ISSN 2706-655X Vol.4, Issue 4: 41-52, DOI: 10.25236/IJFET.2022.040406

Research on Combining Pattern Mining and Evolutionary Algorithm for Critical Node Detection Problems

Hongyuan Ding[1], Jiaqi Li[1,*], Man Li[2], Weichao Ding[1]

[1]Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai, China
[2]Songjiang Power Supply Company, State Grid Shanghai Municipal Electric Power Company, Shanghai, China
y30211039@mail.ecust.edu.cn
**Corresponding author*

Abstract: *The critical node detection problems (CNDPs) have important applications in network security, smart grid, epidemic control, drug design, and risk assessment. The critical node problem (CNP) is one of well-known CNDPs, which is NP-hard. In this work, we combine frequent pattern mining with evolutionary algorithm for solving CNP, where pattern mined from high-quality solutions are used to guide the construction of offspring solution. More specifically, based on the memetic algorithm (Memetic Algorithm for CNP, MACNP), frequent pattern mining was integrated into the memetic algorithm framework. The frequent patterns mined were used to construct the offspring solution, instead of crossover operator in MACNP. In the research, the nodes in frequent items set were directly fixed as part of the offspring solution. So the offspring solution inherited excellent properties from more parent solutions, thus forming the algorithm called Frequent Pattern Based Search for CNP (FPBS-CNP). This strategy improved the algorithm efficiency. Experiments were conducted on synthetic and real-world benchmark. We experimentally compare FPBS-CNP with MACNP. The experiment results show that FPBS-CNP performs well on small instances and has potential in solving large instances.*

Keywords: *Critical node detection problem; Frequent pattern mining; Evolutionary algorithm; Memetic algorithm*

1. Introduction

Complex network is a simplification of the complex system in the real world. The study of complex network helps to strengthen people's cognition of the real world. With the rise of complex network research, the critical node research which plays an important role in complex network research has been paid more and more attention. The research on critical nodes in complex networks has many practical significances. For example, the analysis of critical nodes in the Internet can improve the network performance and enhance the network's destruction resistance, and some networks can be destroyed by deliberately attacking critical nodes in complex networks. In addition, it has a broad application prospect in drug development, commodity sales, military information confrontation, virus prevention and treatment, case detection and so on.

At present, many networks have been proved to be complex networks, such as social network, railway network and protein interaction network. In these networks, most nodes have only a small number of connections with other nodes, but some nodes have a large number of connections. These nodes with a large number of connections play a key role in the whole network, which is called "critical nodes". The critical node problem in complex networks is mainly aimed at removing a subset of nodes from the network so that the remaining networks have minimal pin-wise connectivity. In recent years, the research in the field of machine learning has developed rapidly. Machine learning can analyze a large number of data samples and extract features to build models. Machine learning is introduced into the critical node research of complex networks, and efficient heuristic optimization algorithms are designed and paid more and more attention.

ISSN 2706-655X Vol.4, Issue 4: 41-52, DOI: 10.25236/IJFET.2022.040406

Complex network problems are mainly solved by the traditional precise algorithm and heuristic algorithm. Among them, the precise algorithm often seeks the optimal solution of the problem through exhaustive or branch-bound algorithm based on integer programming, and the algorithm performance is not good enough for large-scale problems. Heuristic algorithms can usually find high quality suboptimal instances of large-scale examples and obtain satisfactory solutions with acceptable computational costs, but they cannot get quality estimates of the solutions. At present, the study of designing learning optimization algorithm based on conventional optimization algorithm and introducing machine learning method is still scarce, especially the study of learning evolutionary algorithm has not been reported in the existing literature. Therefore, based on the evolutionary algorithm, this paper introduces frequent pattern mining technology to find common patterns among high-quality solutions, and uses the mined patterns to guide the construction of the descendants of the evolutionary algorithm to solve the critical node problems. Compared with constructive heuristic algorithm and local search heuristic algorithm, population-based evolutionary algorithm usually has stronger global search ability and is an effective method to solve CNDP.

2. Preliminary Knowledge

2.1. Identification of Critical Nodes

In a complex network, not all nodes are equally important. A small number of nodes are more important than others. They play important roles in the complex network, especially those nodes whose importance is related to network connectivity, and these nodes are usually called "critical nodes". The critical node detection problem is an optimization problem, which is to find the minimum subset of nodes according to some predefined connection metrics, so that the deletion of these nodes will significantly reduce the network connectivity. There are two common methods of critical node detection: The first is the basic critical node detection problem, which minimizes the total number of connected nodes in the graph given the maximum number of nodes that can be removed. The second is the critical node detection problem constrained by cardinality, that is, given the maximum allowable size of connected graph components, the number of nodes to be removed is minimized to achieve a certain measure of graph connectivity minimum.

In literature [1], critical nodes are defined as a group of nodes whose deletion from the network will lead to the maximum disconnection of the remaining graph, which is the beginning of the study on critical nodes. Previous centrality measurements are not suitable for this problem, and new heuristic methods need to be designed [2]. Literature [3] provides a formal definition of the critical node problem and a proof that it is a nondeterministic polynomial complete problem (NP-complete), which is an NP problem whose complexity is related to the complexity of all problems of its kind.

Given an undirected graph $G(V,E)$, where $V=\{1,\dots,n\}$ is the set of n nodes and $E \subseteq V \times V$ is the edge set in G. This article chooses the critical node identification is given an integer K, determine a subset $S \subseteq V$ from the undirected graph G, including $|S| \leq K$, made after the delete S, to minimize the residual graph connectivity metric, namely each connected component in the remaining graph nodes logarithmic minimum sum, The calculation formula of the measurement value is shown in (1).

$$f(S)=\sum_{i=1}^{T}\binom{|C_i|}{2} \tag{1}$$

Where $f(S)$ is the objective function, T is the total number of connected components C_i in the remaining graph, and K is the maximum number of nodes that can be removed from the graph. As shown in Figure 1, after removing nodes V_2 and V_3 in Figure (a), Figure (b) is obtained, which consists of sets $\{V_1\}$ and $\{V_4, V_5\}$. The connectedness of Figure (a) is shown in Figure (2), and that of Figure (b) is shown in Figure (3), and $f(S_b)$ is the minimum connectedness. Therefore, deleted nodes V_2 and V_3 are critical nodes.

$$f(S_a)=\binom{5}{2}=3 \tag{2}$$

$$f(S_b)=0+\binom{2}{2}=1 \tag{3}$$

ISSN 2706-655X Vol.4, Issue 4: 41-52, DOI: 10.25236/IJFET.2022.040406

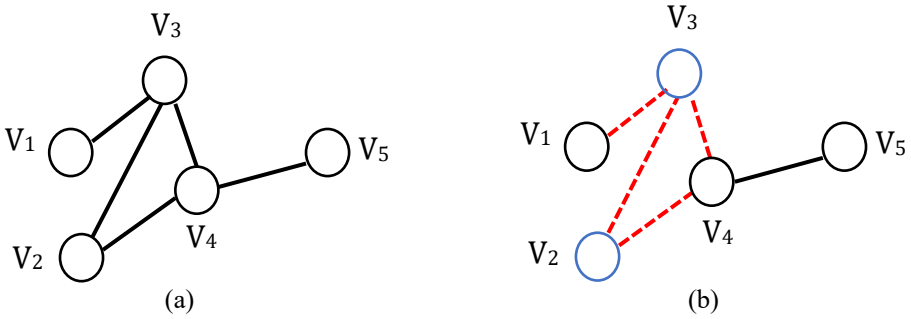

Figure 1: CNP legend

In a complex network with a large number of nodes, such as network $P = \{S_1, S_2, \dots, S_p\}$, a minimum value S^* is initialized, the connectivity measure $f(S_i)$ of S_i after nodes are removed is continuously calculated, and S^* is replaced with the minimum value until all possible combinations are traversed. In addition, the more critical nodes the solution contains, the smaller the value of the objective function will be. However, combined with the actual situation, the number of critical nodes should not exceed a certain value, otherwise it will consume a lot of resources. Therefore, the size of S is generally set as the given value of K.

2.2. Related Algorithm

Many algorithms have been proposed to solve the critical node problem. These algorithms fall into two main categories: precise algorithms and heuristic algorithms. Based on the conventional optimization algorithm, the introduction of machine learning method, the design of learning optimization algorithm is still scarce.

Aiming at finding the optimal solution of a problem, exact algorithms usually seek the optimal solution of a problem through exhaustive or branch-bound algorithm based on integer programming. For the critical node problem, Arulselvan et al. [3] proposed the first integer linear programming model for CNP, which has $O(n^2)$ variables and $O(n^3)$ constraints, and can only accurately solve small-scale networks with no more than 150 nodes. Subsequently, Summa et al. [4] proposed two improved models. The first model is an extended and enhanced version of the model in reference [3]. It has non-polynomial constraints, adopts branch and Cut architecture, and its relaxation can be completed in polynomial time. The second model is based on the quadratic programming reconstruction of the CNP problem. Due to the large number of constraints, with at least $O(n^3)$ constraints, these models are only suitable for some small sparse networks and can solve the network with several hundred nodes at most. To solve this problem, Veremyev et al. [5] proposed an improved compact linear programming model with $O(n^2)$ constraints, which can solve a large network with 1200 nodes quickly and accurately.

Heuristic algorithms can usually find high quality suboptimal instances of large-scale examples. They can get satisfactory solutions with acceptable computational costs, but cannot get quality estimates of solutions. Compared with exact algorithm and approximation algorithm, heuristic algorithm can provide a high-quality suboptimal solution quickly, although it lacks effective theoretical support. The heuristic algorithm based on population has achieved great success in solving the problem of critical node identification. For the CNP problem, Purevsuren et al. [6] combined greedy randomized adaptive search procedure (GRASP) with exterior path-relinking (EPR) proposed an effective mixed meta-heuristic method, wince-EPR. This algorithm first generates some high-quality solutions by MEANS of GRASP, and then uses EPR to establish a path connection between two high quality solutions to find higher quality solutions. Aringhieri et al. [7] proposed an effective general evolutionary algorithm framework, which can effectively solve the critical-node problem and its multiple variants. Wu Zhikang [8] proposed a greedy frame for critical node Problem (GCNP) based on node centrality, using the centrality index to select the initial point coverage set, and then using greedy rules to select critical nodes. The test and analysis under different centrality indexes show that each index under GCNP can accurately identify the critical nodes. Cao Jiuxin et al. [9] proposed a mixed heuristic and greedy strategies based algorithm (MHG), The algorithm achieves a good balance in running time and influence range, and can handle large-scale networks with scalability.

In 2019, Zhou et al proposed using evolutionary Algorithm to solve CNP[10] (Memetic Algorithm for the Classic Critical Node Problem, MACNP), The algorithm combines crossover operators (inheriting superior properties of parent solutions to produce high-quality descendant solutions),

ISSN 2706-655X Vol.4, Issue 4: 41-52, DOI: 10.25236/IJFET.2022.040406

component-based local search, and rank-based population updating processes. In 2020, Zhou et al. proposed Variable Population Memetic Search (VPMS), which uses the strategic Population size mechanism to dynamically adjust the Population size in the Search process, starting from a small Population with only two solutions. Then adjust the population size according to the search state [11]. The algorithm is applied to solve CNP, and the results show that the algorithm has good performance for CNP.

2.3. Frequent Pattern Mining Techniques

Mining Frequent Patterns (FP) is the critical to mining association rules, and a basic problem of mining association rules is mining Frequent item-sets. The FP-growth(frequent pattern growth) algorithm proposed by Professor Han Jiawei [12] et al is a classic algorithm in the field of frequent pattern mining. Its efficient performance comes from its powerful information compression tree -- frequent Pattern tree (FP-tree), which enables overlapping item-sets to share the prefix of corresponding branches by building the tree.

FP-tree displays all the relevant frequency information in the database. Each branch represents a frequent item set. Nodes along the branch are stored in descending order of frequency, and leaf nodes represent the least frequent items. FP-tree also has an associated header table, in which individual entries and their counts are stored in decreasing order of frequency. Entries for items also contain the heads of a linked list of all corresponding nodes of the FP-Tree. To build the FP-Tree, two database scans are required while mining all frequent item-sets. The first time an initial scan of the database computes the frequency of each item to find all frequent items. These items are then inserted into the item header table in decreasing order. In the second scan, when each transaction is scanned, its frequent item-sets are inserted into the FP-tree as branches. If an item set is prefixed with the item set in the tree, the new item set will collectively represent the prefix of the branch of the item set. In addition, a counter is set up associated with each node in the tree [14]. The counter stores the number of transactions containing the itemset represented by the path from the root node to the related node. This counter is updated if a transaction causes a new branch to be inserted. After two scans, the constructed FP-tree contains all frequency information of the database, and the mining database can be transformed into mining FP-tree.

The FP-growth algorithm also relies on the following principle: if X and Y are two item sets, then the count of item set $X \cup Y$ in the database is the number of Y in the database containing X. This restriction is called the conditional schema base of X. The FP-tree constructed from the conditional schema base is called the conditional FP-tree of X and is represented by T_X. In a transactional database, it is easy to generate association rules if you know the support of all frequent item-sets. When a transactional database contains a large number of large frequent item-sets, mining all frequent item-sets can be extremely time-consuming.

Reference [13] proposed FPmax, a variant of the FP-growth method, for mining maximum frequent item-sets. Since array technology accelerates the FP-growth of sparse data sets, it is also used in FPmax algorithm and improved FPmax* algorithm. FPmax* uses not only array technology, but also a new subset testing algorithm. Compared to FPmax, the improved method FPmax* has more efficient subset testing, among other optimizations. FPclose algorithm is also proposed to mine closed-circuit frequent item-sets. In this algorithm, the CFI-tree (another variant of FP-Tree) is used to test the closure of frequent item-sets.

3. Hybrid Pattern Mining and Critical Node Problem Solving for Evolutionary Algorithms

In this paper, the algorithm for solving the critical node problem is a mixture of pattern mining and evolutionary algorithm. After the initial population is established, frequent item-sets are mined by FPmax* algorithm, and subsequent solutions are constructed based on frequent item sets. Finally, the population is updated and iterated repeatedly until the best value is found or the time reaches the upper limit, and the call of the algorithm is ended. For details, see Figure 2.

ISSN 2706-655X Vol.4, Issue 4: 41-52, DOI: 10.25236/IJFET.2022.040406

Algorithm 1 FPBS-CNP

Input: An undirected graph G = (V,E), an integer K
Output: The best solution S^* found so far

$P = \{S_1, S_2, ..., S_p\} \leftarrow$ BuildPopulation();
$S^* \leftarrow \arg\min\{f(S_i) : i = 1, 2, ..., p\}$;
while $T < T_{Limit}$ **do**
  if $NbrIdleNoUpdates > MaxIdleOfNoUpdate \bigvee$ first iteration **then**
    $NbrIdleNoUpdates \leftarrow 0$;
    $Patterns \leftarrow$ MineFrequentPatterns(P, $PatternUp$, θ);
  end if
  $S' \leftarrow$ SolutionConstructDrivenByPatterns($Patterns$, $PatternThreshold$, $PoolSize$);
  $S' \leftarrow$ ComponentBasedNeighborhoodSearch(S');
  if $f(S') < f(S^*)$ **then**
    $S^* \leftarrow S'$;
  end if
  if UpdatePopolation(P, S') == TRUE **then**
    $NbrIdleNoUpdates \leftarrow 0$;
  else
    $NbrIdleNoUpdates \leftarrow NbrIdleNoUpdates + 1$;
  end if
end while

Figure 2: FPBS-CNP Algorithm.

3.1. Frequent Itemset Mining

In reference [14], the FPmax* algorithm for mining maximum frequent item sets is proposed, which accelerates the FP-growth algorithm applied on sparse data sets, and has more effective subset testing and other performance optimizations.

The maximum frequent item set mining algorithm in this paper is based on FPmax*. In FPmax*, a global data structure, the maximum Frequent itemset tree (MFI-tree), is used. Each conditional FP-tree T_X has an MFI-tree M_X that contains all the largest itemsets in the conditional pattern base of X.

In MFI-tree, each node in the subtree has three fields: project name, level, and link node, and all nodes with the same project name are linked together. MFI-tree also has a header table, which is built based on the order of items in the associated FP-Tree table. Each entry in the header table consists of an item name and the head of a list of links. The head points to the first node in the MFI-tree with the same item name. The pseudo-code of FPmax* algorithm is as follows:

Algorithm 2 FPmax*

Input: An FP-tree T, the MFI-tree M for $T.base$
Output: updated M

if T only contains s single path P **then**
  $M \leftarrow M \bigcup P$;
else
  for each i in $T.header$ **do**
    $Y \leftarrow T.base \bigcup \{i\}$;
    if $T.array$ is not NULL **then**
      $Tail \leftarrow$ {frequent items for i in $T.array$};
    else
      $Tail \leftarrow$ {frequent items in i's conditional pattern base};
    end if
    sort $Tail$ in decreasing order of the items counts;
    if $Y \bigcup Tail$ is not a subset of M **then**
      construct Y's conditional FP-tree T_Y and its array A_Y;
      initialize Y's conditional MFI-tree M_Y;
      call FPmax*(T_Y, M_Y);
      $M \leftarrow M \bigcup M_Y$;
    end if
  end for
end if

Figure 3: FPmax Algorithm.*

Firstly, an empty MFI-tree is constructed from the FP-tree of the original database. The collection containing $T.base$ and items in the current FP-tree is not a subset of any existing MFI before recursive

ISSN 2706-655X Vol.4, Issue 4: 41-52, DOI: 10.25236/IJFET.2022.040406

calls. During the recursive call, if there is only one path in T, then the path and T. case together are a new MFI, and the MFI is inserted into M; If T contains more than one path, recursively call FPmax*(T_Y, M_Y) for each item I in the head table, where $Y = T.base \cup \{i\}$. The items in the T-head table are arranged in increasing frequency order. Line 10 calls the subset checking function (that is, superset pruning) to check whether all frequent items in Y and $Y.base$ are subsets of the existing MFI in M. FPmax*(T_Y, M_Y) is recursively called if the subset checking function returns false. If $Y \cup Tail$ is not a subset of any MFI, there is no way to determine whether $Y \cup Tail$ is frequent. By constructing the condition FP-tree T_Y of Y and finding that T_Y has only one path, we can conclude that $Y \cup Tail$ is frequent. Since $Y \cup Tail$ is not a subset of any existing MFI, it is inserted into M_Y as a new MFI [14], as show in Figure 3.

The first FP-tree must fit into main memory before FPmax* is called for the first time. When you scan the database for the first time, delete items with frequencies below the minimum support level. In evolutionary algorithms, local search is needed during population initialization to optimize the initial solution, that is, to obtain a high-quality parent solution set. As shown in Figure 4, the hypothesis 1 contains the node as parent solution $\{a, b, c, d, f, h, j, l, n, p\}$, parent solution 2 contains nodes $\{a, c, d, e, f, i, k, q, r, s\}$, The parent solution 3 contains nodes $\{a, c, d, f, g, h, l, m, n, o\}$. A to z indicate nodes. These high-quality solutions often contain a large number of common nodes. After the first scan removes the nodes whose frequency is lower than the minimum support, there is a high possibility of inclusion relationship between the solutions. In the most extreme case, as shown in Figure 5, the FP-tree built after the initial scan contains only one path, at which point the core of FPmax* cannot proceed.

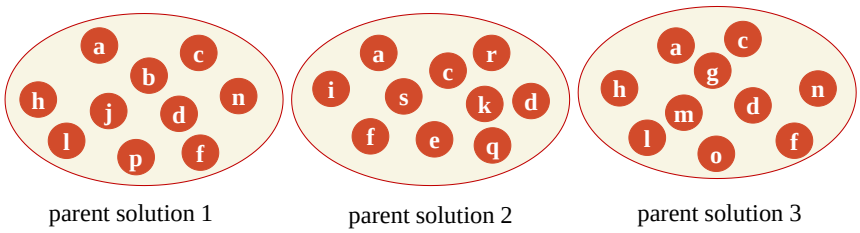

Figure 4: High similarity population.

If the population initialization process is too random, it is inevitable that the parent solution contains a large number of the same elements, and the reference basis of the offspring solution is not high enough to generate high-quality offspring solutions, which will affect the subsequent process of population evolution. If the "Single Path" problem is ignored, in FPmax* algorithm, if a single-path FP-tree is generated after the first scan of the database, the maximum frequent item set on the Path is directly output, which contains a large number of the same elements in the parent solution. The "Single Path" problem was found to be extremely rare in multiple tests, so ignoring it had little impact on the results.

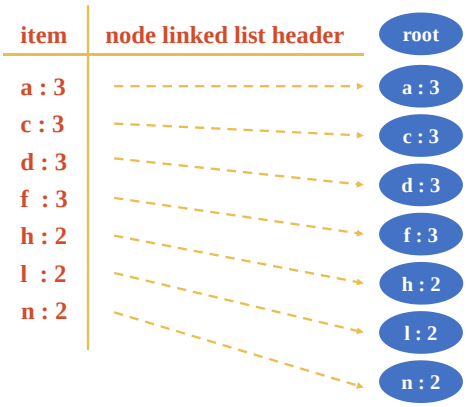

Figure 5: "Single Path" problem.

The main idea of frequent patterns based search for CNP (FPBS-CNP) proposed in this paper is to construct descendant solutions based on maximum frequent patterns to replace the dual parent crossover operators in evolutionary algorithms. So that the quality of the descendant solution constructed is higher. After years of research, FPmax* algorithm has formed a more mature code

ISSN 2706-655X Vol.4, Issue 4: 41-52, DOI: 10.25236/IJFET.2022.040406

version, so this paper directly called the executable file of FPmax* algorithm, using TXT files to store the required database and mined patterns, through the file reading and writing to complete the mining of the population, The pseudo-codes of frequent pattern mining in FPBS-CNP algorithm are as follows:

Algorithm 3 MineFrequentPatterns
Input: P, maximum number of item sets:$PatternUp$, the probability of accepting an item set:θ
Output: Best frequent item sets vector $Patterns$

output $P = \{S_1, S_2, ..., S_p\}$ to elite.txt;
call FPmax*.exe to mine frequent item sets from elite.txt, and save results in patterns.txt;
$NbrPatterns \leftarrow 0$;
$Patterns \leftarrow \emptyset$;
while $p \leftarrow$ readline from patterns.txt **do**
  if $NbrPatterns >= PatternUp$ **then**
    find the smallest set p_{min} in $Patterns$, the size of which is $size_{min}$;
    if $p.size > size_{min} \bigvee (p.size == size_{min} \bigwedge$ random probability $r < \theta)$ **then**
      $p_{min} \leftarrow p$;
    end if
  else
    $Patterns \leftarrow Patterns \bigcup p$;
  end if
end while

Figure 6: MineFrequentPatterns Algorithm.

Where $PatternUp$ refers to the largest set of accepted items, and the value is 40. θ refers to the likelihood of accepting a set of items, with a value of 0.5. First, the population $P = \{S_1, S_2, \dots, S_p\}$ is established. If the first iteration or the number of non-improved iterations of the population reaches the set value $PatternUp$, the solution of the population is written to elite.txt. FPmax*.exe is called to mine frequent item sets and construct descendant solutions based on current frequent item sets. If the new solution is better than the worst solution of the population, it will be replaced. The above process is iterated repeatedly until the best value is found or the upper limit of time is reached. The specific flow chart of FPBS-CNP algorithm is shown in Figure 4.

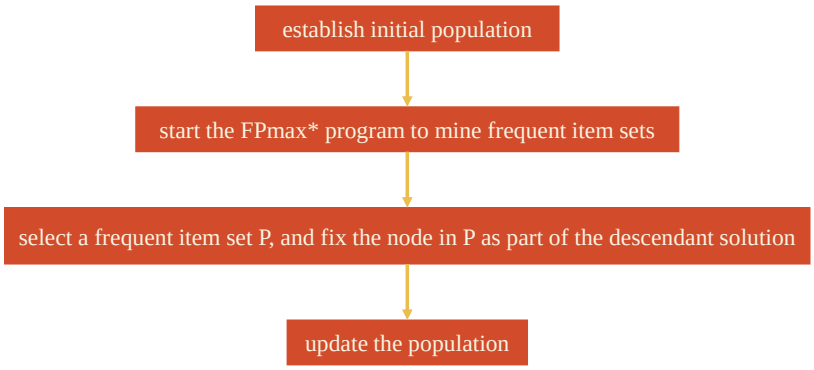

Figure 7: Flow chart of FPBS-CNP algorithm.

First, the executable file of FPmax* algorithm is directly called in the program in the way of ostringstream output stream, and the elite set in the file of the storage population is mined as a database, and output to the file of the specified storage mode. Finally, the main program reads the file of the storage mode to construct the descendant solution.

Where, the population size is set to the common scale of 20. The results show that the number of patterns mined in this way can be more or less, and reading all the patterns in can take up a lot of space, while only one pattern is needed to construct the descendant solution. So we carried on the further improvement and advance in the application set an upper limit, if the quantity reach the ceiling, while reading patterns are the next every time I read from an existing pattern in the vector length of the shortest one is chosen, and read in comparison, a new model of length at this time if the new model is longer, the alternative model is chosen, if equal length, with a certain probability, Otherwise, discard the new mode and proceed to the next round of reading until all modes have been read.

ISSN 2706-655X Vol.4, Issue 4: 41-52, DOI: 10.25236/IJFET.2022.040406

3.2. Descendant Solution Construction Based on Frequent Itemsets

Genetic algorithm (GA) is a computational model that imitates the natural selection and genetic mechanism of Darwin's biological evolution. It follows the principles of survival of the fittest and survival of the fittest and conducts random search by simulating the process of natural evolution. The selection operation simulates survival of the fittest. It will select operators on the population, put the optimized individuals into the matching pool, and generate new individuals through pairing crossover and then inherit to the next generation. The individuals with higher fitness will have more possibility to be inherited to the next generation. In this paper, tournament selection mechanism is selected to realize the selection of mined patterns to construct descendant solutions.

The tournament selection mechanism is to take a certain number of individuals from the population at a time (put them back into the sample), and then select the best one of them into the offspring population. Repeat this operation until the new population size reaches the original population size. An n-variate tournament is when you take N individuals out of the population at once

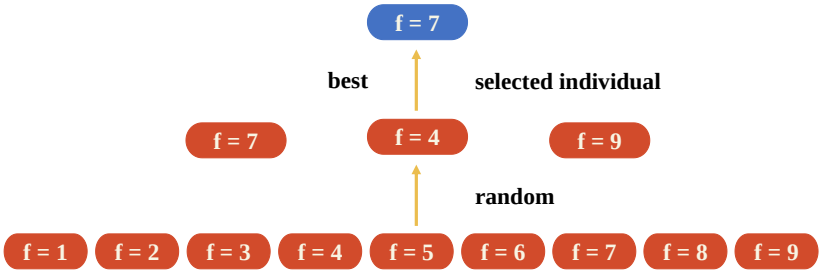

Figure 8: Tournament selection Mechanism.

In our experiment, it is assumed that the frequent item set P obtained in the previous step includes n item sets. Since we mined frequent item sets through FPmax* algorithm, n must not exceed K (that is, the size of the mined parent solution). Define the size of the tournament pool as m (set as 4), randomly select m($1 \leq m \leq n$) frequent itemsets from the frequent itemsets P (each item set has the same probability of being selected), and then select the one with the best fitness value (i.e., the largest size). If there are multiple frequent item sets with maximum size, one of them is randomly selected to enter the offspring population. Repeat the steps above (repeat times for population size), supplementing offspring solutions until the new population size reaches the original population size. The pseudocode is shown below, where $PatternThreshold$ refers to the threshold number of frequent itemsets for constructing child solutions, set to 15, and $PoolSize$ refers to the size of the tournament pool, set to 4.

Algorithm 4 SolutionConstructDrivenByPatterns

Input: Frequent item sets $Patterns$, threshold num of frequent item sets: $PatternThreshold$, the size of tournament pool: $PoolSize$

Output: An offspring solution S'

```
if Patterns.size > PatternThreshold then
    Pool ← ∅;
    for i = 0; i < PoolSize; i++ do
        Pool ← Pool ⋃ a random item set from Patterns;
    end for
    p_selected ← biggest item set in Pool;
else p_selected ← randomly select an item set from Patterns;
end if
S' ← ∅;
S' ← S' ⋃ p_selected;
while S'.size < K do
    S' ← S' ⋃ a random node in a large component;
end while
```

Figure 9: Solution construct driven by patterns.

Above the tournament selection mechanism of time complexity is $O(|P|)$. Its advantage is that selection pressure can be easily adjusted by changing the size of tournament pool m [15], that is, it can be used in both maximization and minimization problems. Among them, the larger the tournament pool, the smaller the chance of selecting smaller frequent item sets.

ISSN 2706-655X Vol.4, Issue 4: 41-52, DOI: 10.25236/IJFET.2022.040406

Generally speaking, the method of evolutionary algorithm to supplement the descendant solution is to randomly select the nodes in the large component, which greatly reduces the search space, but some nodes in the large component do not improve the value of the objective function, and may miss the nodes with good performance.

4. Experimental Results and Analysis

The instances used in the test studies are from two commonly used baseline sets.

(1) The comprehensive reference set (Model) contains 10 examples (235-5000 nodes), which are divided into four categories: Barabasi–Albert (BA) figure，Erdos–Renyi (ER) figure，Forest-Fire (FF) figure.

Table 1: The parameters setting.

Parameter	Instructions	Final value
PatternUP	Upper limit of accepted frequent itemsets	30
θ	The probability of accepting a frequent item set	0.5
PatternThreshold	The number of frequent itemsets to refer to when constructing descendant solutions	15
PoolSize	Tournament Pool size	4

(2) The Realworld benchmark set (Realworld) consists of 20 Realworld diagrams (121-23133 nodes) from a variety of practical applications in the fields of biology, electronics, transportation, and complex networks.

The experiment was tested on the server, and the language environment was C++. Each instance was set to run 30 times, and the upper limit of each running time was 3600 seconds.

4.1. Parameter Settings

Table 1 shows the parameters involved in the FPBS-CNP algorithm. The sizes of PatternUP and PoolSize refer to the Settings [15]. The value for PatternThreshold is set to half the value for PatternUP, taking into account the number of frequent item sets of reference when constructing descendant solutions. θ only works if the size of the read frequent itemset is equal to the existing minimum frequent itemset, and the two can be considered to be the same, so the probability value is 0.5.

4.2. Comparison of FPBS-CNP and MACNP Results

Table 2 shows the experimental results of the comparison between the FPBS-CNP algorithm and MACNP algorithm formed by replacing the crossover operator in MACNP with the descendant solution constructed based on frequent item sets. The experimental data of MACNP in Table 2 are from literature [5]. In order to facilitate observation, the results of 5 representative instances (EU_flights, condmat, facebook, OClinks, Hamilton2000) are selected and made into a histogram, as shown in Figure 7. It can be seen from the figure that FPBS-CNP has a certain potential in large-scale calculation.

4.3. Cumulative Probability Distribution

The time-to-target (TTT) graph [16] can show the probability of the algorithm finding a solution at least as good as the given target value within a given running Time. In this experiment, in order to ensure the fairness of comparison, the stop conditions of both FPBS-CNP and MACNP were set as 100 generations, and they were run for 30 times respectively. In Figure 8, (a) and (b) are TTT graphs on a given instance (ER235, humanDi) respectively. It can be seen that the FPBS-CNP algorithm achieves the given target value significantly faster than the MACNP algorithm.

ISSN 2706-655X Vol.4, Issue 4: 41-52, DOI: 10.25236/IJFET.2022.040406

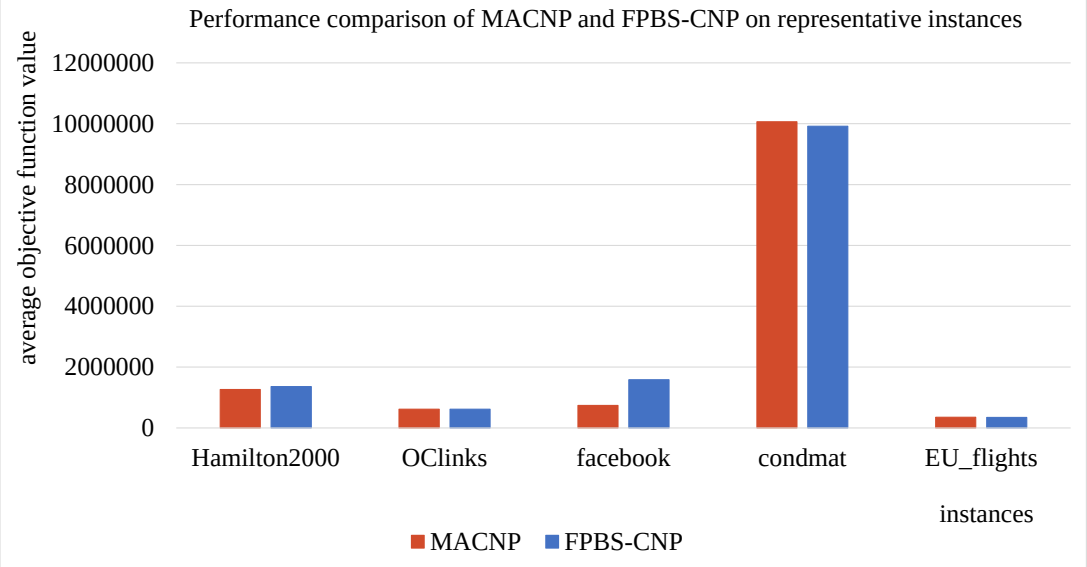

Figure 10: Comparison of FPBS-CNP and MACNP results.

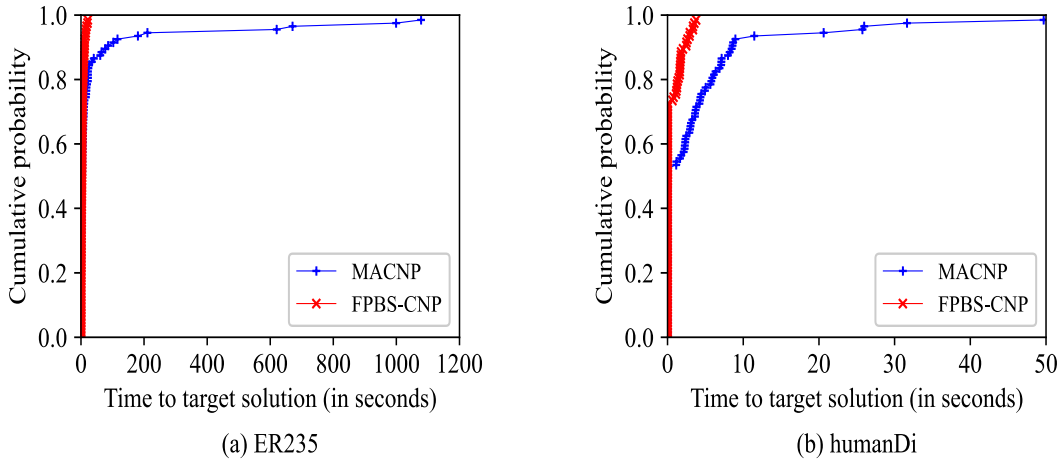

Figure 11: TTT plots of measured data of FPBS-CNP and MACNP.

5. Conclusion

Constructing progenitor solutions based on pattern mining is to apply the knowledge of machine learning to heuristic algorithms, and use the mined frequent item sets to guide the construction of progenitor solutions, so as to inherit the excellent properties of the parent solutions to the greatest extent. In this paper, the executable exe file of FPmax* algorithm is directly called to complete the mining and reading of frequent item sets by reading and writing the file. This process can be promoted, because each call process involves many times to read or write the relevant txt file, will cause time consumption, if you can directly transfer parameters to replace this process, will save part of the time for the main process of the algorithm, which can be tried in the future work.

After the nodes in the frequent item set are added to the descendant solution, the nodes in the large component are selected first for the complement of the remaining solution, so as to improve the quality of the descendant solution and save the searching time. However, the process of pattern mining in this paper is time-consuming, especially not suitable for large-scale instances. Future work can focus on improving the efficiency of pattern mining. If a fast and efficient method can be found, the algorithm will be greatly improved.

ISSN 2706-655X Vol.4, Issue 4: 41-52, DOI: 10.25236/IJFET.2022.040406

Table 2: The performance comparison of FPBS-CNP and MACNP.

	MACNP			FPBS-CNP		
Instance	f_{best}	f_{avg}	t_{avg}	f_{best}	f_{avg}	t_{avg}
BA500	**195**	**195.0**	0.0	**195**	**195.0**	0.0
BA1000	**558**	**558.0**	0.3	**558**	**558.0**	1.8
BA2500	**3704**	**3704.0**	0.7	**3704**	**3704.0**	0.8
BA5000	**10196**	**10196.0**	6.5	**10196**	**10196.0**	3.9
ER235	**295**	**295.0**	7.1	**295**	**295.0**	1.3
ER466	**1524**	**1524.0**	28.5	**1524**	**1524.0**	22.1
FF250	**194**	**194.0**	0.0	**194**	**194.0**	0.0
FF500	**257**	**257.0**	0.4	**257**	**257.0**	126.5
FF1000	**1260**	**1260.0**	84.9	**1260**	**1260.0**	9.0
FF2000	**4545**	**4545.0**	107.6	**4545**	**4545.0**	22.2
Bovine	**268**	**268.0**	0.0	**268**	**268.0**	0.0
Ecoli	**806**	**806.0**	0.0	**806**	**806.0**	0.0
Circuit	**2099**	**2099.0**	0.2	**2099**	**2099.0**	0.8
humanDi	**1115**	**1115.0**	0.6	**1115**	**1115.0**	0.3
TreniR	**918**	**918.0**	0.3	**918**	**918.0**	1.2
yeast1	**1412**	**1412.0**	21.7	**1412**	**1412.0**	16.4
H2000	**1243859**	**1263495.6**	2861.9	1351312	1361106.7	1551.4
H3000a	**2844393**	**2884781.7**	3280.7	3082504	3103419.2	2039.1
H3000b	**2841270**	**2885087.0**	3252.9	3060982	3097384.9	2208.3
H3000c	**2838429**	**2869348.5**	3307.5	3065881	3092126.9	1825.0
H3000d	**2831311**	**2892562.7**	3250.9	3059184	3099530.1	1490.6
H3000e	**2847909**	**2887525.7**	1906.1	3078532	3103887.1	1906.1
H4000	**5152977**	**5267375.5**	2907	5547466	5575804.7	1898.8
H5000	**7972525**	**8094812.6**	3226.6	8678861	8776659.2	1970.8
openfli	**26842**	**28704.3**	2093.7	28798	28835.2	1589.6
OClinks	**612303**	**614544.0**	584.6	614467	614504.0	195.1
facebook	**643162**	**739436.6**	2978.5	1517022	1584413.5	1734.2
astroph	62068926	62547898.1	1911.4	**61593761**	**62530611.1**	2011.5
condmat	9454361	10061807.8	1779.5	**8969669**	**9919703.9**	2036.2
EU_fli	**348268**	351657.0	232.6	348269	**349765.5**	1387.1

Acknowledgements

This work was supported by the Shanghai Municipal College Student' Innovation and Entrepreneurship Training Program under grant No. S20068.

References

[1] S. Boccaletti, V. Latora, Y. Moreno, et al. Complex networks: Structure and dynamics. Physics Reports, 2006, 424:175-308.
[2] S. P. Borgatti. Identifying sets of key players in a social network. Computational and Mathematical Organization Theory, 2006, 12:21-34.
[3] Arulselvan A, Commander C W, Elefteriadou L, et al. Detecting critical nodes in sparse graphs. Computer & Operations Research, 2009, 36(7):2193-2200.
[4] Summa M D, Grosso A, Locatelli M. Branch and cut algorithm for detecting critical nodes in undirected graphs. Computational Optimization and Applications, 2012, 53(3):649-680.
[5] Veremyev A, Boginski V, Pasiliao E L. Exact identification of critical nodes in sparse networks via new compact formulations. Optimization Letter, 2014, 8(4):1245-1259.
[6] Purevsuren D, Cui G, Qu M, et al. Hybridization of GRASP with exterior path relinking for identifying critical nodes in graphs. IAENG International Journal of Computer Science, 2017, 44(2):1-9.
[7] Aringhieri R, Grosso A, Hosteins P, et al. A general evolutionary framework for different classes of critical node problems.Engineering Applications of Artificial Intelligence, 2016, 55:128-145.
[8] Wu Z, Research on the Critical Node Detection Problem based on the Local Characteristic of

ISSN 2706-655X Vol.4, Issue 4: 41-52, DOI: 10.25236/IJFET.2022.040406

Network[D]. Shanxi University, 2019.
[9] Cao J, Min H, Liu S, et al. Mixed heuristic and greedy strategies based algorithm for influence maximization in social networks. Journal of Southeast University(Natural Science Edition) , 2016, 46(05):950-956.
[10] Zhou Y, Hao J, Glover F. Memetic search for identifying critical nodes in sparse graphs. IEEE Transactions on Cybernetics, 2019, 49(10): 3699-3712.
[11] Zhou Y, Hao J, Fu Z, et al. Variable population memetic search: A case study on the critical node problem. IEEE Transactions on Evolutionary Computation, 2021, 25(1):187-200.
[12] Han J, Jian P, Yin Y, et al. Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Mining & Knowledge Discovery, 2004, 8(1):53-87.
[13] Grahne G, Zhu J. High performance mining of maximal frequent itemsets[A]. In: Proceedings of the 3rd IEEE International Conference on Data Mining[C]. Melbourne, 2003.
[14] Grahne G, Zhu J. Efficiently using prefix-trees in mining frequent itemsets[A]. In: Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations[C]. Frequent Itemset Mining Implementations. Melbourne, 2003.
[15] Zhou Y, Hao J, Duval B. Frequent pattern-based search: A case study on the quadratic assignment problem. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2020, 1-13.
[16] Aiex R M, Resende M, Ribeiro C C. TTT plots: A perl program to create time-to-target plots. Optimization Letters, 2007, 1(4):355-366.