An empirical evaluation of approximation algorithms to find maximal cliques in hypergraphs

Alexandra Iosif
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
a.iosif@student.utwente.nl

ABSTRACT

In this research, two methods based on the replicator transformation are applied to circulant k-hypergraphs in order to find their maximal cliques. An experiment is conducted on hypergraphs to test the algorithms considering three main parameters in their generation: the edge size k, the edge density d, and the total number of vertices n. The paper shows the relation between these algorithms and the above-mentioned parameters. Specifically, by increasing the edge density, a better time performance of the algorithms is seen. Further, it concludes that the replicator dynamics method is more efficient than the descent method.

Keywords

uniform hypergraph, maximal clique, replicator dynamics, descent method.

1 Introduction

Mathematical optimization has been a key player in the performance of existing algorithms and in the development of new solution approaches for problems in computer science. Numerical techniques such as the descent type methods are used to (approximately) solve this optimization problem. The descent methods are simple to employ and depend on a single parameter. The algorithms gained more attention with the advent of deep learning, where a variation of descent algorithm that is gradient descent is used and parameters are termed as the learning rate.

Cliques in graphs and hypergraphs have influenced the way of constructing combinatorial objects in many fields, from unrestricted error-correcting codes to Keller's conjecture and partitions of codes and designs [7]. The maximum clique problem is also remarkable for its applications in computer vision, experimental design, information retrieval and fault tolerance [2]. Moreover, graph theory has greatly contributed to the research in social networks, as stated by Luce and Perry [4].

The present paper offers a thorough characterization of two special methods, as described in [1], applied on searching maximal cliques in circulant k-hypergraphs. These methods, which are based on the replicator transformation, aim at computing local or global maximizers of homogeneous polynomials over the unit simplex. The global maximization, equivalent to maximum cliques in hypergraphs, is an NP-hard problem [5]. However, this paper [1] concentrates on delivering local maximizers of homogeneous polynomials with degree greater than 2, using the replicator dynamics as the first proposed method. The replicator dynamics algorithm is further refined to include a descent step in order to gain convergence speed. The descent algorithm depends on a step size that has been chosen by dividing the interval into discrete steps.

Since finding the maximum clique is an NP-hard problem, one resorts to developing heuristics in order to achieve not optimal but satisfactory results. One type of these heuristics deploys recursive backtracking. A more recent approach is the Russian doll algorithm proposed by Östergård [6]. It uses the Russian doll technique [12] over the basic backtracking algorithm in order to prune results with its dynamic programming structure. As presented in [9], a new outlook based on combining the Russian doll and the necklace algorithm that constructs binary necklaces [10] has resulted in the Russian necklace algorithm. Because of the special necklace structure, it has been applied to circulant k-hypergraphs on different edge densities in order to find maximum cliques. The results show that the Russian necklace algorithm is more efficient for hypergraphs with higher edge density. The paper [9] underlines the performance differences between the Russian necklace algorithm and the above mentioned: necklace, Russian doll, and backtrack algorithms.

This paper is organized as follows. Section 1.1 introduces cliques in circulant k-hypergraphs starting by defining notions of graphs, uniform hypergraphs, cliques, and necklaces. Then an optimization problem is formulated, aiming to find strict local minimizers of a homogeneous polynomial of degree k. Section 2 details step-by-step the algorithms *replicator dynamics* and *descent method* applied on k-hypergraphs, then further studies the parameters used to generate the instances for experiments. Section 3 discusses the main results, and section 4 concludes the present paper.

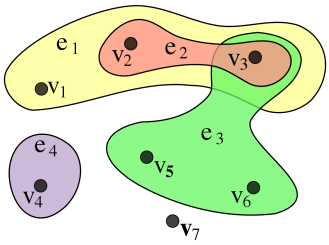
1.1 Preliminaries

This section defines the notions and notations used in this paper.

A *graph* is defined as a set $G = (V, E)$, where V is the set of n vertices $v_i \in V, i = \overline{0, n-1}$ and E is the set of edges, each edge corresponding to a pair (v_i, v_j) of vertices, $v_i, v_j \in V, i, j \leq n-1$. A graph is *complete* if all its pairs of vertices are connected by an edge. A *subgraph* A of a graph

36th Twente Student Conference on IT Febr. 4th, 2022, Enschede, The Netherlands.

Figure 1. Hypergraph Example. [3]



$G = (V, E)$ is a set (V_a, E_a) in which $V_a \subset V, E_a \subset E$. Note that not all edges connected with V_a found in E are necessarily in E_a. A subgraph $A = (V_a, E_a)$ of a graph $G = (V, E)$ is *induced* if all edges of E with endpoints in V_a are found in E_a.

A *clique* C in a graph is defined as a complete induced subgraph. A clique C is *maximal* in a graph $G = (V, E)$ if there are no other vertices $v_i \in V, v_i \notin V_c$ such that $V_c \cup v_i$ forms a clique. A *maximum* clique is the largest maximal clique in a graph. However, multiple cliques may have the maximum size within the same graph.

A *hypergraph* $H = (V, E)$ is a generalization of a graph, in which an element of E is a subset of vertices $v_i \in V$. Figure 1 shows an example of a hypergraph with four edges. A hypergraph $H = (V, E)$ is *complete* if the edge set E is the power set of vertices V. A k-hypergraph is a hypergraph with all its edges being subsets of vertices with the same cardinality k. A k-hypergraph is *complete* if all k-subsets of its vertices are edges. A clique in a k-hypergraph is a complete induced k-subhypergraph.

The complement of a hypergraph $H = (V, E)$ has all edges not found in H on the same set of vertices V, such that $\overline{H} = (V, \overline{E})$ with $\overline{E} = P_k(V)/E$.

1.1.1 Binary Canonical Necklace

Binary canonical necklaces have been used by Plant [9] to generate circulant k-hypergraphs and to find maximum cliques through the algorithm Russian necklace search. However, in this paper they are used only to generate the hypergraphs. In order to define circulant k-hypergraphs, the concept of binary canonical necklaces will be introduced through an example.

Consider the set $N = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$ and the subset $A = \{1, 3, 5, 6\}$, $A \subset N$. The *characteristic string* of A is $\alpha = 010101100$ so that each element of N is represented by 1 if it is found in subset A and 0 otherwise.

The right justified characteristic string is formed by shifting the characteristic string to the right until the first 1 is met at the end of the string (or by $max(N) - max(A)$ places, specifically $8 - 6 = 2$ places in this case). Thus, the right justified characteristic string is 000101011.

A rotation to the right hand side by three places is applied. The characteristic string becomes $\beta = R(\alpha, 3) = 100010101$, so $B = \{0, 4, 6, 8\}$.

Under rotation, the strings α and β are equivalent and thus referred to as binary necklaces. Further, necklaces can be defined as isomorphism classes of subsets of a set under rotation. A *canonical necklace* is the right justified characteristic string of a subset.

A *circulant* graph has its adjacency matrix a circulant matrix. For circulant k-hypergraphs, symmetric adjacency tensors are used; tensors can be seen as multi-dimensional matrices. As defined by Plant [8], a k-hypergraph $H = (V, E)$ is *circulant* if there exists a relabeling of the vertices $V = \{0, 1, ..., |V|-1\}$ and a set $\mathcal{N}$ of canonical subset necklaces, each being a subset of V and having cardinality k, such that $E = \{R(N, i) : N \subset \mathcal{N}, 0 \leq i \leq |V| - 1\}$. In other words, a k-hypergraph is circulant if the hypergraph has an automorphism that is a cyclic permutation of its vertices [9].

1.1.2 Continuous Optimization Formulation

This section is based on the work presented in [1].
For a given $\boldsymbol{x} \in \mathbb{R}^n$, the *Lagrangian function* of a complement k-hypergraph $\overline{H} = (V, \overline{E})$ is defined as:

$$L_{\overline{H}}(\boldsymbol{x}) = \sum_{e \in \overline{E}} \left(\prod_{i \in e} x_i \right), \tag{1}$$

where

$$\Delta = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} \geq 0, \sum_{i=0}^{n-1} x_i = 1\} \tag{2}$$

represents the *unit simplex* [2].
Consider the following homogeneous polynomial of degree k:

$$h^{\tau}_{\overline{H}}(x) = L_{\overline{H}}(x) + \tau \sum_{i=0}^{n-1} x_i^{k-1}, \tau > 0, \tag{3}$$

then the optimization problem:

$$\min h^{\tau}_{\overline{H}}(x), \text{for } x \in \Delta_n. \tag{4}$$

The *characteristic vector* x^S of a subset $S \subset V$ consists of the components:

$$x_i^S = \begin{cases} \frac{1}{|S|}, & i \in S, \\ 0, & i \notin S. \end{cases} \tag{5}$$

The following theorem connects the maximum clique problem in a hypergraph with the continuous optimization problem as presented in 4.
Theorem Let $H = (V, E)$ be a k-hypergraph and let $0 < \tau \leq \frac{1}{k(k-1)}$. Then $x \in \Delta_n$ is a strict local minimizer of $\min h^{\tau}_{\overline{H}}(x)$ if and only if $x = x^C$ is the characteristic vector of a maximal clique C of H.

1.2 Problem Statement

The maximum clique problem has interesting applications, as mentioned in Section 1. Since it is an NP-hard problem, the only approach is to rely on heuristics to solve it. Likewise, circulant k-hypergraphs or circulant tensors are used in various fields. Therefore, the focus is on finding cliques in circulant k-hypergraphs only, by analyzing existing algorithms. In particular, the methods based on the replicator transformation studied in [1] will have a dedicated implementation for the special structure of hypergraphs. Afterwards, they will be tested on the same benchmark instances used by Plant [8]. The benchmark instances are generated to compare his algorithm, the Russian necklace, with the necklace algorithm developed by Tzanakis et. al. [11]. An empirical comparison of the performances of these algorithms and their results will follow.

2 Methodology

This section studies the replicator dynamics method and the descent method applied to circulant k-hypergraphs.

2.1 Algorithms

In this section, we derive an algorithm from the work of Ahmed & Still. The algorithm is based on the replicator equation as presented in the section 1.1. In the following, we describe the algorithm and comment on its computational complexity. The basic algorithm is given below.

1: Compute the complement hypergraph $\overline{H} = (V, \overline{E})$
2: Choose $\tau \in \left[0, \frac{1}{k(k-1)}\right]$
3: Choose starting $\boldsymbol{x} \in \mathbb{R}^n_+$
4: Normalize $\boldsymbol{x}$ such that $\boldsymbol{x} \in \Delta_n$
5: Set $\epsilon = 10^{-3}$
6: Set *flag* = *true*
7: **while** flag **do**
8: Compute h as:
$h^{\tau}_{\overline{H}}(x) = \sum_{e\in\overline{E}} \left(\prod_{i\in e} x_i \right) + \tau \sum_{i=0}^{n-1} x_i^k$
9: **for** $j = 0, 1, ..., n-1$ **do**
10: $y_j = \frac{1}{h^{\tau}_{\overline{H}}(x)} x_j \left(\sum_{e\in\overline{E}} \delta_{je} \left(\prod_{i\in e/j} x_i \right) + \tau k x_j^{k-1} \right)$,
▷ where $\delta_{je} = \begin{cases} 1, & j \in e, \\ 0, & j \notin e. \end{cases}$
11: **end for**
12: **if** $||\boldsymbol{y} - \boldsymbol{x}|| < \epsilon$ **then**
13: flag = false
14: **end if**
15: $\boldsymbol{x} = \boldsymbol{y}$
16: **end while**
17: Computed $\boldsymbol{x}$ is the characteristic vector for the maximal clique

The algorithm requires a k-hypergraph $H = (V, E)$, where V is the set of vertices with $|V| = n$, and E the set of edges. This requirement has been ensured by taking the set of edges of the hypergraph, each edge having size k, and the number of vertices n as input.
The algorithm ensures the characteristic vector $\boldsymbol{x} \in \Delta_n$ of the maximal clique. Thus, the algorithm returns a list of elements ranging between 0 and 1 which, when added, give value 1. In other words, it returns a normalized vector of length n.

Step 1 The first step of the algorithm is to compute the complement hypergraph for H: $\overline{H} = (V, \overline{E})$. This step is done by removing the elements of the edge set of H from a set containing all combinations of n elements taken k times. This way, it results in the set of edges of the complement hypergraph, $\overline{E}$. Thus, the space complexity of this step is $\binom{n}{k}$. In the worst case, a hypergraph with the edge density close to $d = 0\%$ would have the complement hypergraph with $d = 100\%$, where $\overline{E} = \binom{n}{k}$. For time complexity, $O\binom{n}{k}$ is achieved if an edge of H is the last element of the $\binom{n}{k}$ set, since it would require $\binom{n}{k}$ steps to eliminate the edge. However, the average time complexity is $O\left(|\overline{E}| < \binom{n}{k}\right)$. This complexity requires an exponential growth of time.

Step 2 In the second step, τ has to be chosen considering $\tau \in \left[0, \frac{1}{k(k-1)}\right]$. This is a static step as τ is always taken in implementation as $\tau = \frac{1}{k(k-1)}$.

Step 3 The third step initializes a vector $\boldsymbol{x} \in \mathbb{R}^n_+$. In terms of complexity, the space complexity grows linearly with n, while the time complexity is constant, depending on the randomization function. To generate a random array of length n, numpy.random.rand() has been used over random.random() offered by Python, since it creates an ndarray of n random elements without the need of a loop, thus it is more efficient.

Step 4 In the fourth step, the vector $\boldsymbol{x}$ defined previously needs to be normalized. For this, the following manner of normalization has been considered: $\boldsymbol{x} = \frac{\boldsymbol{x}}{\sum_{i=0}^{n-1} x_i}$. This is a two-step operation, firstly all elements of $\boldsymbol{x}$ need to be added, and secondly, each element needs to be divided by the sum. This results in a linear time complexity of $O(n)$, and a space complexity of n.

Step 5 This step is devoted to set $\epsilon = 10^{-3}$. It is a fixed step and does not influence either the complexity.

Step 6 This step is meant to initialise $flag = true$. This is used as a breaker of a the while-loop that will follow next.

Step 7 First, a parameter *counter* is initialized to 0 in order to ensure that the while-loop is not called more than 2000 times. Then, the while-loop starts and runs under the conditions: *flag* = *true* and counter <= 2000.

Step 8 The homogeneous polynomial of degree k, $h^{\tau}_{\overline{H}}(x)$, is computed in this step. Considering 3, the first addend is the Lagrangian function1 for the complement hypergraph $\overline{H}$. To get the sum inside the function, for each edge e in the edge set $\overline{E}$, it is computed the product between the elements x_i of the normalized vector $\boldsymbol{x}$ found on the positions corresponding to the vertices $i \in e$ of the specific edge. Adding all these products, it results in the Lagrangian function $L_{\overline{H}}(\boldsymbol{x})$. As the complement has $|\overline{E}| < \binom{n}{k}$ edges, the time complexity is $O(|\overline{E}| * k)$, $|\overline{E}|$ by taking each edge and k given that each edge has this size. The Lagrangian function has an exponential growth.
The second addend, let it be called the tau sum, computes the sum of all elements x_i^k and multiplies it by τ. The time complexity in this case is $O(n * k)$, thus it is linear. The space complexity remains n.
Considering the complexities of the Lagrangian function and of the tau sum, the sum of these gives the complexity of the homogeneous polynomial h, which is the exponential $O((|\overline{E}| + n) * k)$.

Step 9-11 This step computes vector $\boldsymbol{y} \in \mathbb{R}^n_+$. For each element y_j, the first step is to compute the sum of the products between x_i's. The product is solved in the following way: for each edge e in the edge set $\overline{E}$, if $j \in e$, then it is computed the product between x_i elements, where $i \in e$ but $e \neq j$. Then the products for all edges with $j \in e$ are added. Taking each edge requires $|\overline{E}| < \binom{n}{k}$ steps, while all edges having exactly k elements, the complexity of this part is the exponential $O(|\overline{E}| * k)$ and the space complexity is $|\overline{E}|$. The final addend to this sum is $\tau k x_j^{k-1}$, which is only a set of operations, without influencing the complexity.
After this sum is completed, it is divided by polynomial h and multiplied by x_j. Once more, this is not significant to the final complexity.
If the complexity for computing each element y_j is $O(|\overline{E}| *$

k) and there is a total of n elements, the complexity in the average case for obtaining vector $\boldsymbol{y}$ is $O(|\overline{E}| * n * k)$.

Step 12-14 This step checks the following statement considering the Euclidean norm: $||\boldsymbol{y} - \boldsymbol{x}|| < \epsilon$. If it is true, then *flag* is changed to *false*. In order to compute the norm, a set of operations is performed on vectors $\boldsymbol{x}$, and $\boldsymbol{y}$. Then the result is compared to ϵ. Thus, the complexity here is linear, $O(n)$.

Step 15-16 This is the last step of the while-loop. The $\boldsymbol{x}$ vector is changed to vector $\boldsymbol{y}$: $\boldsymbol{x} = \boldsymbol{y}$. The space complexity is n^2, while the time complexity gets to linear $O(n)$. Depending on the value of the *flag*, the algorithm might return to **Step 7**, otherwise exit it.

Step 17 The algorithm finishes returning vector $\boldsymbol{x}$ as the characteristic vector of the maximal clique of the hypergraph H.

Considering the complexities for each step mentioned above, the time complexity of the algorithm *replicator dynamics* is the largest complexity of all steps, thus it is $O(|\overline{E}|*n*k)$.

2.1.1 The Descent Method

The *descent method* is similar to the above algorithm, the only change is to extend Step 15. A step-size α is set to $\alpha = 0.01$. Then it follows:

Set $\boldsymbol{z} = \boldsymbol{x} + \alpha(\boldsymbol{y} - \boldsymbol{x})$
if $\sum_{i=0}^{n-1} z_i = 1$ **then**
  $\boldsymbol{x} = \boldsymbol{z}$
else
  $\boldsymbol{x} = \boldsymbol{y}$
end if

The complexity of this if-statement is $O(n)$, without influencing considerably the total complexity of this algorithm. Thus, the time complexity of both algorithms is the same: $O(|\overline{E}| * n * k)$.

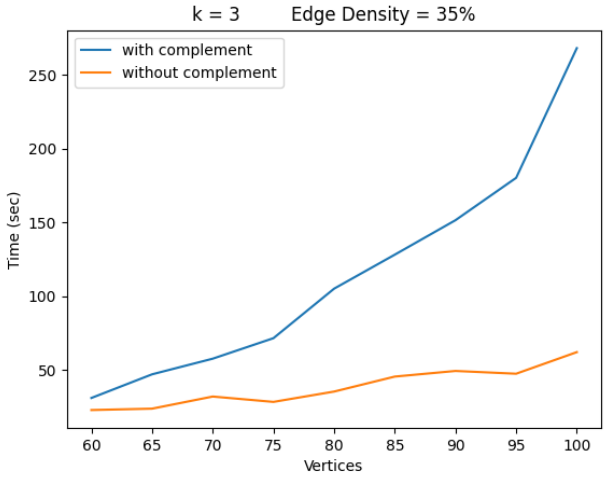
2.2 Experiments

An experimental evaluation of the algorithms is performed by comparing the algorithms based on replicator dynamics and descent methods. The experiments are conducted using randomly generated circulant k-hypergraphs with regard to the same parameters imposed by Plant in his own experiment: number of vertices n, edge size k, and edge density d. The edge size k varies between 3 and 5 vertices. The first parameter n is set to 60 for $k = 3, 4$ until $n = 100$ with an increase of 5 vertices at each step. Because of the memory required by $k = 5$, it is reduced to ranging from 20 to 50 vertices, maintaining the increase of 5 vertices with each step. In this case, the search space is likely 2^5 or 32 times larger with each rise [8]. The edge density d starts with 0.35, increasing by 0.05 until 0.95 is reached. This is meant to represent the division between the number of edges found in the k-hypergraph and all possible edges. However, a circulant k-hypergraph is formed from its canonical necklaces with all their rotations involved and the number of rotations per canonical necklace can differ. In Table 1, the canonical necklaces with their rotations of a 3-hypergraph on 6 vertices are shown. As it can be seen, the probability differs for each set of edges. Thus, the edges do not have the same weight of being chosen, taking into account their rotations necessarily to be included in the formation of the hypergraph. A reasonable solution to this problem is to choose randomly only between the canonical necklaces so that the edge density becomes only an approximation of the total number of edges found in the hypergraph. To achieve this, a random $0 \leq r \leq 1$ is generated for each canonical necklace. It is selected to be an edge, together with its rotations, for $r \leq d$.

Table 1.

Edge Probability			
Edge	Rotations	Period	Probability
0, 1, 2	0,1,5 0,4,5 1,2,3 2,3,4 3,4,5	6	$\frac{6}{20} = 30\%$
0, 1, 3	0,2,5 0,3,4 1,2,4 1,4,5 2,3,5	6	$\frac{6}{20} = 30\%$
0, 1, 4	0,2,3 0,3,5 1,2,5 1,3,4 2,4,5	6	$\frac{6}{20} = 30\%$
0, 2, 4	1,3,5	2	$\frac{2}{20} = 10\%$

Figure 2.



Since the complexity of computing the complement of the edge set is $O\binom{n}{k}$, it generates an exponential time increase with each new step of vertices addition. Consider Figure 2 showing the runtime needed for the algorithm of replicator dynamics when $k = 3$, $d = 35\%$, and $n = \overline{60, 100}$. The method for computing the complement of the hypergraph takes the spotlight, reaching over 250 seconds runtime for 100 vertices. Meanwhile, the rest of the algorithm keeps its computation time below 60 seconds. This way, the difference between the two methods in runtime becomes difficult to be observed. For this reason, the runtime of generating the complement hypergraph is excluded for the rest of the plots. Further, for $k \geq 4$, this step is entirely omitted. Instead, hypergraphs are directly generated with the edge density $\overline{d} = \overline{5, 65}\%$, as opposed to $d = \overline{35, 95}\%$, and these are accepted as the complement hypergraphs. Thus, the algorithms can be computed using hypergraphs with the highest suitable range of vertices n for all k's, without the possible time inconvenience created by generating the complement hypergraph.

3 Results

The algorithms have been implemented and tested in Python 3.7.5 using Pycharm as IDE. All experiments have run on an Intel Core i7-8750H CPU @ 2.20GHz having Windows

Figure 3.

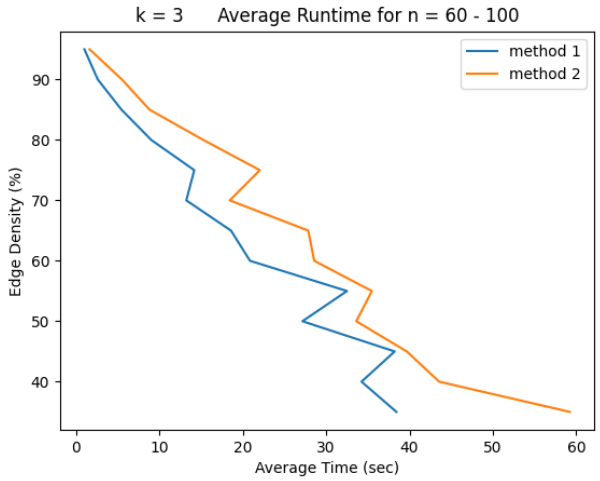
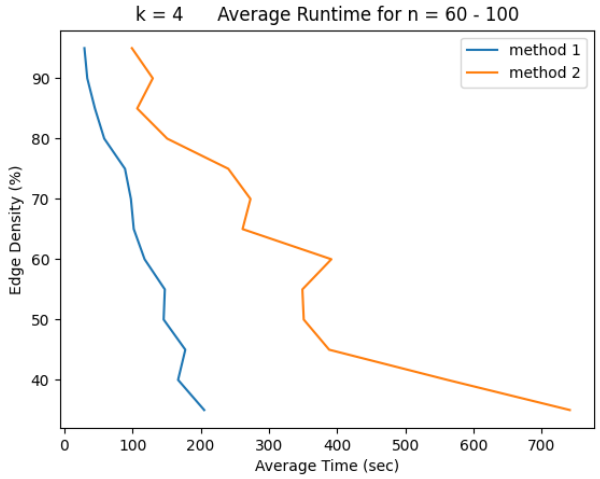


Figure 4.

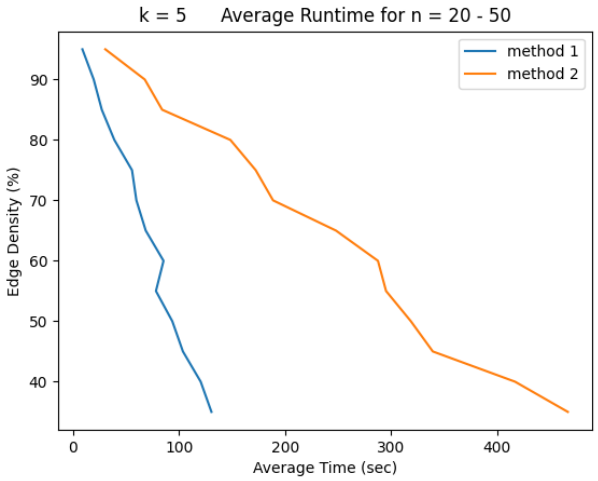


11.
The algorithms are tested considering the parameters: n, k, and d. For each edge size k and edge density d, the average runtime of all runtimes for $n = \overline{60, 100}$ and $n = \overline{20, 50}$, respectively, is computed. The remaining figures show the average runtimes for each $k = \overline{3, 5}$ and $d = \overline{35, 95}\%$. *Method 1* represents the algorithm for the replicator dynamics method, and *method 2* for the descent method.

At a first glance, considering Figures 3, 4, and 5, the algorithm based on replicator dynamics appears to be more optimal than the descent method. Furthermore, both algorithms are more time-efficient with the increase in edge density. This happens because the algorithms work on the complement hypergraph. Thus, while the hypergraph gets more edges with the increase in edge density, its complement becomes the opposite, having reduced the number of edges. Additionally, the second algorithm seems to be more affected in runtime with the change in edge density. Another point can be that the more we increase the edge size k, the more it grows the discrepancy between the average times of these two algorithms.

Looking at the average runtimes, we observe an exponential increase with the edge size k. Considering Figure 3 and 4 which both have the average times for $n = \overline{60, 100}$, the first method shows a range of 2-40 seconds for $k = 3$, while for $k = 4$ we see a large growth to 30-200 seconds. For the descent method, the ranges are 3-60 seconds for $k = 3$ and 110-750 seconds for $k = 4$. For $k = 5$, although the number of vertices is reduced to $n = \overline{20 - 50}$, the first method requires on average more than 100 seconds for $d \leq 50\%$, while the second method takes over 300 seconds for the same d.

Figure 5.



In these plots, the lines are not straight. This can be a result due to the approximation of the edge density d. Since every canonical necklace receives a random number from 0 to 1 in the generation of the hypergraphs, it could happen to a hypergraph to have more or less edges than the number required by d.

4 Discussion and Conclusion

In this paper, we have discussed an optimization problem with the aim of finding maximal cliques in circulant k-hypergraphs. We have introduced the concepts behind cliques in hypergraphs and formulate a continuous optimization based on the minimization of a homogeneous polynomial as defined in 3. Then we have applied them to create a dedicated implementation of the replicator dynamics to circulant k-hypergraphs. This has been analyzed step by step with regard to its complexity. Afterwards, Section 2.1.1 looks at a possible optimization introducing a step-size α in implementation. As it is shown in results, the first method, replicator dynamics, is more efficient than the descent method for hypergraphs with $k = \overline{3, 5}$, $d = \overline{35, 95}$, and $n = \overline{60, 100}/n = \overline{20, 50}$, respectively. Both algorithms perform better with the increase of the edge density d due to the requirement imposed by these methods to work with the complement hypergraph.

5 References

[1] F. Ahmed and G. Still. Two methods for the maximization of homogeneous polynomials over the simplex. *Computational Optimization and Applications*, 80(2):523–548, 2021.

[2] S. R. Bulo and M. Pelillo. A continuous characterization of maximal cliques in k-uniform

hypergraphs. In *International Conference on Learning and Intelligent Optimization*, pages 220–233. Springer, 2007.

[3] Kilom691. An example of a hypergraph. retrieved on 23/01/2022 from https://en.wikipedia.org/wiki/hypergraph/media/file:hypergraph-wikipedia.svg.

[4] R. D. Luce and A. D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.

[5] K. G. Murty and S. N. Kabadi. Some np-complete problems in quadratic and nonlinear programming. Technical report, 1985.

[6] P. R. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207, 2002.

[7] P. R. Östergård. Constructing combinatorial objects via cliques. In *Surveys in combinatorics*, pages 57–82, 2005.

[8] L. Plant. Maximum clique search in circulant k-hypergraphs. Master's thesis, Université d'Ottawa/University of Ottawa, 2018.

[9] L. Plant and L. Moura. Maximum clique exhaustive search in circulant k-hypergraphs. In C. J. Colbourn, R. Grossi, and N. Pisanti, editors, *Combinatorial Algorithms*, pages 378–392, Cham, 2019. Springer International Publishing.

[10] F. Ruskey, C. Savage, and T. M. Y. Wang. Generating necklaces. *Journal of Algorithms*, 13(3):414–430, 1992.

[11] G. Tzanakis, L. Moura, D. Panario, and B. Stevens. Constructing new covering arrays from lfsr sequences over finite fields. *Discrete Mathematics*, 339(3):1158–1171, 2016.

[12] G. Verfaillie, M. Lemaitre, and T. Schiex. Russian doll search for solving constraint optimization problems. In *AAAI/IAAI, Vol. 1*, pages 181–187, 1996.