A Polynomial-time Solution for the Maximum Subsets Problem

Khalil Challita, *Member, IAENG,* Jacques Bou Abdo, Hikmat Farhat, Mireille Makary

***Abstract*—The main purpose of this paper is to determine a non-trivial tractable class of the maximum (k,m)-subsets problem. More specifically, we show that we can solve this problem in polynomial time for $m = 1$. Depending on the value of k with respect to n, we prove that in the best-case scenario our algorithm runs in $O(\sqrt{n})$ time. On the other hand, an upper bound for solving it is given by $O(n^4/k^3)$. Furthermore, for the case where $n = k^2$, we design and implement an algorithm in Python that yields a solution in $O(n^{5/2})$.**

***Index Terms*—Algorithms, Discrete mathematics, tractability.**

I. Introduction

In this paper, we define a new problem that stems from set theory. This problem can be formulated as follows: given a set S of n elements, find the largest set F that includes subsets of S of size k each, and such that the intersection of any two subsets of F contains at most m elements. Since this is a maximization problem that involves three variables, we decided to refer to it as the maximum (k,m)-subsets problem. Several similar problems were proven to be NP-hard, such as the subset sum problem studied by Karp [15], the maximum subarray problem [18], and the k maximum sums problem [3].
An important application of such a problem is load balancing several tasks over a specific number of resources. Given for example a cluster of n computers, assume that each task is allocated one sub-cluster of size k. To maximize the efficiency of our network, we decide to enforce the following constraint: any two sub-clusters are not allowed to share more than m computers.
Since this problem has been showed to be NP-hard [4], determining the maximum number of simultaneous tasks a cluster of n computers can handle may require exponential time. It becomes practically infeasible to answer this question in case the network is dynamic; in a sense where the number of resources can change over time.
Determining the complexity of a problem has many applications in different areas of mathematics, biology, physics, and computer science [6], [7], [14], [5], [8], [12], [13], [19]. The main contribution of this paper is to show that the maximum (k,m)-subsets problem becomes tractable for $m = 1$, irrespective of the values of n and k. In other words, we show in this particular case that we can determine the maximum number of simultaneous tasks in polynomial time. We started by considering the case for which $n = k^2$ (i.e. n is a perfect square). Afterwards, we extended our proof to any value of n.
The remainder of this paper is divided as follows. We formally define the problem in Section II. Section III deals with the case where $n = k^2$. We provide here a pseudo-code that constructs the set F in $T(n) = O(n^{5/2})$. We show in Section IV that the cardinal of F is bounded by $O((n/k)^2)$ when n is a perfect power of k. We use this result to prove in Section V that constructing F requires up to $O(n^4/k^3)$ in the general case. We also exhibit a method that constructs F in $\Theta(k)$ when k is even and $n = k^2$.

Manuscript submitted October 25, 2021; revised March 11, 2022.

Khalil Challita is a Senior Teaching Fellow in the Department of Computer Science, Warwick University, Coventry, United Kingdom, (e-mail: khalil.challita@warwick.ac.uk).

Jacques Bou Abdo is an Assistant Professor in the College of Business and Technology, University of Nebraska at Kearney, USA, (e-mail: bouabdoj@unk.edu).

Hikmat Farhat is a Senior Teaching Fellow in the School of Electronics and Computer Science, University of Southampton, United Kingdom, (e-mail: h.farhat@soton.ac.uk).

Mireille Makary is an Assistant Professor in the Department of Computer Science, Holy Spirit University, Kaslik, Lebanon, (e-mail: mireille.makary@usek.edu.lb).

II. Preliminaries

In this section, we simply provide a formal definition of our problem.

Definition 1: **Maximum (k,m)-subsets problem**
Let n, k, m, where $U_n = \{a_1, \ldots, a_n\}$, and $0 \leq m \leq k < n$. We denote by $U_{k,n} = \{L \subseteq U_n : Card(L) = k\}$ the set of subsets of U_n that have exactly k elements.
We say that $F^m_{k,n} \in 2^{U_{k,n}}$ is a maximum (k,m)-subset of U_n if it satisfies the following conditions:

1) $\forall L, L' \in F^m_{k,n}$, $Card(L \cap L') \leq m$.
2) $\forall F' \in 2^{U_{k,n}}$ that satisfies the above condition, we have $Card(F') \leq Card(F^m_{k,n})$.

It is easy to see that solving a general instance of our problem using a brute-force approach has a double exponential leading factor.
Indeed, enumerating all the subsets of U_n of size k requires exponential time. Then finding the largest subset of $U_{k,n}$ that satisfies the conditions of Definition 1 involves processing subsets of a set that contains an exponential number of elements. Therefore, the overall cost of this method has a double-exponential upper bound.
For the remainder of this paper, we shall refer to $F^m_{k,n}$ as the solution to our problem.

Proposition 1: A brute-force algorithm for solving the maximum (k,m)-subsets problem has a leading factor equal to $2^{C^k_n}$.

Proof A straightforward method is to enumerate all the possible subsets of U_n that has size k. We have C^k_n such subsets. Then we process all the subsets of this set that satisfy the conditions of Definition 1, keeping track of the largest one. Note that we have $2^{C^k_n}$ such subsets.
For $k = \Theta(n)$, we have $2^{C^k_n} = O(2^{2^{n-1}})$, which is a double-exponential.

At first glance, this problem seems to be at least as difficult to solve as the set cover problem, where we have to find

the smallest sub-collection of a set $S \subseteq 2^{U_n}$ whose union covers U_n. But actually the maximum (k,m)-subsets problem is EXP-hard. Indeed, solving any instance of this problem where $k = \Theta(n)$ and $m = k-1$ requires at least exponential time. This is because $F_{k,n}^{k-1}$ contains all the subsets of $U_{k,n}$, and it takes an exponential time to enumerate the solution that consists of C_n^k subsets.

We suspect this problem to be NEXP-complete. For that one shall reduce a known NEXP-complete problem (e.g. the succinct K-coloring problem [17], [11]) to the maximum (k,m)-subsets problem, possibly using binary decision diagrams([16], [1]) to capture the intersection of subsets of $U_{k,n}$.

III. Case where $n = k^2$

In this section we turn our attention to solving the maximum (k,m)-subsets problem where n is a perfect square.

By determining a lower bound to compute C_n^k, we immediately notice that using a brute-force approach for solving this particular instance of our problem is impractical.

Proposition 2: For $n = k^2$, we get the following lower bound to compute C_n^k:

$$C_{k^2}^k = \Omega(k^k) = \Omega((log\, n)^{log\, n})$$

Proof Let $n = k^2$.

For all i where $1 \leq i \leq k-1$, it is easy to check that $\frac{n}{k} \leq \frac{n-i}{k-i}$.

Thus we have:

$$\begin{aligned} \frac{n}{k} \times \cdots \times \frac{n}{k} &\leq \frac{n}{k} \times \frac{n^k}{k^k} \\ \frac{n}{k} \times \frac{n^k}{k^k} &\leq C_n^k \\ \frac{k^{2k}}{k^k} &\leq C_{k^2}^k \\ k^k &\leq C_{k^2}^k \end{aligned}$$

This concludes our proof since $k^k = (log\, n)^{log\, n}$.

Corollary 1: Recalling Proposition 1, solving our problem requires

$$T(n) = \Omega(2^{(log\, n)^{log\, n}})$$

The key idea for solving any instance of the (k,m)-subsets problem (when $m = 1$) in polynomial time is to represent the elements of U_n using a $k \times k$ square matrix.

Denote by A the below matrix where $A[i,j] = a_{ij}$.

$$\begin{matrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{matrix}$$

We start by providing a factorial upper bound for the number of elements in $Card(F_{k,k^2}^1)$.

Proposition 3: Given an integer $k \geq 2$, we certify that $Card(F_{k,k^2}^1) = O(\sqrt{n}\,!)$

Proof We start by selecting all the elements from each row of A to form subsets of size k. Then we do the same for the columns of A, for a total of $2k$ subsets.

Afterwards, we select the remaining subsets as follows: choose one element a_{1j_1} from the first row, then a second element a_{2j_2} from the second row, and so on, until we select element a_{kj_k} from the last row; making sure that for all $l \neq l'$ we have $j_l \neq j_{l'}$. In other words, no two elements of a subset belong to the same column.

Obviously, we have k choices for the first element, $k-1$ for the second, and so on; for a total of $k!$ possible subsets. Therefore, an upper bound for $Card(F_{k,k^2}^1)$ is $2k + k!$, with $k = \sqrt{n}$.

We next show that F_{k,k^2}^1 contains a linear number of subsets.

Proposition 4: Given an integer $k \geq 2$, we certify that $Card(F_{k,k^2}^1) = O(k^2) = O(n)$

Proof As we did in Proposition 3, we start by processing the k rows of A to build k distinct subsets. Then, for each of the k elements a_{1j} (where $1 \leq j \leq k$) of the first row we repeat the following:

Select subsequent elements from the following rows with a shift of 0, then with a shift of 1, and so on until we select an element from the last row with a shift of $k-1$. Note that the shifts are computed modulus k. In other words, for each a_{1j} we construct the k subsets:

$$\begin{matrix} \{a_{1j}, & a_{2j}, & a_{3j}, & \cdots\} \\ \{a_{1j}, & a_{2[(j+1) mod\, k]}, & a_{3[(j+2) mod\, k]}, & \cdots\} \\ \{a_{1j}, & a_{2[(j+2) mod\, k]}, & a_{3[(j+4) mod\, k]}, & \cdots\} \\ & \vdots & & \\ \{a_{1j}, & a_{2[(j+k-1) mod\, k]}, & a_{3[(j+2(k-1)) mod\, k]}, & \cdots\} \end{matrix}$$

For a total of $k^2 + k = O(n)$ generated subsets.

Obviously, our method generates all the potential subsets of size k that form F_{k,k^2}^1. For suppose we can add a subset T of size k that is different from all the constructed ones so far, then its elements must be selected from different rows and columns across the rows of the matrix. But since there are at most $k-1$ possible distinct shifts to form a subset of k elements (note that shifts 0 and k are identical), and that our method already considered all the shifts from 0 to $k-1$, the intersection of T with some previously constructed subset must contain at least two elements (e.g. a_{ij} and $a_{i(j+shift)}$, for some i, j and $shift$).

Therefore $Card(F_{k,k^2}^1) = O(k^2 + k) = O(n)$.

Remark 1: Each time we construct a subset, we check to see if its intersection with all the previously constructed ones contains at most one element. If this is the case we add it to F_{k,k^2}^1, otherwise we discard it.

We illustrate our method in the next two examples, before proposing an algorithm that solves this class of problems in polynomial time.

Without loss of generality, we assume that the elements of the set U_n are numbered from 1 to n.

Example 1: Let $k = 3$ and $n = 9$. The elements of U_9 are:

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

We build $F_{3,3^2}^1$ as follows: we select the 3-subsets from each row first. Then we select one element from each row with a shift of i, where $0 \leq i \leq 3$. Whenever appropriate (see Remark 1), we add the processed subset to $F_{3,3^2}^1$.

$F_{3,3^2}^1 = \{\{1,2,3\}, \{4,5,6\}, \{7,8,9\},$

$\{1,4,7\},\{1,5,9\},\{1,6,8\},$
$\{2,5,8\},\{2,6,7\},\{2,4,9\},$
$\{3,6,9\},\{3,4,8\},\{3,5,7\}\}$.

In this case, the cardinal of $F^1_{3,3^2}$ is equal to $k+k^2 = 3+9 = 12$.

Example 2: Let $k = 4$ and $n = 16$. The elements of U_{16} are:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

In this case we get:
$F^1_{4,4^2} = \{\{1,2,3,4\},\{5,6,7,8\},\{9,10,11,12\},$
$\{13,14,15,16\},\{1,5,9,13\},\{1,6,11,16\},$
$\{2,6,10,14\},\{2,7,12,13\},\{3,8,9,14\},$
$\{3,7,11,15\},\{4,8,12,16\},\{4,5,10,15\}\}$.

Not all the generated subsets are part of the solution. For example, $\{1,7,9,15\}$ and $\{1,8,11,14\}$) are not included since their intersection with previously constructed subsets contains two elements.
The cardinal of $F^1_{4,4^2}$ is equal to 12.

Algorithm 1 generates $k^2 + k$ subsets of size k each and returns F^1_{k,k^2}, which is a solution to our problem.

Proposition 5: For $n = k^2$, the maximum (k,m)-subsets problem can be solved in:

$$T(n) = O(k^5) = O(n^{5/2})$$

Proof This result follows from Algorithm 1. It is easy to see that the running time of the nested for loops at lines 3 to 7 is $\Theta(k^2)$. We add to it the cost of the for loops between lines 10 and 30. The outer two for loops have cost $\Theta(k^2)$, multiplied by the sum of the two inner for loops: the one at lines 14 to 17 has cost $\Theta(k)$, and the one at lines 19 to 28 requires $O(k^3)$. Indeed, we know from Proposition 4 that there are at most $O(k^2)$ subsets in F^1_{k,k^2}. Since all the elements in the subsets are sorted (by construction), we can compare two subsets in $O(k)$ (instead of the usual $O(k^2)$ time); for a total number of $O(k^3)$ operations. Thus the overall running time of the algorithm becomes:
$T(n) = O(k^2 + k^2 \times (k + k^3)) = O(k^5) = O(n^{5/2})$.

IV. Case where $n = k^p$, $p > 2$

We consider here instances of our problem where $n = k^p$ (for some $p > 2$), which can be viewed as a generalization of the result established in the previous section.
Unless stated otherwise, we represent the elements of U_n in blocks (i.e. matrices) of size $k \times k$, as shown in Figure 1.

Algorithm 1 Maximum (k,m)-subsets problems where $n = k^2$

Require: $n, k \in N$, with $n = k^2$ and $m = 1$
Ensure: Returns the largest set F^1_{k,k^2}
1: Initialize the $k \times k$ square matrix A with values 0 to n-1
2: $F^1_{k,n} = \emptyset$
3: **for** $i = 0$ to $k-1$ **do**
4: $S = \emptyset$
5: **for** $j = 0$ to $k-1$ **do**
6: $S = S \cup \{A[i,j]\}$ {Reading off the elements row by row}
7: **end for**
8: $F^1_{k,n} = F^1_{k,n} \cup S$
9: **end for**
10: **for** $i = 0$ to $k-1$ **do**
11: **for** $j = 0$ to $k-1$ **do**
12: shift = j {Specifies the shift from 0 to k-1}
13: $S = \{A[0,i]\}$
14: **for** $l = 1$ to $k-1$ **do**
15: $S = S \cup \{A[(l),(i+shift) mod\, k]\}$
16: $shift = shift + j$ {Ensures that a constant spacing is preserved across the rows of A}
17: **end for**
18: counter = 0 {Test to see if we checked all the intersections with all the subsets of F}
19: **for all** $S' \in F^1_{k,n}$ **do**
20: **if** $Card(S' \cap S) <= 1$ **then**
21: counter = counter + 1
22: **else**
23: break
24: **end if**
25: **if** counter == $card(F^1_{k,n})$ **then**
26: $F^1_{k,n} = F^1_{k,n} \cup S$
27: **end if**
28: **end for**
29: **end for**
30: **end for**
31: **return** F^1_{k,k^2}

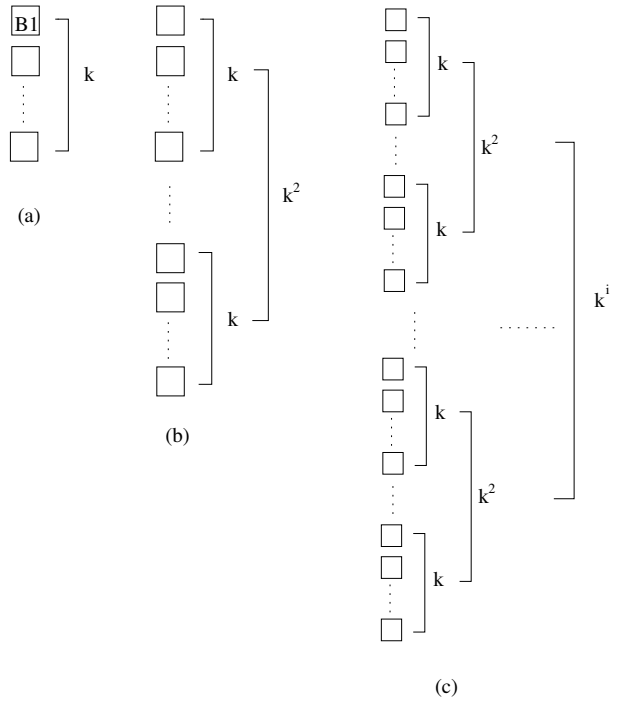

Fig. 1. Blocks of size $k \times k$ representing U_{k^3}, U_{k^4}, and $U_{k^{i+2}}$, respectively.

Lemmas 1 and 2 will be used in Proposition 6 to prove that the number of subsets of F^1_{k,k^p} is sub-quadratic.

Lemma 1: Given k blocks of k^2 elements each, there are at most $k^2 \times k^2$ subsets that belong to F^1_{k,k^p} and that satisfy the following condition: for any subset, no two elements belong to the same $k \times k$ block.

Proof Referring to Figure 1 (a), it is easy to see that we can construct at most k^2 subsets of size k that contain the same element from the first block and that satisfy the conditions of Definition 1. For example, if we fix the first element of Block 1 (denoted by B1 in Figure 1), we construct the first subset by selecting the remaining elements from each of the following $k-1$ blocks. Once done, we have k^2-1 choices left to construct the second subset. Indeed, we cannot choose any of the previously selected elements again without violating the first condition of Definition 1. In general, after step i, we have at most k^2-i choices left to construct subset $i+1$. Repeating this process, we can construct up to k^2 subsets that contain the first element of Block 1.
Applying the same reasoning to all of the k^2 elements of the first block, we conclude that we can construct at most $k^2 \times k^2 = k^4$ subsets from the given k blocks.

Lemma 2: Given k^i blocks of k^2 elements each, divided into k subgroups of k^{i-1} blocks, there are at most $k^{i+1} \times k^{i+1}$ subsets that belong to F^1_{k,k^p} and that satisfy the following condition: for any subset, no two elements belong to the same subgroup.

Proof This result is a generalization of the previous lemma. Figure 1 (b) represents the case for $i=2$. Applying the same reasoning as before, we can check that by fixing one element from the first subgroup we can construct at most $k^2 \times k = k^3$ subsets of size k that satisfy the condition of the lemma. Since we have k^3 different elements in the first subgroup, one can build up to $k^3 \times k^3$ subsets. Generally speaking (refer to Figure 1 (c)), given k^i blocks divided into k subgroups containing k^{i-1} blocks each, by fixing an element in the first subgroup we check that we can construct at most $k^2 \times k^{i-1} = k^{i+1}$ subsets of size k.
We conclude by noticing that there are k^{i+1} different elements in the first subgroup.

Proposition 6: For $n = k^p$, $card(F^1_{k,n}) = O((n/k)^2)$

Proof Given the set U_{k^p}, we represent its elements with blocks of size $k \times k$ as shown in Figure 2. Then we (iteratively) divide them into subgroups of size $k, k^2, \cdots, k^{p-2}$. All the subgroups at a specific level contain the same number of blocks. One can easily check that there are $k^p/k^2 = k^{p-2}$ subgroups at level 0, k^{p-3} subgroups at level 1, and so on, till we get $k^0 = 1$ subgroup at level $p-2$.

In general, there are $k^{p-2-i} = k^{p-1-i}$ subgroups at level i. So $card(F^1_{k,n})$ is bounded by the below summation S_n, where the left terms indicate the number of different subsets of size k we can construct from one group at a given level (refer to Lemma 2), and the right terms represent the total number of subgroups we have at each level .

$$\begin{aligned} S_n &= [(k \times k) \times k^{p-2} \\ &+ (k^2 \times k^2) \times k^{p-3} \\ &+ \vdots \\ &+ (k^i \times k^i) \times k^{p-1-i} \\ &+ (k^{(i+1)} \times k^{(i+1)}) \times k^{p-2-i} \\ &+ \vdots \\ &+ (k^{p-1} \times k^{p-1}) \times k^0] \end{aligned}$$

So we have:

$$\begin{aligned} S_n &= \sum_{i=0}^{p-2}((k^2)^{i+1} \times k^{p-2-i}) \\ &= \sum_{i=0}^{p-2} k^{i+p} \\ &= k^p \times \sum_{i=0}^{p-2} k^i \\ &= k^p \times \frac{k^{p-1}-1}{k-1} \\ &= (k^{p-1})^2 \end{aligned}$$

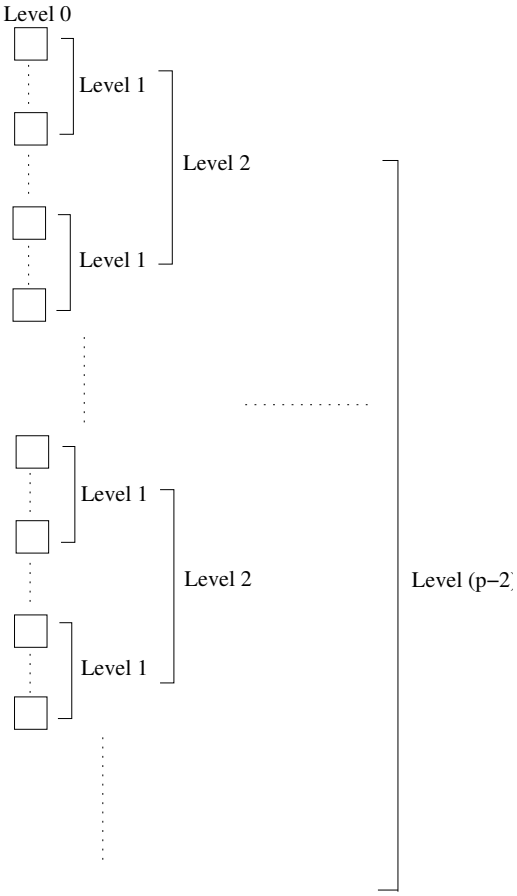

Fig. 2. Subgroups of size $k, k^2, \ldots, k^{p-2}$ representing U_{k^p}.

V. General case

We show in this section how to compute $F^1_{k,n}$ in polynomial time. We distinguish two cases.

A. Case where $n \leq k^2$

If $n = k^2$, then $Card(F^1_{k,n}) = O(n)$ as we showed in Proposition 4. Otherwise we have the following result:

Proposition 7: Let $n = q \times k + r$. If $n < k^2$ then we have:

$$Card(F^1_{k,n}) = \begin{cases} q & \text{if } r = 0 \\ O(n) & \text{otherwise} \end{cases}$$

Proof We reresent the elements of U_n in rows of k elements:

$$\begin{array}{ccccc} a_{11} & a_{12} & \cdots & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2k} \\ & \vdots & & & \\ a_{(q+1)1} & a_{(q+2)1} & \cdots & a_{(q+1)r} & \end{array}$$

If n is an exact multiple of k, then $Card(F^1_{k,qk}) = q$. We select the q rows of the above table to form the subsets of $F^1_{k,qk}$ (note that in this case the last row is empty). No other subset can be included since it will contain at least two elements from a specific row.
Otherwise, the worst case scenario occurs when $n = k^2 - 1$ (i.e. the last row is missing only one element). Following the same reasoning as we did in Proposition 4, we know that we can build up to $k - 1$ subsets once we fix an element in the first row, for a total of $O(k \times (k-1)) = O(k^2)$ subsets.

B. Case where $n > k^2$

We extend in this subsection the result established in Proposition 6.

Definition 2: Let i be a positive integer. We say that a subgroup of k^i elements (represented in blocks of size $k \times k$) has been processed if the following holds: constructing any new subset of size k by choosing two elements that belong to the subgroup will violate the first condition of Definition 1.

Referring to Figure 2 for example, if all the subgroups at some level j are processed, then we have to select k elements from different subgroups to form a new subset at level $j + 1$.

Proposition 8: For $n > k^2$, $Card(F^1_{k,n}) = O(n^2/k)$.

Proof This result follows directly from Proposition 6. Indeed, there exists some integer $p > 2$ such that $k^p \leq n \leq k^{p+1}$. So in the worst-case scenario, the number of subsets we can construct is bounded by the expression $S_n = \sum_{i=0}^{p-1}((k^2)^{i+1} \times k^{p-2-i})$, which includes one more term than the summation given in Proposition 6. Therefore :

$$\begin{aligned} S_n &= k^p \times \sum_{i=0}^{p-1} k^i \\ &= k^p \times \frac{k^p - 1}{k - 1} \\ &= (k^{2p-1}) \end{aligned}$$

Proposition 9: An upper bound for solving the maximum (k,m)-subsets problem is given by $T(n) = O(n^4)$.

Proof We consider two cases here.
Let $n \leq k^2$. We know from Proposition 5 that we can solve our problem in $T(n) = O(n^{5/2})$ in case $n = k^2$. The same complexity bound can be achieved when $n < k^2$. Indeed, referring to Proposition 7, we know that at most $O(n)$ subsets belong to $F^1_{k,n}$. Comparing a given subset with all the previously constructed ones requires $O(kn)$, as already explained in the proof of Proposition 5, for a total running time bounded by $O(kn^2)$.
Consider the case where $n > k^2$. Checking whether or not we can add a subset of size k to $F^1_{k,n}$ requires at most $O(k \times (n^2/k))$, where k is the cost of comparing two ordered subsets of size k, and $O(n^2/k)$ is the maximum size of $F^1_{k,n}$ (see Proposition 8). Therefore, the overall running time required to solve our problem is bounded by $O(n^2 \times n^2) = O(n^4)$.

Proposition 10: The maximum (k,m)-subsets problem can be solved in $O(n^4/k^3)$.

We can improve the upper bound of Proposition 9 by taking advantage of the repetitive structure of the problem as shown in Figure 2. Indeed, we know from Lemma 2 that there are k^{p-2-i} subgroups at level i. In order to process (recall Definition 2) a subgroup at level i, we only need to process one of its blocks B_i, then map the selected subsets (that are part of the solution) to the remaining blocks of this subgroup.
Processing a subgroup at level i requires $(k^i \times k^i)^2 \times k$ steps: one can generate at most $k^i \times k^i$ subgroups, and each subgroup must be checked against the $k^i \times k^i$ generated ones; multiplied by the cost of comparing two ordered subsets of size k. On the other hand, we need $O(k^{p-2-i}) \times (k^{(i+1)} \times k^{(i+1)})$ steps for the mapping: which is equal to the total number of subgroups at level i, multiplied by the total number of subsets we get after processing B_i.
Denote by $T_n = T_{k^2 \times k^{p-2}} + T_r$ the overall running time for constructing $F^1_{k,n}$, where $T_{k^2 \times k^{p-2}}$ (resp. T_r) is the time to process k^p (resp. r) blocks. It is easy to see that $T_r = O(T_{k^2 \times k^{p-2}})$. We compute T_n following the same line of reasoning as in Proposition 6.

$$\begin{aligned} T_n &= [(k \times k)^2 \times k + O(k^{p-2}) \times (k \times k) \\ &+ (k^2 \times k^2)^2 \times k + O(k^{p-3}) \times (k^2 \times k^2) \\ &+ \vdots \\ &+ (k^i \times k^i)^2 \times k + O(k^{p-1-i}) \times (k^i \times k^i) \\ &+ \vdots \\ &+ (k^{p-1} \times k^{p-1})^2 \times k + O(k^0) \times (k^{p-1} \times k^{p-1})] \end{aligned}$$

So we have

$$\begin{aligned} T_n &= k \sum_{i=0}^{p-2} (k^{2(i+1))^2} + \sum_{i=0}^{p-2} O(k^{p-2-i}) \times (k^{2(i+1)}) \\ &= k^5 \sum_{i=0}^{p-2} (k^4)^i + \sum_{i=0}^{p-2} O(k^{p+i}) \\ &= k^5 \times \frac{(k^4)^{p-1} - 1}{k^4 - 1} + O(k^p \times \frac{k^{p-1} - 1}{k - 1}) \\ &= O(k^{4p-3}) + O(k^{2p-2}) \\ &= O(k^{4p-3}) \end{aligned}$$

VI. Implementation and results

Using the Spyder IDE, we implemented Algorithm 1 in Python and ran it on a 64-bit Windows operating system with the following specifications: Intel Core I7, 3.4 GHz, and 8 GB RAM.
We noticed that when k is even, the total number of subsets that form the solution to our problem is (always) equal to $3k$. On the other hand, when k is odd, this number is (most of the time) equal to $k^2 + k$. Figure 3 represents the cardinal of

F^1_{k,k^2} for values of k ranging from 0 to 50. We also represent (in dashed line) the function:

$$f(k) = \begin{cases} 3k & \text{if } k \text{ is even} \\ k^2 + k & \text{if } k \text{ is odd} \end{cases}$$

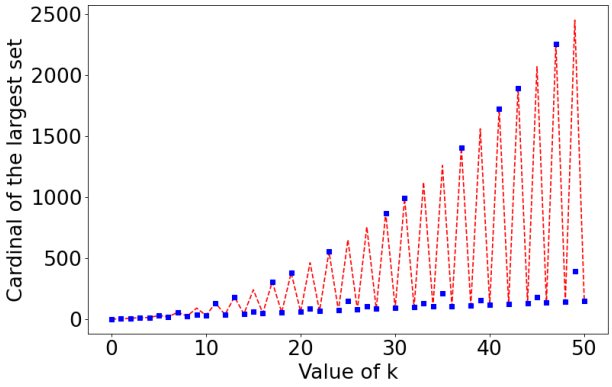

Fig. 3. Size of F^1_{k,k^2} for $k = 0$ to $k = 50$.

After further analysis, we found out that when k is even $Card(F^1_{k,k^2}) = 3k$ (for all $k \leq 200$). The values of k for which there is a mismatch with the function $k^2 + k$ are:
9, 15, 21, 25, 27, 33, 35, 39, 45, 49, 51, 55, 57, 63, 65, 69, 75, 77, 81, 85, 87, 91, 93, 95, 99, 105, 111, 115, 117, 119, 121, 123, 125, 129, 133, 135, 141, 143, 145, 147, 153, 155, 159, 161, 165, 169, 171, 175, 177, 183, 185, 187, 189, 195.
And the respective values of $card(F^1_{k,k^2})$ are:
36, 60, 84, 150, 108, 132, 210, 156, 180, 392, 204, 330, 228, 252, 390, 276, 300, 616, 324, 510, 348, 728, 372, 570, 396, 420, 444, 690, 468, 952, 1452, 492, 750, 516, 1064, 540, 564, 1716, 870, 588, 612, 930, 636, 1288, 660, 2366, 684, 1050, 708, 732, 1110, 2244, 756, 780.

We notice that the mismatches always happen after the first, second, or third consecutive odd values of k (this is reflected by spacings of 2, 4, and 6 for the above values of k). We just mentioned this fact in case one is interested in determining whether this pattern also holds for $k > 200$.

Proposition 11: When $n = k^2$ and k is even, the maximum (k,m)-subsets problem can be solved in $\Theta(\sqrt{n})$.

Proof This proof is based on the empirical result we obtained in this section. We know in this case that $card(F^1_{k,k^2}) = 3k = \Theta(\sqrt{n})$.
Representing the elements of U_n in a matrix A, we select the $3k$ subsets directly from the rows, columns, and diagonals (modulus k) of A. We know that all these subsets satisfy the conditions of Definition 1. Furthermore, no comparisons with previously constructed subsets are done here.
Intuitively, trying to add any subset to the solution by considering elements from different rows of A (as we did in Proposition 4) would violate the first condition of Definition 1.

Table I summarizes the size of $F^1_{k,n}$ and the cost of solving our problem for the different values of n with respect to k.

Before concluding, it is worth noting that the space requirements for solving our problem matches the time requirements given in Table I. This is because we need to keep in memory all the constructed subsets when computing $F^1_{k,n}$.

TABLE I
TIME TO COMPUTE $F^1_{k,n}$.

	Cardinal of $F^1_{k,n}$	**Time to compute $F^1_{k,n}$**
$n = k^2$ (k odd)	$\Theta(n)$	$\Theta(n^2\sqrt{n})$
$n = k^2$ (k even)	$\Theta(k)$	$\Theta(\sqrt{n})$
$n < k^2$	$O(n)$	$O(n^2\sqrt{n})$
$n > k^2$	$O(n^2/k)$	$O(n^4/k^3)$

VII. CONCLUSION AND FUTURE WORK

We tackled in this paper the maximum (k,m)-subsets problem and showed that for $m = 1$, this hard problem can be solved in polynomial time. The key idea was to consider the special case $n = k^2$, where the elements of the initial set can be represented by a square matrix. We designed and implemented (in Python) an algorithm that can solve this problem in $\Theta(n^2\sqrt{n})$ when n is odd, and in $\Theta(\sqrt{n})$ when n is even. The latter result is the most efficient one since it provides a strict sublinear solution to the original problem. We then distinguished two cases. For $k < n^2$, we showed that we can solve this problem in $O(n^2\sqrt{n})$. As for $k > n^2$, we exhibited a solution in $O(n^4)$ that we improved to $O(n^4/k^3)$ by taking advantage of the repetitive structure of the problem. Our next aim is to find other tractable classes of the maximum (k,m)-subsets problem. Another interesting question to address is that of determining the boundaries between hard and easy instances of this problem, as this was done for SAT [9] for example; where the authors proposed a polynomial-time solution to 2-SAT [2], knowing that 3-SAT is NP-hard [15]. Last but not least, one must determine the exact complexity class of this problem. Considering its hardest instances, we believe that the maximum (k,m)-subsets problem is NEXP-complete. To prove our claim, one must reduce a known NEXP-complete problem (e.g. the succinct K-coloring problem [17], [11]) to the maximum (k,m)-subsets problem, possibly using binary decision diagrams [16], [1] to capture the intersection of subsets of $U_{k,n}$.

REFERENCES

[1] S. B. Akers, Binary Decision Diagrams, IEEE Transactions on Computers, pp. 509–516, 1978.
[2] B. Aspvall, M. Plass, and R. Tarjan, A Linear-time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas, Information Processing Letters, pp. 121–123, 1979.
[3] F. Bengtsson, J. Chen, Efficient Algorithms for k Maximum Sums. In: R. Fleischer, G. Trippen, (eds.). LNCS, pp. 137–148, 2004.
[4] K. Challita, J.B. Abdo, The Maximum (k,m)-subsets Problem is in the Class NEXP. IAENG International Journal of Computer Science, vol. 48, no. 2, pp. 451–455, 2021.
[5] K. Challita, Infinite RCC8 Networks, International Journal of Artificial Intelligence, vol. 15, pp. 147–162, 2017.
[6] D.L. Yuan, and Y. Xu, Lightweight Vehicle Detection Algorithm Based on Improved YOLOv4. Engineering Letters, vol. 29, no.4, pp. 1544–1551, 2021.
[7] Efron Manik, Relationship Between Segment Edges and Thresholds on Segmentation Generated by Minimum Spanning Trees. Engineering Letters, vol. 28, no.3, pp. 796–802, 2020.
[8] Jie Qian, Hongyu Long, Yi Long, and Chenxu Zhao, Improved NSGA-III Algorithm and BP Fuel-cost Prediction Network for Many-objective Optimal Power Flow Problems. IAENG International Journal of Applied Mathematics, vol. 51, no.2, pp. 307–320, 2021.
[9] S. Cook, The Complexity of Theorem Proving Procedures. Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC71), pp. 151–158, 1971.
[10] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction to Algorithms, Second Edition, McGraw-Hill Science, 2001.
[11] H. Galperin, A. Wigderson, Succinct Representations of Graphs. Information and Control, pp. 183–198, 1983.

[12] H. Farhat. Composition of partially-observable services. *IEEE Access*, 7:2281–2290, 2018.
[13] H. Farhat. Web service composition via supervisory control theory. *IEEE Access*, 6:59779–59789, 2018.
[14] K. Challita, Reasoning with Lines in the Euclidean Space, 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), pp. 148–157, 2009.
[15] R. Karp, Reducibility Among Combinatorial Problems, In R. E. Miller; J. W. Thatcher; J.D. Bohlinger (eds.). Complexity of Computer Computations. New York: Plenum. pp. 85 - 103, 1972.
[16] C. Y. Lee, Representation of Switching Circuits by Binary-Decision Programs. Bell System Technical Journal, vol 38, pp. 985–999, 1959.
[17] C. Papadimitriou, M. Yannakakis, A Note on Succinct Representations of Graphs. Information and Control, pp. 181–185, 1986.
[18] T. Takaoka, Efficient Algorithms for the Maximum Subarray Problem by Distance Matrix Multiplication. Electronic Notes in Theoretical Computer Science, pp. 191–200, 2002.
[19] H. Farhat. Control of nondeterministic systems for bisimulation equivalence under partial information. *IEEE Transactions on Automatic Control*, 65(12):5437–5443, 2020.