj	0	0	1	3
	1	3	0	2
	3	1	2	0

Fig.6b. P=4, n=3

Hence, it is safe to conclude that feasible base set is not necessarily unique. Also, since in this case, all elements appear twice in both difference sets, these two cyclic quorum sets are said to have a redundancy of two [7].

VI. 3 Search Optimizations

After knowing there may be multiple feasible base sets for a given configuration of distributed system and how to tell whether a base set is feasible or not, it is time to focus on the search process and the start point of our optimization is naïve exhaustive search. In the rest paper, indexing follows C-language style, which counts from *0* instead of *1*.

4.31 Naïve Exhaustive Search

Without any tricks, the naïve exhaustive search strategy is traversing all possible sets and run feasibility check in each case. So, the total number of computations is P^n. Here is the pseudocode for naïve search.

```
Given $P$, $n$
  for $a_0 \leftarrow 0$ to $P$-$1$ do
    for $a_1 \leftarrow 0$ to $P$-$1$ do
      …
      for $a_{n-1} \leftarrow 0$ to $P$-$1$ do
        feasiblity_check()
      end for
      …
    end for
  end for
```

4.32 Optimization Strategies

As mentioned before, to optimize the search process, the only thing can be done is avoiding specific spaces. After study of the raw results generated by naïve search, we explore several optimization strategies that drastically reduce number of searches.

Theorem 2. ***Optimization #1***: Loop a_0 and a_1 run once.

Proof. Due to Theorem *1*, it is safe to conclude that all feasible sets have equivalent interest set, in other words, if all interest sets are found, all feasible sets are found. Because they can be derived from interest sets by permutation or a cyclic-manner generation. By the definition of interest set, its first two elements are *0* and *1*. Therefore, the first two loops only need to consider values *0* and *1*, respectively.

□

Theorem 3. ***Optimization #2***: For the indices ranges of rest *n-2* loops, we have $a_i \in [a_{i-1}+1,\ P\text{-}n+i]$.

Proof. Since we are searching interest sets, whose elements are in ascending order. The starting index of a loop should be one greater than that of its outer loop and the ending index of this loop should be one fewer than its inner loop. Thus, for the range of a_i, its starting index is $a_{i-1}+1$. The ending index of a_{n-1} is *P-1*, so the max value of a_{n-2} is *P-2*, so on and so forth. Therefore, the ending index of a_i is *P-n+i*. Hence, this theorem holds.

□

4.33 Pairing Property

By observing all feasible base sets for $P = 4\ldots,111$, we found a very interesting phenomenon. If we are given one feasible base set, we can derive another base set by using name remapping. We call this a paring property. Due to this property, we developed a novel search pattern that reduces the total number of searches to almost in half.

Theorem 4. Given *P* and *n*, two base sets $\{a_0, a_1, a_2, a_3, \ldots, a_{n-1}\}$ **and** $\{P+1\text{-}a_1, P+1\text{-}a_0, P+1–a_{n-1}, P+1–a_{n-2}, \ldots, P+1–a_2\}$ **have the same feasibility and we call them two paired sets.**

Proof. Let *X* and Y denote set $\{a_0, a_1, a_2, a_3, \ldots, a_{n-1}\}$ and set $\{P+1\text{-}a_1, P+1\text{-}a_0, P+1–a_{n-1}, P+1–a_{n-2}, \ldots, P+1–a_2\}$ respectively, then we have $f{:}X \longrightarrow Y,\ \ f(x) = (P + 1 – x) \bmod P$ and $g{:}Y \longrightarrow X,\ g(y) = (P + 1 – y) \bmod P$. Since for any $i \neq j$, $a_i – a_j = (P + 1 – a_j) – (P + 1 – a_i)$, these two sets have the same *(P,n)*-difference set. Therefore, either both sets are feasible or not feasible. Hence, the theorem holds.

□

Definition 3. ***Self-paired***: If the paired set of a base set is itself, we call this base set self-paired. For example, given $P=5$, $n=3$, one interest set is $\{0, 1, 3\}$ and its paired set is $\{0, 1, 5+1–3\}=\{0, 1, 3\}$, which is the same set. We call this set self-paired.

Since the space we search does not have the outermost two loops, we only focus on $\{a_2, a_3, \ldots, a_{n-1}\}$ in the rest of this section. Therefore, the current search space has *n-2* loops and the ranges of each loop index as shown below.

$$\begin{cases} a_2 \in [2, P-6-n] \\ a_3 \in [a_2+1, P-5-n] \\ \quad\ldots \\ a_{n-1} \in [a_{n-2}+1, P-1] \end{cases}$$

For example, if given $P=8$, $n=4$, we have $a_2 \in [2, 6]$ and $a_3 \in [a_2+1, 7]$. Traversal is designed as below.

```
Given $P=8$, $n=4$
  $a_0=0$, $a_1=1$
  for $a_2 \leftarrow 2$ to $6$ do
    for $a_1 \leftarrow a_2+1$ to $8$ do
      feasiblity_check()
    end for
  end for
```

In Table 1. we list all searches in pairs.

$\{a_2, a_3\}$	$\{P+1-a_3, P+1-a_2\}$
(2, 3)	(6, 7)
(2, 4)	(5, 7)
(2, 5)	(4, 7)
(2, 6)	(3, 7)
(2, 7)	(2, 7) – self
(3, 4)	(5, 6)
(3, 5)	(4, 6)
(3, 6)	(3, 6) – self
(4, 5)	(4, 5) – self

Table.1. Paired sets

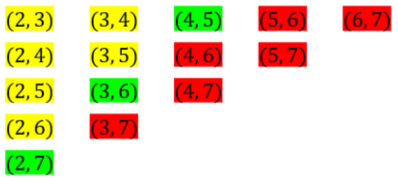


Fig.8. P=8, n=4

To visualize the pairing process, put one set in yellow area, its paired set in red area and self-paired sets in green area as we traverse all sets in sequence. We put all sets in difference colors as can be seen on the right. So, the problem turns to, how to divide the whole search space in half to take the advantage of pairing property. If this is realized, we only traverse half of them, and the other half is automatically searched.

The solution is to search only the yellow and green areas. Due to the existence of self-paired interest sets, we cannot always divide the whole space in exactly two equal halves. It is often the case that we have to search slightly more than half of the space. So, in this case, the range of a_2 is $[2, 4]$ and that of a_3 is $[a_2+1, 9-a_2]$. Both ranges are contiguous, and this is what we want because it makes the search space continuous and easy to implement in a program.

With this basic idea in mind, let's analyze an example where $P=14$, $n=5$. All possible searches are colored as shown in Fig. 9. There is no green area, which means no self-paired sets in this case. And the index ranges are not as regular as in the previous example, you have access to the detail of this case in appendix, the ranges of index do not have a specific pattern, which makes the implementation troublesome. At some point, the iteration should be skipped because its paired area has already been searched. Or we need to repeat part of the search to maintain the smoothness of traversal.

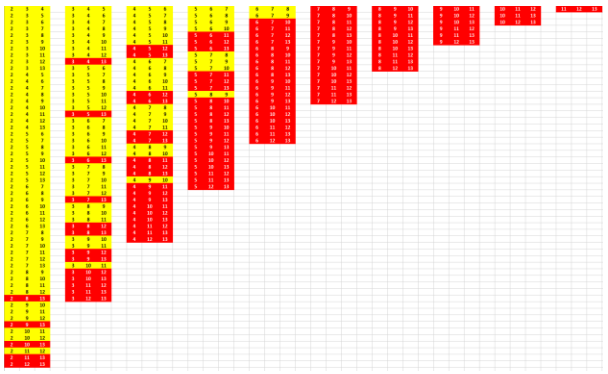


Fig.9. P=14, n=5

Is there a strategy that only checks those needed while makes no compromise on the performance? Fortunately, we figured out a general searching pattern, which divides the discontinuous range of index into two exclusive parts and for each part, the traversal is contiguous, then we combine two results together. Here is how this strategy works.

Step1: Color all searches.
Step2: Change the sets $\{a_2, \ldots, P+1-a_2\}$ in yellow area to green.
Step3: Traverse yellow area and green area as in theorem 5.

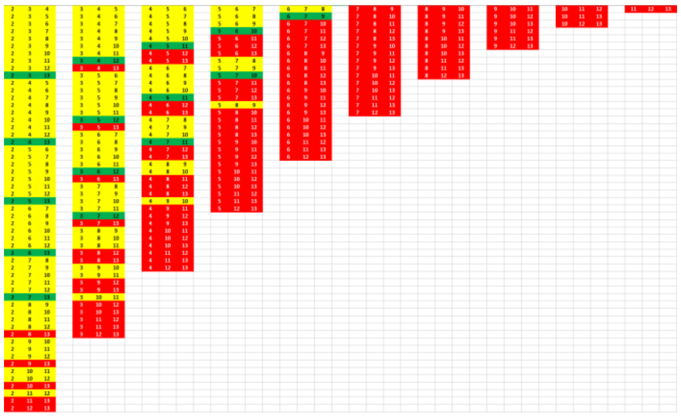


Fig.10. New Colorway

Theorem 5. ***Optimization #3***
Traversing yellow area:

$$\#\ of\ loops = n-2$$

$$\begin{cases} a_2 \in \left[2, \left\lfloor \frac{P-n+3}{2} \right\rfloor\right] \\ a_3 \in [a_2+1, P-a_2+4-n] \\ \quad\ldots \\ a_{n-2} \in [a_{n-3}+1, P-a_2-1] \\ a_{n-1} \in [a_{n-2}+1, P-a_2] \end{cases}$$

Traversing green area:

$$\#\ of\ loops = n-3$$

$$\begin{cases} a_2 \in \left[2, \left\lfloor \frac{P+4-n}{2} \right\rfloor\right] \\ a_3 \in \left[a_2+1, \left\lfloor \frac{P+4-n}{2} \right\rfloor + 1\right] \\ \quad\ldots \\ a_{n-2} \in \left[a_{n-3}+1, \left\lfloor \frac{P+4-n}{2} \right\rfloor + n - 4\right] \end{cases}$$

Proof. The way to layout all searches is putting all sets with same first elements(a_2) in one column. In the same column, there are some paired sets. So, one of them will be in yellow area while the other in red. And this is how continuity is broken. All these column-wise paired sets satisfy $a_2+a_{n-1}=P+1$. So, theoretically, first half of these are in yellow and the latter half are in red. In some cases where exist self-paired sets, they are in green and right in the middle of these column-wise paired sets.

Before pairing, the whole search space is continuous. But this continuity is broken by the red half of column-wise paired sets. In order to retain the contiguous search space, we put the yellow half of column-wise paired sets in green because they share the same pattern and after getting rid of them, the rest yellow area are contiguous again.

For the ranges of indices in yellow area, the starting indices remain the same. To determine the ending index of a_{n-1}, since traversing all $a_2=i$ will automatically put those sets with $a_{n-1}=P+1-i$ into red area, which means the ceiling of a_{n-1} is at most $P+1-i-1=P-i=P-a_2$ because all sets with $a_{n-1}=P+1-i$, $a_{n-1}=P+1-(i-1)$ … $a_{n-1}=P+1-2$ are in red area. And ceiling of a_i, for $3\leq i\leq n-2$, is $a_{i+1}-1$. To determine the ending index of a_2, the sufficient and necessary condition of the rest $n-3$ loops being valid, which means $P-a_2+4-n \geq a_2+1$, then we have $a_2 \leq (P+3-n)/2$ and we take the floor value as its ending index.

For green area, first, there are $n-3$ loops because the sum of first and last elements is a fixed value, $P+1$. The starting indices are the same as previously stated, to determine the ending index of a_2, suppose the max value is k, then the max value of a_3 is $k+1$, max value of a_4 is $k+2$, …, the max value of a_{n-1} is $k+n-3$. Since $a_2+a_{n-1}=P+1$, we have $k+k+n-3 \leq P+1, k\leq (P+4-n)/2$. And ceiling of a_i, for $3\leq i\leq n-2$, is $a_{i-1}+1$.

□

4.34 Further Optimizations for Special P

To satisfy all-pairs property of an optimal cyclic quorum set, total job performed by every node must be no fewer than what is required. In a distributed system with P nodes, each node has a base set of size n. Suppose data set is equally divided into P subsets, then number of interaction required is $P(P+1)/2$. Total number of interactions could be performed by P nodes is $Pn(n-1)/2$. When these two numbers are the same, there is no redundant interactions performed and the whole system has the best performance. Taking advantage of no redundancy in these special cases where $P=n(n-1)+1$, we explore an optimization strategy that further compresses the search space.

Theorem 5. ***Optimization #4:*** Only search the space where the differences between every two contiguous loop indices are mutually different.

Proof. Suppose there is a base set $S'=\{a_0, a_1, ..., a_{n-1}\}$, in which $a_{j+1}-a_j = a_{i+1}-a_i = k$. We apply feasibility check on it: generate cyclic quorum sets and we get n base sets that have 0. We name them $S^{(0)}, S^{(1)},..., S^{(n-1)}$ respectively and their elements are denoted as $S^{(i)}=\{a_0^{(i)}, a_1^{(i)},..., a_{n-1}^{(i)}\}$ for $-1<i<n$. Base set S' is named as $S^{(0)}$, so $a_{i+1}^{(0)}-a_i^{(0)} = a_{j+1}^{(0)}-a_j^{(0)} = k$. Suppose in $S^{(x)}$, we have $a_i^{(0)}=0, a_{i+1}^{(0)}=k$ and in $S^{(y)}$, we have $a_j^{(0)}=0, a_{j+1}^{(0)}=k$. Among all elements in $S^{(i)}$, there are two of them with value k. Since no redundancy exists, the base set is impossible to satisfy all-pairs property. So, S' is not feasible. Hence, this theorem holds.

□

V. Function and Algorithm Analysis

V.1 Feasibility Check Function

Given a base set, we first compute n difference between each element and P. Then compute each element plus n differences modulus P. Put all these results into an array and check if all values from 0 to $P-1$ are present in the array. If all values are contained, return feasible. Otherwise, return not feasible. Here is the pseudocode for this function:

```
Function feasibility_check()
 Given a base set {$a_0, a_1, ... , a_{n-1}$} and P
    for $i \leftarrow 0$ to $n-1$ do
      difference[i] = $P-a_i$
        for $j \leftarrow 0$ to $n-1$ do
          put difference[i] + $a_j$ mod P into an array arr[n*n]
          end for
    end for

 for $i \leftarrow 0$ to $n*n-1$ do
   if(arr[i] contains all values from 0 to P)
       return "Feasible"
  else
       return "Not feasible"
   endif
  endfor
```

Filling *arr[]* costs $O(n^2)$ time. Checking values in arr[] costs $O(n^2)$ time. So, the overall time complexity of this function is $O(n^2)$.

V.2 Search Algorithm

If given a random value of P and base set size n, our goal is to find all feasible base sets that could generate optimal cyclic quorum sets with the given set size. For those measures introduced in Section 4.34, it cannot serve for this situation because we are discussing a generic case.

With the function *feasiblity_check()* and indices ranges determined in section 4.33, we state the algorithm below.

```
Algorithm CQS_searching
  Given P, n
  $a_0 \leftarrow 0$
  $a_1 \leftarrow 1$
   // yellow sets
   for $a_2 \leftarrow a_1+1$ to floor((P-n+3)/2) do
     for $a_3 \leftarrow a_2+1$ to $P-a_2+4-n$ do
        …
       for $a_{n-1} \leftarrow a_{n-2}+1$ to $P-a_2$ do
         feasibility_check()
       end for
        …
     end for
   end for
   // green sets
   for $a_2 \leftarrow a_1+1$ to floor((P-n+4)/2) do
     for $a_3 \leftarrow a_2+1$ to floor((P-n+4)/2)+1 do
        …
       for $a_{n-2} \leftarrow a_{n-3}+1$ to floor((P-n+4)/2)+n-4 do
          $a_{n-1}=P+1-a_2$
         feasibility_check()
       end for
        …
     end for
   end for
```

The time complexity of this algorithm is $O(n^k+n^{k-1})$, where k is the size of CQS. Because traversal of yellow and green sets each takes $O(n^{k-2})$ and $O(n^{k-3})$ time and feasibility check costs $O(n^k)$ time.

VI. Results

Due to pairing property we introduced in Section 4.33, we only need to search almost half the space and the other half of results can be generated correspondingly. Theoretically, *N* can be reduced by a factor of two. But due to the existence of self-paired sets, the number of searches will be slightly higher than 50%.

VI.1 Number of Computations with common P value

In the *CQS_searching* algorithms proposed in the previous section, the number of computations is equivalent to the number of iterations of the nested loops.

With *P* and *n* given, Optimization #1 removes first two loops. Optimization #2 narrows the index ranges, for example, in the *i*-th loop, the starting index, denoted by a_i, is $a_{i-1}+1$, and the ending index is $P–n+i+1$. While Optimization #3 takes the index decrease one step further, the total number of searches almost halved.

Here we provide *N* values with naïve search and after applying optimization #1, #2 and #3 for *P=4 to 6, 8 to 14.*

P	*n*	*N (Naïve)*	*N(#1)*	*N (#2)*	*N (#3)*
4	3	64	4	2	1
5	3	125	5	3	2
6	3	216	6	4	2
8	4	4096	64	15	9
9	4	6561	81	21	13
10	4	10000	100	28	16
11	4	14641	121	36	21
12	4	20736	144	45	25
14	5	537824	196	220	110

VI.2 Number of Computations with special P value

Here is part of results given special number of nodes. One thing to mention is that feasible base sets founded by applying naïve search have many equivalent results, but they appear as the outcome of the program with Naïve search algorithm. In the later phases, these repeated results will be eliminated. Therefore, figures drop in the last column.

P	*n*	*N (Naïve)*	*# of base sets*
7	3	243	84
13	4	28561	1248
21	5	4084101	5040
31	6	887503681	223200
57	8	111429157682001	27578880

If these special cases are dealt as normal ones, *N* values are shown below. Here is an example how it searches. Given *P=7, n=3*. There is *n–2=1* loop with starting index of 2 and ending index of *P–n+1+1=6*. So, five base sets are checked: *{0, 1, 2},{0, 1, 3},{0, 1, 4},{0, 1, 5},{0, 1, 6}*. And two of them are feasible, which are *{0, 1, 3}* and *{0, 1, 5}*.

P	*n*	*N (#1 #2)*	*# of base sets*
7	3	5	2
13	4	55	4
21	5	969	2
31	6	23751	10
57	8	28989675	12

After applying Optimization #3 and #4, *N* is further reduced. In the same case where *P=7, n=3.* After applying previous three optimizations, we know there are five base sets, but we do not check all of them. For example, *{0, 1, 2}* do not need to be checked. According to optimization #4, it is impossible to be a feasible one because its two differences are the same of this set. As a result, they only two sets to check are *{0, 1, 3}* and *{0, 1, 5}*. If Optimization #3 is also applied, *{0, 1, 5}* will be skipped because it is the pairing set of *{0, 1, 3}*.

P	*n*	*N (#4)*	*N (#3)*	*# of base sets(paired sets)*
7	3	2	1	2
13	4	18	9	4
21	5	216	126	2
31	6	3600	2292	10
57	8	2197440	1098720	12

VII. Conclusion

Currently there is no known measure to reduce the time complexity of exhaustive search algorithms to solve all-pairs problem. Therefore, we choose another aspect to optimize this process. According to our research, drastic number of unnecessary searches are avoided because it is not possible to have a feasible base set in those spaces.

The way to develop the searching optimization is special because first we used traditional naïve search algorithm to get all possible results. Then according to the observation and quantitative analysis of those results, we proposed several conjectures on optimization strategies accordingly. Some of them are proved to be wrong but others be very useful, such as the those shared in this paper. The algorithm we proposed in this still has unsatisfying performance speaking of time complexity. But the searching time was reduced to a very large extent.

For future work, we think there are at least two directions. To determine whether the application of different cyclic quorum sets have different performance. If so, what pattern does the cyclic quorum base set has the best performance; otherwise, we only need the first feasible cyclic quorum base set, which will save tons of time during the search. Or it is possible to replace the exhaustive search with randomized algorithm that may solve the searching problem in linear time with a very high probability. The other research direction is how to generate cyclic quorum base set with a given redundancy value[8], which benefits the design of a distributed system in fault-tolerant aspect.

References

[1] M. Driscoll, E. Georganas, P. Koanantakool, E. Solomonik, and K. Yelick, “A communication-optimal N-body algorithm for direct interactions,” in Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.,

[2] C. J. Kleinheksel and A. K. Somani, "Efficient Distributed All-Pairs Algorithms: Management Using Optimal Cyclic Quorums," in IEEE Transactions on Parallel and Distributed Systems, vol. 29, no. 2, pp. 391-404, 1 Feb. 2018.

[3] Wai-Shing Luk and Tien-Tsin Wong, "Two new quorum based algorithms for distributed mutual exclusion," Proceedings of 17th International Conference on Distributed Computing Systems, Baltimore, MD, USA, 1997, pp. 100-106.

[4] Mamoru Maekawa. A √N algorithm for mutual exclusion in decentralized systems. ACM Trans. Comput. Syst. 3, 2 (May 1985), 145–159. DOI:https://doi.org/10.1145/214438.214445

[5] T. Balharith and F. Alhaidari, "Round Robin Scheduling Algorithm in CPU and Cloud Computing: A review," 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 2019, pp. 1-7.

[6] R. Hedegaard, “Handshake problem,” Jan. 2016. [Online]. Available: http://mathworld.wolfram.com/HandshakeProblem.html

[7] C. J. Kleinheksel and A. K. Somani, "Resource efficient redundancy using quorum-based cycle routing in optical networks," 2015 17th International Conference on Transparent Optical Networks (ICTON), Budapest, 2015, pp. 1-4. 2013, pp. 1075–1084.

[8] C. J. Kleinheksel and A. K. Somani, "Unidirectional Quorum-Based Cycle Planning for Efficient Resource Utilization and Fault-Tolerance," 2016 25th International Conference on Computer Communication and Networks (ICCCN), Waikoloa, HI, 2016, pp. 1-8.

Appendix

Table 1: *N* with different algorithms

P	*n*	Naïve	OPT1	Result	Percentage(%) of OPT1
8	4	4096	12	9	75.00
9	4	6561	15	13	86.67
10	4	10000	22	16	72.73
11	4	14641	36	21	58.33
12	4	20736	45	25	55.56
13	4	28561	18	9	50.00
14	5	537824	220	110	50.00
15	5	759375	286	146	51.05
16	5	1.05E+06	364	182	50.00
17	5	1.42E+06	455	231	50.77
18	5	1.89E+06	560	280	50.00
19	5	2.48E+06	680	344	50.59
21	5	4.08E+06	216	126	58.33
31	6	8.88E+08	3600	2292	63.67
57	8	1.11E+14	2197440	1.1E+06	50.06

Table 2: All feasible base for *P=7, 13, 21, 31, 57*

P=7, n=3	
{0, 1, 3}	{0, 1, 5}
P=13, n=4	
{0, 1, 3, 9}	{0, 1, 5, 11}
{0, 1, 4, 6}	{0, 1, 8, 10}
P=21, n=5	
{0, 1, 4, 14, 16}	{0, 1, 6, 8, 18}
P=31, n=6	
{0, 1, 3, 8, 12, 18}	{0, 1, 14, 20, 24, 29}
{0, 1, 3, 10, 14, 26}	{0, 1, 6, 18, 22, 29}
{0, 1, 4, 6, 13, 21}	{0, 1, 11, 19, 26, 28}
{0, 1, 4, 10, 12, 17}	{0, 1, 15, 20, 22, 28}
{0, 1, 8, 11, 13, 17}	{0, 1, 15, 19, 21, 24}
P=57, n=8	
{0, 1, 3, 13, 32, 36, 43, 52}	{0, 1, 6, 15, 22, 26, 45, 55}
{0, 1, 4, 9, 20, 22, 34, 51}	{0, 1, 7, 24, 36, 38, 49, 54}
{0, 1, 4, 12, 14, 30, 37, 52}	{0, 1, 6, 21, 28, 44, 46, 54}
{0, 1, 5, 7, 17, 35, 38, 49}	{0, 1, 9, 20, 23, 41, 51, 53}
{0, 1, 5, 27, 34, 37, 43, 45}	{0, 1, 13, 15, 21, 24, 31, 53}
{0, 1, 7, 19, 23, 44, 47, 49}	{0, 1, 9, 11, 14, 35, 39, 51}

Table 3: All feasible base sets for *P=4 to 6, 8 to 12, 14*

P=4, n=3	
{0, 1, 2}	{0, 1, 3}
P=5, n=3	
{0, 1, 2}	{0, 1, 4}
{0, 1, 3}	self
P=6, n=3	
{0, 1, 3}	{0, 1, 4}
P=8, n=4	
{0, 1, 2, 4}	{0, 1, 5, 7}
{0, 1, 2, 5}	{0, 1, 4, 7}
{0, 1, 2, 6}	{0, 1, 3, 7}
{0, 1, 3, 4}	{0, 1, 5, 6}
{0, 1, 3, 5}	{0, 1, 4, 6}
P=9, n=4	
{0, 1, 2, 4}	{0, 1, 6, 8}
{0, 1, 2, 5}	{0, 1, 5, 8}
{0, 1, 2, 6}	{0, 1, 4, 8}
{0, 1, 2, 7}	{0, 1, 3, 8}
{0, 1, 3, 4}	{0, 1, 6, 7}
{0, 1, 3, 5}	{0, 1, 5, 7}
{0, 1, 3, 6}	{0, 1, 4, 7}
{0, 1, 3, 7}	self
{0, 1, 4, 6}	self
P=10, n=4	
{0, 1, 2, 5}	{0, 1, 6, 9}
{0, 1, 2, 7}	{0, 1, 4, 9}
{0, 1, 3, 5}	{0, 1, 6, 8}
{0, 1, 3, 6}	{0, 1, 5, 8}
{0, 1, 4, 6}	{0, 1, 5, 7}
P=11, n=4	
{0, 1, 2, 5}	{0, 1, 7, 10}
{0, 1, 2, 8}	{0, 1, 4, 10}
{0, 1, 3, 5}	{0, 1, 7, 9}
{0, 1, 3, 7}	{0, 1, 5, 9}
{0, 1, 3, 8}	{0, 1, 4, 9}
{0, 1, 4, 6}	{0, 1, 6, 8}
P=12, n=4	
{0, 1, 3, 7}	{0, 1, 6, 10}
{0, 1, 4, 6}	{0, 1, 7, 9}
P=14, n=5	
{0, 1, 2, 3, 7}	{0, 1, 8, 12, 13}
{0, 1, 2, 4, 7}	{0, 1, 8, 11, 13}
{0, 1, 2, 4, 9}	{0, 1, 6, 11, 13}
{0, 1, 2, 5, 7}	{0, 1, 8, 10, 13}
{0, 1, 2, 5, 8}	{0, 1, 7, 10, 12}

{0, 1, 2, 5, 9}	{0, 1, 6, 10, 13}
{0, 1, 2, 6, 9}	{0, 1, 6, 9, 13}
{0, 1, 2, 8, 11}	{0, 1, 4, 7, 13}
{0, 1, 3, 4, 8}	{0, 1, 7, 11, 12}
{0, 1, 3, 5, 7}	{0, 1, 8, 10, 12}
{0, 1, 3, 6, 7}	{0, 1, 8, 9, 12}
{0, 1, 3, 6, 10}	{0, 1, 5, 9, 12}
{0, 1, 3, 7, 8}	{0, 1, 7, 8, 12}
{0, 1, 3, 7, 9}	{0, 1, 6, 8, 12}
{0, 1, 3, 7, 10}	{0, 1, 5, 8, 12}
{0, 1, 3, 7, 12}	{0, 1, 3, 8, 12}
{0, 1, 3, 8, 10}	{0, 1, 5, 7, 12}
{0, 1, 3, 9, 10}	{0, 1, 5, 6, 12}
{0, 1, 3, 10, 11}	{0, 1, 4, 5, 12}
{0, 1, 4, 6, 8}	{0, 1, 7, 9, 11}
{0, 1, 4, 6, 11}	{0, 1, 4, 9, 11}
{0, 1, 4, 7, 9}	{0, 1, 6, 8, 11}
{0, 1, 4, 8, 10}	{0, 1, 5, 7, 11}
{0, 1, 5, 7, 10}	{0, 1, 5, 8, 10}

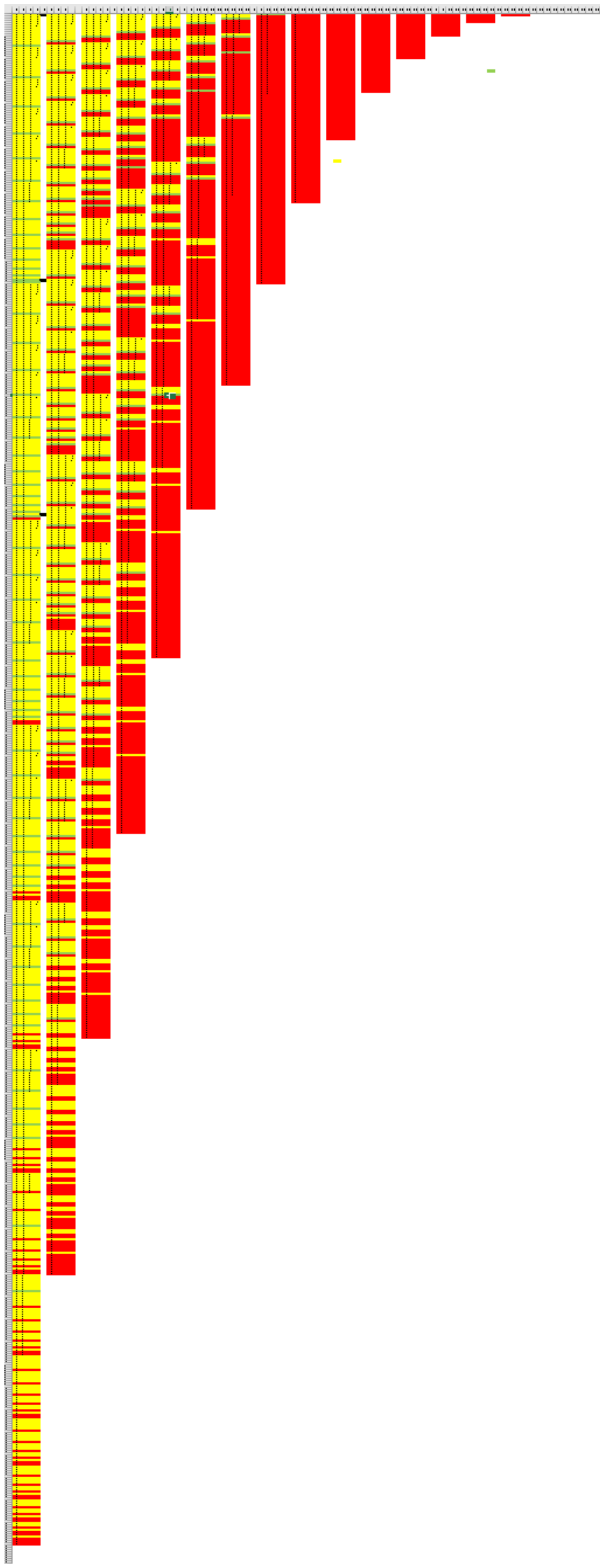


Figure 11. Overview of Search Space Division *P=20,n=6*

2	3	4	🟨
2	3	5	🟨
2	3	6	🟨
2	3	7	🟨
2	3	8	🟨
2	3	9	🟨
2	3	10	🟨
2	3	11	🟨
2	3	12	🟨
2	3	13	🟩
2	4	5	🟨
2	4	6	🟨
2	4	7	🟨
2	4	8	🟨
2	4	9	🟨
2	4	10	🟨
2	4	11	🟨
2	4	12	🟨
2	4	13	🟩
2	5	6	🟨
2	5	7	🟨
2	5	8	🟨
2	5	9	🟨
2	5	10	🟨
2	5	11	🟨
2	5	12	🟨
2	5	13	🟩
2	6	7	🟨
2	6	8	🟨
2	6	9	🟨
2	6	10	🟨
2	6	11	🟨
2	6	12	🟨
2	6	13	🟩
2	7	8	🟨
2	7	9	🟨
2	7	10	🟨
2	7	11	🟨
2	7	12	🟨
2	7	13	🟩
2	8	9	🟨
2	8	10	🟨
2	8	11	🟨
2	8	12	🟨
2	8	13	🟥
2	9	10	🟨
2	9	11	🟨
2	9	12	🟨
2	9	13	🟥
2	10	11	🟨
2	10	12	🟨
2	10	13	🟥
2	11	12	🟨
2	11	13	🟥
2	12	13	🟥

3	4	5	🟨
3	4	6	🟨
3	4	7	🟨
3	4	8	🟨
3	4	9	🟨
3	4	10	🟨
3	4	11	🟨
3	4	12	🟩
3	4	13	🟥
3	5	6	🟨
3	5	7	🟨
3	5	8	🟨
3	5	9	🟨
3	5	10	🟨
3	5	11	🟨
3	5	12	🟩
3	5	13	🟥
3	6	7	🟨
3	6	8	🟨
3	6	9	🟨
3	6	10	🟨
3	6	11	🟨
3	6	12	🟩
3	6	13	🟥
3	7	8	🟨
3	7	9	🟨
3	7	10	🟨
3	7	11	🟨
3	7	12	🟩
3	7	13	🟥
3	8	9	🟨
3	8	10	🟨
3	8	11	🟨
3	8	12	🟥
3	8	13	🟥
3	9	10	🟨
3	9	11	🟨
3	9	12	🟥
3	9	13	🟥
3	10	11	🟨
3	10	12	🟥
3	10	13	🟥
3	11	12	🟥
3	11	13	🟥
3	12	13	🟥

4	5	6	🟨
4	5	7	🟨
4	5	8	🟨
4	5	9	🟨
4	5	10	🟨
4	5	11	🟩
4	5	12	🟥
4	5	13	🟥
4	6	7	🟨
4	6	8	🟨
4	6	9	🟨
4	6	10	🟨
4	6	11	🟩
4	6	12	🟥
4	6	13	🟥
4	7	8	🟨
4	7	9	🟨
4	7	10	🟨
4	7	11	🟩
4	7	12	🟥
4	7	13	🟥
4	8	9	🟨
4	8	10	🟨
4	8	11	🟥
4	8	12	🟥
4	8	13	🟥
4	9	10	🟨
4	9	11	🟥
4	9	12	🟥
4	9	13	🟥
4	10	11	🟥
4	10	12	🟥
4	10	13	🟥
4	11	12	🟥
4	11	13	🟥
4	12	13	🟥

5	6	7	🟨
5	6	8	🟨
5	6	9	🟨
5	6	10	🟩
5	6	11	🟥
5	6	12	🟥
5	6	13	🟥
5	7	8	🟨
5	7	9	🟨
5	7	10	🟩
5	7	11	🟥
5	7	12	🟥
5	7	13	🟥
5	8	9	🟨
5	8	10	🟥
5	8	11	🟥
5	8	12	🟥
5	8	13	🟥
5	9	10	🟥
5	9	11	🟥
5	9	12	🟥
5	9	13	🟥
5	10	11	🟥
5	10	12	🟥
5	10	13	🟥
5	11	12	🟥
5	11	13	🟥
5	12	13	🟥

6	7	8	🟨
6	7	9	🟩
6	7	10	🟥
6	7	11	🟥
6	7	12	🟥
6	7	13	🟥
6	8	9	🟥
6	8	10	🟥
6	8	11	🟥
6	8	12	🟥
6	8	13	🟥
6	9	10	🟥
6	9	11	🟥
6	9	12	🟥
6	9	13	🟥
6	10	11	🟥
6	10	12	🟥
6	10	13	🟥
6	11	12	🟥
6	11	13	🟥
6	12	13	🟥

7	8	9	🟥
7	8	10	🟥
7	8	11	🟥
7	8	12	🟥
7	8	13	🟥
7	9	10	🟥
7	9	11	🟥
7	9	12	🟥
7	9	13	🟥
7	10	11	🟥
7	10	12	🟥
7	10	13	🟥
7	11	12	🟥
7	11	13	🟥
7	12	13	🟥

8	9	10	🟥
8	9	11	🟥
8	9	12	🟥
8	9	13	🟥
8	10	11	🟥
8	10	12	🟥
8	10	13	🟥
8	11	12	🟥
8	11	13	🟥
8	12	13	🟥

9	10	11	🟥
9	10	12	🟥
9	10	13	🟥
9	11	12	🟥
9	11	13	🟥
9	12	13	🟥

10	11	12	🟥
10	11	13	🟥
10	12	13	🟥

11	12	13	🟥