Solving dense subgraph problems using genetic algorithm

Joseph L Pachuau[1], Nongmeikapam Brajabidhu Singh[1], Laiphrakpam Dolendro Singh[1] and Anish Kumar Saha[1]

Abstract— **In graph theory, the dense subgraph problem aims at finding the densest subgraph for a given graph. The subgraph is defined as a subset of a large graph and the density is calculated as the number of edges as per the number of vertices. Real-life graph networks are complicated and large in size; finding the dense subgraph is difficult as the number of subgraphs is numerous. It is an NP-hard problem and in most cases, it is infeasible to find an exact solution. Here we propose a genetic algorithm approach to solve it. The problem is a combinatorial optimization problem with a constraint that all solutions must be a subgraph of the given graph. Here, the crossover and mutation are designed to produce only feasible solutions. The proposed algorithm is able to give close approximation within a reasonable time.**

Keywords- Dense subgraph, Genetic algorithm, Graph crossover, Graph mutation, Fiedler vector, Adjacency matrix

I. INTRODUCTION

The density of a graph is given by the ratio of edges to vertices. The aim of a dense graph problem involves finding which subgraph of a given graph has the highest possible density. Dense subgraphs are useful in various domains to analyse complex networks. The dense components of a graph are easily traversed in networks because they are highly connected. It has also been observed that if a node in a dense network transmits a message to all of its neighbours, it is highly probable that the message will reach all of the diameters of the graph. In social media, these dense components can be used to identify communities [1]. It is also applied in market data analysis [2] and in biology, DNA motif finding methods [3] and finding molecular complexes in protein networks [4]. Other real-world datasets like Karate Club, Intel Lab, LastFM, HomoSapiens, Biomine, and Friendster graph datasets are from social networks, computational chemistry, and biology. The dense graph problem is an NP-hard problem because a graph can generate an exponential number of sub-graphs [5]. Earlier attempts for solutions to dense subgraph problems include the max flow method [6] and the LP approach [7].

A number of variation to this problem has been considered by specifying the size of the sub-graph. It is referred to as a k-subgraph problem when the subgraph size is fixed as k. Another variation is the least k-subgraph problem, where the minimum size of the subgraph is limited to k. As real-life graphs are large, finding a dense subgraph is a difficult task [8]. In uncertain real-world graphs, the most probable densest subgraph approach is more effective than the deterministic densest subgraph approach. Genetic algorithm (GA) is found to be effective in such cases [9].

[1] Joseph L Pachuau, Nongmeikapam Brajabidhu Singh, Laiphrakpam Dolendro Singh and Anish Kumar Saha* are with the Department of Computer Science and Engineering, National Institute of Technology Silchar, Assam-788010, India. anishkumarsaha@gmail.com*

GA stores the solution as a chromosome and operates directly on these chromosomes will improve the solution. A suitable design of chromosome design is required to fit the problem statement. For the graph coloring problem presented in [10], an array is used as a chromosome; the indices number represents the vertices and the number stored in the corresponding index denotes the color. In a graph partitioning problem, a string of binary bits represents a chromosome [11]. Bit 0 or 1 shows the different section of the graph and equal numbers of 0 and 1 represents graph bisection. The number of genes on each chromosome represents the number of vertices in the graph. The cut sizes are used to calculate the fitness values. In large-scale graph partitioning problems, heterogeneous clusters can be handled using the genetic graph partitioning algorithms in [12]. Each solution represents a possible cut, so the chromosome is represented as a set of vertices denoting the endpoint of a cut. These algorithms can speed up graph operation. In [13], Y. Lu proposed a GA for solving network graph clustering problems. Newman's modularity index in the graph partition is used as the fitness criterion. The edge list of the graph network is taken as the chromosome, with the number of nodes n in the network as the length of the chromosome. A chromosome is clustered into communities. They used a crossover scheme called the crossover pool, where the cardinalities depend on crossover probabilities. Their mutation probability is low, and they randomly replaced genes in the chromosomes with valid node numbers in the adjacency matrix for mutation.

In this paper, a GA algorithm is proposed for finding the densest sub-graph of a graph. The Chromosome structure, operators, adjacency matrices, and other functions are designed using linear algebra. A suitable graph crossover and mutation are proposed for such chromosomes. Fiedler vector is used here for determining the algebraic connectivity in a graph. The experiments are tuned for different values to obtain the suitable value of population size, mutation rate, and generation numbers for optimal solutions.

II. RELATED WORKS

A. Steenbeek et al. [14] proposed a hybrid genetic graph partitioning algorithm for solving graph bi-partition problems. It is a heuristic algorithm to find dense subgraphs with relatively high edge density. They suggested two procedures for the proposed GA. The first procedure is used for preprocessing to find dense clusters and the second procedure is an operator for the local improvement of chromosomes. The experimental results were compared against the existing heuristic algorithms and the empirical evidence showed that the hybrid GA outperforms the other heuristic algorithms. A detailed analysis of an incremental hybrid GA's effectiveness at resolving subgraph isomorphism problems is presented in [15]. This algorithm breaks down the entire subgraph isomorphism problem into a series of subproblems with the best possible structures. Then, by implementing a hybrid GA, each of these subproblems is solved one after the other, with the solution obtained from solving one subproblem being fed to the next subproblem as its initial solution. Experiments are conducted using both sizable real-world graph datasets and synthetic random graph datasets. In [16], H. Choi et al. implemented the hybrid incremental GA to solve subgraph isomorphism problems. They experimented with existing graph datasets and made a comparison with the existing approach.

In [17], K. Semertzidis et al. introduced Best Friend Forever and On-Off Best Friend Forever Problems for defining dense subgraphs in graph histories. They studied the complexities of the multiple variants of the Best Friend Forever and On-Off Best Friend Forever Problems. Appropriate algorithms were proposed. They proved the significance of these problems by conducting experiments with real and synthetic datasets. An algorithm for finding the k densest subgraphs in graph theory is presented in [18]. The k densest subgraph problem has a variant defined. A graph instance is used as the input, and the output is a collection of unique pairwise subgraphs. The goal is to maximize the sum of the densities for each of these subgraphs. For the k-densest subgraphs, the effectiveness of their polynomial time approximation scheme is discussed. A k-clique densest subgraph problem is introduced in [19]. This is a generalization of the problem of the densest subgraph. The triangle densest subgraph problem is a type of problem formulation. The finding is that large near-cliques in graphs are possible to discover using the computational problem known as the triangle counting problem. There is a method introduced for solving these issues in graph mining applications.

Approximation algorithms are presented in [20] for solving the densest k-subgraph and sparsest k-subgraph problems. When associated with structural parameters like block deletion number, neighbourhood density deletion number, distance-hereditary, and cograph deletion number, it turns out that these problems are fixed-parameter polynomial time. An algorithm for resolving problems with locally densest subgraphs is discussed in [21]. This algorithm is based on the Frank-Wolfe algorithm used in convex programming. The implementation includes nine distinct graph datasets from social networks, e-commerce sites, and video platforms. When compared to existing algorithms, their algorithm performs effectively. A scalable spectral and combinatorial algorithm for resolving the densest k-subgraph problems in real-world graphs is described in [22]. To put their algorithm into practise, a framework called Spannogram is presented. Graph datasets like com-LiveJournal, com-DBLP, Notre Dame, and Facebook network graph datasets are utilised in experiments for various ranked versions of their algorithms. Their algorithm demonstrates scalability and efficiency in the search for dense subgraphs. The densest k-subgraph problems are solved in [23] by using a polynomial time approximation scheme on stars of cliques. Algorithms are suggested for finding optimal solutions to the same problem on the trees of cliques and the paths of cliques. Their complexity analysis gives an $O(nk^{m+1})$ optimal algorithm for the densest k-subgraph problems on the trees of cliques where n is the sum of all vertices in the cliques and m is the maximum degree of this tree. This problem on the paths of cliques has a $O(nk^3)$ algorithm. Algorithms for solving s-PLEX, Non-Induced and Induced (k_1, k_2) biclique, and s-Defective Clique problems are introduced in [24]. Problems with dominating sets, such as those with Independent Dominating Sets and Dominating Cliques, are solved. Theorems and corollaries that are relevant are thoroughly discussed. It proves that graphs with small weak closure numbers are useful for finding dense or sparse subgraphs or dense or sparse dominating sets in graph problems.

K-vertex dense subgraph maximation problems are investigated in [25]. These are NP-hard problems. A dense subgraph of k vertices is generated from an input of a graph instance G with n vertices. The aim is to find the densest subgraph. Procedures for approximating the graph's densest k-vertices subgraph are introduced. In [26], dense k-subgraph and Max-Haf problems are solved using the Gaussian boson sampling(GBS). In order to create GBS samples, the weighted adjacency matrix of the graph is encoded into a GBS device. The GBS sampling distribution is used to calculate the populations of Toronto and Hafnian. The combinatorial search space for stochastic algorithms is constrained by the density of a graph in binary data, which has a positive correlation to Hafnians. A time-bin encoded GBS device is used in [27] to produce GBS graph samples in order to enhance the dense subgraph search classical algorithm. In their random search method, n GBS samples are generated with k nodes. The density of each sample is then determined. Finally, the dense subgraph is provided by the sample with the highest density. Multiple iterations of the sampling procedure are used to account for statistical fluctuations. A GA is proposed in [28] for solving the subgraph isomorphism problem. The chromosome is represented by the permutation of the vertices in the larger graph G for the subgraph isomorphism relation. In order, the number of vertices n in the smaller subgraph H corresponds to the number of vertices n in the chromosome. When dealing with fitness functions, a multi-objective approach is used. The final fitness function is calculated as the product of the relative weight and fitness value of each fitness function. Initial chromosomes are chosen at random for the population. For chromosome selection, a roulette-wheel approach is used. A 20% chance of cycle crossover and mutation is used. With each iteration, new offsprings are produced to replace the worst chromosomes. The experimental results show an improvement in results over the existing methods. A GA-based iterative local search method is used to solve subgraph isomorphism problems in [29]. Iterative local search and GA performance can complement each other. Their proposed hybrid approach and the existing GA are compared in terms of performance. In solving subgraph isomorphism problems, their approach outperforms the existing GA.

III. Problem statement and the proposed algorithm

A dense subgraph problem is known for finding the maximum density where the ratio between the number of

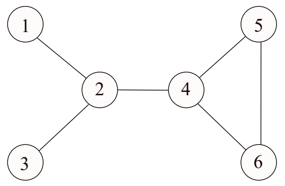

Fig. 1: A small example of graph

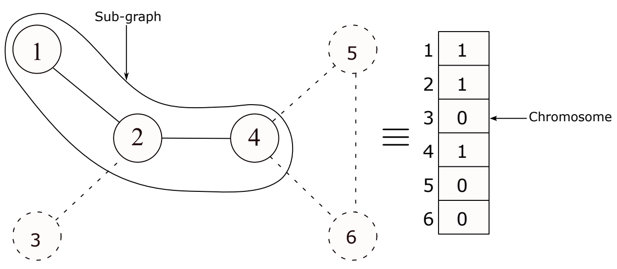

Fig. 2: Chromosome structure

edges to the number of vertices is maximum. The same is stated below.

For a given $G(V,E)$ and G', the objective is to find,

$$\begin{aligned} \max\ Z = & \frac{\text{No of edges}(G')}{\text{No of nodes}(G')} \\ \text{s.t. } & G' \subseteq G \end{aligned} \tag{1}$$

A. Representation of graph

Graphs contain a set of nodes/vertices that are connected by edges. Graphs are generally represented by an adjacency matrix, incidence matrix, or adjacency list. Here, the adjacency matrix is used to represent the graph. For a graph with n number of nodes, the adjacency matrix (A) is an $n \times n$ matrix, which is defined as below,

$$A(i,j) = \begin{cases} 1, & \text{if edge between node } i \text{ \& } j \text{ exists} \\ 0, & \text{otherwise.} \end{cases}$$

An example is given for the Fig. 1 as below,

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

B. Chromosome

The chromosome represents a solution to a problem statement. The solution of a dense subgraph is a subgraph for a graph. Here it is represented by a column vector of length m. Each value, known as a gene in the chromosome denotes the presence of the corresponding nodes in the subgraph, as below.

$$g(i) = \begin{cases} 1, & \text{if node } i \text{ is included} \\ 0, & \text{if node } i \text{ is not included.} \end{cases}$$

Fig. 2 shows an example of a chromosome structure for the subgraph {1,2,4}.

C. Objective and fitness function

The objective is to find the dense subgraph i.e., a subgraph of the maximum value E/V. Here the same function is used for both the objective function and fitness function to calculate the strength of the chromosome.

$$\approx \frac{\langle 1|^{\otimes n} A \odot (g \cdot g^T) |1\rangle^{\otimes n}}{g^T . g}$$

Symbol $\odot$ denotes element-wise multiplication, g^T denotes transpose of the graph, $\langle 1|^{\otimes n}$ denotes $[1,1,1,1,1,..1]$, and $|1\rangle^{\otimes n}$ denotes $\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$ respectively.

D. Population

The initial population is a process of random generation of chromosomes. To generate a chromosome, initially, a node is taken and then a random neighbour node is selected which will be merged with that node. This process continues by selecting another random neighbor of the subgraph and then it will be again merged with the subgraph. The random selection of neighbour will continue until a random exit happens for that chromosome. The generation of populations continues until the desired number of chromosomes is obtained. The initial nodes are selected evenly among all the nodes to construct a diverse population. The working principle of the finding of neighbours of a subgraph is explained below.

$$\begin{aligned} N = & A \odot ((J_{m,1} - g) \cdot g^T) \\ = & A \odot \left(\left(\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right) \cdot \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \right) \\ = & A \odot \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{aligned}$$

$$\text{Neighbours } n = N \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Since adjacency matrix A is a logical/ binary matrix, thus any obtained result after any operation will be bound to 0 and 1 only. Here for the example, the neighbors for $g\{1,2,4\}$ is $n\{3,5,6\}$. One of the random neighbors from n will be merged to v.

E. Crossover

A random chromosome is chosen and then a random node is either added or removed from the graph. In the case of an addition, a random node is added from all determined neighbors. In case of removal of a node, make sure that the resultant is required to be a connected graph. Fiedler value is used here for the algebraic connectivity of a graph[30]. The Fiedler value is determined by the Laplacian matrix. For the adjacency matrix A, the degree of each node is represented by a matrix D as below.

$$D(i,j) = \begin{cases} \text{degree of node i}, & \text{if } i = j \\ 0, & \text{otherwise.} \end{cases}$$

The Laplacian matrix (L) is then calculated as, $L = D - A$. After calculating the eigenvalues of L, the smallest eigenvalue is the Fiedler value and the corresponding eigenvector is known as the Fiedler vector. Each element of the Fiedler vector corresponds to each node of the graph. It basically divides the graph into two parts: negative and positive values. Values with higher negative and positive values are far from the point of disconnect; therefore they can be deleted. We will choose a random node that is either the maximum or minimum in the Fielder vector.

F. Mutation

A random node is chosen which is absent in the offspring to be mutated. Here the aim of the mutation is to make a small graph where the missing nodes are included in the population. The same function for the generation of individual chromosomes is used here. In GA, the rate of mutation is generally low value to stop over diversified in the population and for this, 0.10 is considered the ideal value to be used.

G. Exit criteria

Usually, the exit criteria are the number of generations, rate of convergence, almost the fitness value for the population, etc. Here, the proposed method is tuned for both the number of generations and small changes in fitness.

Algorithm 1 Dense_subgraph

```
Global Data:
$size$: Number of solutions/chromosomes in the population.
$m$: Total number of nodes in the original graph.
$g$: A random selection of a chromosome for performing crossover and mutation.
$polpulation[size]$: Collection of all sub-graph in one generation.
$A$: Adjacency matrix of the original graph.
procedure GA
    POPULATION()
    while exit()==true do
        CROSSOVER()
        MUTATION()
        SURVIVOR()
    end while
end procedure

procedure CHROMOSOME
    while random(true, false) ==true do
        $n \leftarrow NEIGHBOURS(g)$
        $g[random\_index(n)] \leftarrow 1$          ▷ Neighbour merged to the sub-graph
    end while
    return $g$
end procedure
procedure POPULATION
    for $i = 1 : size$ do
        $index = \frac{i}{size} \times m$          ▷ Creates diverse population
        $population[i] \leftarrow CHROMOSOME(index)$
    end for
end procedure

procedure NEIGHBOURS(g)
    $N = A \odot ((J_{m,1} - g) \cdot g^T)$
    $n = N \cdot J_{m,1}$          ▷ $J$ is an all-ones matrix
    return $n$
end procedure

procedure CROSSOVER
    switch random($ADD, REM$) do
        case $ADD$
            $n \leftarrow NEIGHBOURS(g)$
            $g[random\_index(n)] \leftarrow 1$
        case $REM$
            $F \leftarrow fiedler\_vector(g)$
            do
                $i \leftarrow random\_index(g)$
            while $g[i] \neq max(F)$ or $g[i] \neq min(F)$
            $g[i] \leftarrow 0$
    return $v$
end procedure

procedure MUTATION
    $index \leftarrow random\_index(J - n)$
    $temp[index] \leftarrow 1$
    $g \leftarrow CHROMOSOME(temp)$
    return $g$
end procedure

procedure SURVIVOR
    $index \leftarrow find\_weak\_fitness()$
    if fitness(g)>fitness(population[index]) then
        $population[index] \leftarrow g$
    end if
end procedure
```

Definition:

- random(A): generates a random number from the set A. when (a,b) parameters are used then a random number from the range a-b is generated.
- random_index(g): returns a random index of a node from the chromosome g.
- find_weak_fitness(): returns the index of the weak chromosome from the population.
- fiedler_vector(g): returns the Fiedler vector for the sub-graph g.

IV. Result and Analysis

The proposed algorithm was tested on a graph with 100 nodes. The graph was randomly generated and by exhaustive search, it was determined that the densest sub-graph has a density of 4.735394. Our proposed algorithm was able to find a dense sub-graph which is a close approximation of the optimal solution. The algorithm was executed with different configurations to find suitable values of different parameters for the dense graph problem.

Fig. 3 shows the increase in fitness vs. generations for different population sizes, keeping the rate of mutation 0.1. Lower population sizes are less diverse, thus they are slower in finding the optimal solution and carry a risk of getting stuck at local minima. The population size 20 converges prematurely and while a population size 40 finds a good solution closer to 300 generations; it has a slower increase than the larger size of populations. Population size beyond 60 gives good results but does not significantly improve over 60. Therefore, 60 size is chosen as the optimal number of generations.

Fig. 4 shows the variation of fitness vs. generation for different mutation rates, keeping the population size 60.

Initially, zero-mutation performs well, but after more generations, with-mutation gives better results. Because zero-mutation got stuck in local minima for longer generations and almost all the graphs were larger in size. The introduction of mutation helps to get some small graphs to diversify the optimal result. When the rate of mutation is lower value, GA generally gets stuck in local minima, thus it does not converge to an optimal solution. A larger mutation rate avoids this problem, but further increases in value will not necessarily increase the final quality of the results. The mutation rate is required to be maintained at a level for avoiding local minima. The value 0.1 is therefore decided as large enough to avoid premature convergence.

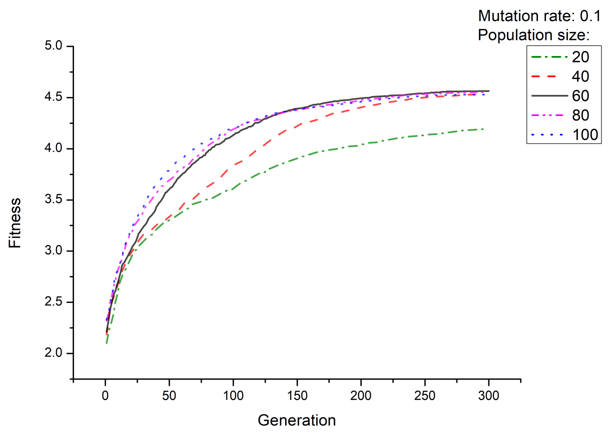

Fig. 3: Best Fitness value with each generation for different population size

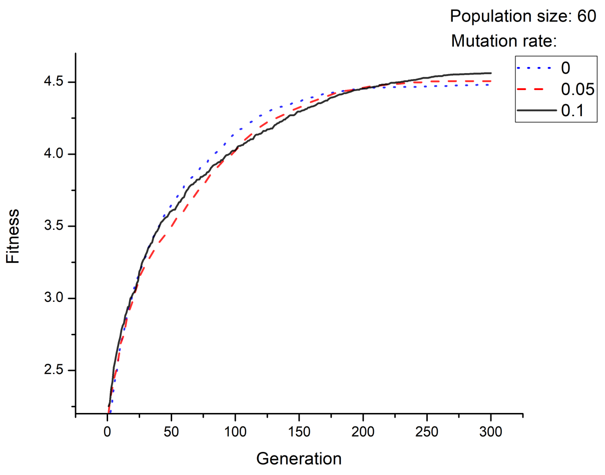

Fig. 4: Best Fitness value with each generation for different mutation rate

V. Conclusion

A GA-based optimization is proposed for finding the dense subgraph, a commonly known NP problem statement. A simple structure of the chromosome is used for implementing crossover and mutation. Linear algebraic operations are explained for finding neighbors, fitness, and connectivity of graphs. Generally, crossover and mutation are hard to propose for a graph. Here the crossover and mutation generate new chromosomes maintaining the connectivity of the graph. Fiedler value helped in maintaining the connectivity in the offspring. The proposed algorithm finds a close approximation to the dense subgraph for 100 nodes. An exhaustive search of the graph in the worst-case scenario requires $\sum_{i=1}^{100} {}^{100}C_i$ evaluations of each individuals, while the GA with population size 60 and 300 generations requires $60 \times 300 = 18,000$ evaluations of each individuals. The algorithm was experimented with the variation of mutation rate, population size, and number for generation for obtaining the fine-tuned parameter value.

References

[1] J. Chen and Y. Saad, "Dense subgraph extraction with application to community detection," *IEEE Transactions on knowledge and data engineering*, vol. 24, no. 7, pp. 1216–1230, 2010.

[2] V. Boginski, S. Butenko, and P. M. Pardalos, "Mining market data: A network approach," *Computers & Operations Research*, vol. 33, no. 11, pp. 3171–3184, 2006.

[3] E. Fratkin, B. T. Naughton, D. L. Brutlag, and S. Batzoglou, "Motifcut: regulatory motifs finding with maximum density subgraphs," *Bioinformatics*, vol. 22, no. 14, pp. e150–e157, 2006.

[4] G. D. Bader and C. W. Hogue, "An automated method for finding molecular complexes in large protein interaction networks," *BMC bioinformatics*, vol. 4, no. 1, pp. 1–27, 2003.

[5] S.-C. Chang, L.-H. Chen, L.-J. Hung, S.-S. Kao, and R. Klasing, "The hardness and approximation of the densest k-subgraph problem in parameterized metric graphs," in *2020 International Computer Symposium (ICS)*. IEEE, 2020, pp. 126–130.

[6] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan, "A fast parametric maximum flow algorithm and applications," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 30–55, 1989.

[7] M. Charikar, "Greedy approximation algorithms for finding dense components in a graph," in *Approximation Algorithms for Combinatorial Optimization: Third International Workshop, APPROX 2000 Saarbrücken, Germany, September 5–8, 2000 Proceedings*. Springer, 2003, pp. 84–95.

[8] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *Proceedings of the 31st international conference on Very large data bases*, 2005, pp. 721–732.

[9] A. Saha, X. Ke, A. Khan, and C. Long, "Most probable densest subgraphs," *ArXiv*, vol. abs/2212.08820, 2022.

[10] B. Ray, A. J. Pal, D. Bhattacharyya, and T.-h. Kim, "An efficient ga with multipoint guided mutation for graph coloring problems," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 3, no. 2, pp. 51–58, 2010.

[11] T. N. Bui and B. R. Moon, "Genetic algorithm and graph partitioning," *IEEE Transactions on computers*, vol. 45, no. 7, pp. 841–855, 1996.

[12] M. Li, H. Cui, C. Zhou, and S. Xu, "Gap: Genetic algorithm based large-scale graph partition in heterogeneous cluster," *IEEE Access*, vol. 8, pp. 144 197–144 204, 2020.

[13] Y. Lu, G. Chakraborty, and M. Matsuhara, "A two stage converging genetic algorithm for graph clustering," *International Journal of Applied Science and Engineering*, vol. 17, pp. 299–310, 2020.

[14] A. Steenbeek, E. Marchiori, and A. Eiben, "Finding balanced graph bi-partitions using a hybrid genetic algorithm," in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*. IEEE, 1998, pp. 90–95.

[15] H. Choi, J.-H. Kim, Y. Yoon, and B. R. Moon, "Investigation of incremental hybrid genetic algorithm with subgraph isomorphism problem," *Swarm Evol. Comput.*, vol. 49, pp. 75–86, 2019.

[16] H. Choi, J. Kim, and B. R. Moon, "A hybrid incremental genetic algorithm for subgraph isomorphism problem," *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 2014.

[17] K. Semertzidis, E. Pitoura, E. Terzi, and P. Tsaparas, "Finding lasting dense subgraphs," *Data Mining and Knowledge Discovery*, vol. 33, pp. 1417–1445, 2019.

[18] R. Dondi and D. Hermelin, "Computing the k densest subgraphs of a graph," *Information Processing Letters*, vol. 179, p. 106316, 2023.

[19] C. E. Tsourakakis, "The k-clique densest subgraph problem," *Proceedings of the 24th International Conference on World Wide Web*, 2015.

[20] T. Hanaka, “Computing densest k-subgraph with structural parameters,” *Journal of Combinatorial Optimization*, vol. 45, pp. 1–17, 2022.

[21] C. Ma, R. Cheng, L. V. S. Lakshmanan, and X. Han, “Finding locally densest subgraphs,” *Proceedings of the VLDB Endowment*, 2022.

[22] D. Papailiopoulos, I. Mitliagkas, A. G. Dimakis, and C. Caramanis, “Finding dense subgraphs via low-rank bilinear optimization,” in *International Conference on Machine Learning*, 2014.

[23] M. Liazi, I. Milis, F. Pascual, and V. Zissimopoulos, “The densest k-subgraph problem on clique graphs,” *Journal of combinatorial optimization*, vol. 14, pp. 465–474, 2007.

[24] T. Koana, C. Komusiewicz, and F. Sommer, “Computing dense and sparse subgraphs of weakly closed graphs,” *Algorithmica*, pp. 1–32, 2023.

[25] U. Feige, D. Peleg, and G. Kortsarz, “The dense k-subgraph problem,” *Algorithmica*, vol. 29, pp. 410–421, 2001.

[26] Y.-H. Deng, S.-Q. Gong, Y.-C. Gu, Z.-J. Zhang, H.-L. Liu, H. Su, H.-Y. Tang, J.-M. Xu, M.-H. Jia, M.-C. Chen *et al.*, “Solving graph problems using gaussian boson sampling,” *arXiv preprint arXiv:2302.00936*, 2023.

[27] S. Sempere-Llagostera, R. B. Patel, I. A. Walmsley, and W. S. Kolthammer, “Experimentally finding dense subgraphs using a time-bin encoded gaussian boson sampling device,” *Physical Review X*, 2022.

[28] J. Choi, Y. Yoon, and B.-R. Moon, “An efficient genetic algorithm for subgraph isomorphism,” in *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, 2012, pp. 361–368.

[29] M. M. Farahani and S. K. Chaharsoughi, “A genetic and iterative local search algorithm for solving subgraph isomorphism problem,” *2015 International Conference on Industrial Engineering and Operations Management (IEOM)*, pp. 1–6, 2015.

[30] M. Fiedler, “Algebraic connectivity of graphs,” *Czechoslovak mathematical journal*, vol. 23, no. 2, pp. 298–305, 1973.