An Evolutionary Computational Method for *N*-Connection Subgraph Discovery

Enhong Chen Xujia Chen
Department of CS
University of Science & Technology of China, Hefei
Anhui, P.R.China
cheneh@ustc.edu.cn
xjchen@mail.ustc.edu.cn

Phillip C-Y Sheu
State Key Lab of Software Engineering, China (Wuhan University) and
Department of EECS
University of California
Irvine, CA 92697
psheu@uci.edu

Tieyun Qian
Department of Computer Science, Huazhong University of Science and Technology,
Wuhan, Hubei, P.R.China
qty@mail.hust.edu.cn

Abstract

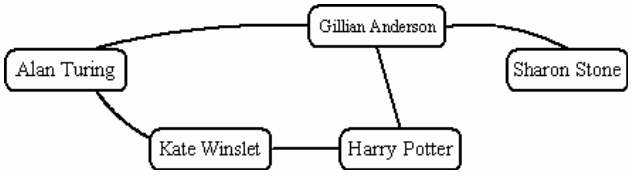
The Problem of n-connection subgraph discovery (*n-CSDP for short*) *is to find a small sized subgraph that can well capture the relationship among the n given nodes in a large graph. However there have been very few researches directly addressing the CSDP problem. Furthermore the currently available methods, for example, the electricity analogues based algorithm can only be suitable for tackling the 2-Keynodes CSDP and does not work anymore when n is greater than two. To deal with this problem, we propose an effective approach to discover the subgraph in two stages. In the first stage we propose a neighbor-growth based method to extract a relatively bigger candidate subgraph compared with that of result subgraph. In the second stage an evolutionary algorithm for optimizing the result subgraph is proposed. For this purpose, UTM code, a transformed representation of the adjacent matrix of graphs is designed to encode the topology of subgraph as individuals. Then corresponding evolutionary operators able to be directly performed on UTM code are given. Thus the efficiency of the algorithm is largely improved. The experimental results obtained on two real large scale graphs with different topology characteristics demonstrate that our method solves n-connection subgraph discovery problems effectively.*

1. Introduction

More and more data sets represented as network graphs with large size are available now. It is usually necessary or important to find some special structures from the networks. For example, from the social network modeling the relationship among people, can we find the solution to an interesting problem, what is the relationship between Alan Turing and Sharon Stone? Alan Turing is a departed famous scientist, and Sharon Stone is a famous actress of the time. They live in different time and their jobs are totally different. Is there any interesting relationship between them?

A feasible way to solve the problem is to construct a network on the basis of the relationship among people, in which every person is denoted as a node, and any two nodes are connected with an edge if the corresponding two persons have a certain kind of relationship. Furthermore, every edge is associated with a weight representing the strength of their relationship. With such network we can find the relationship between any two persons by discovering a subgraph containing these two nodes. Of course the subgraph should be of appropriate size, and also the nodes in the subgraph should be important enough. In such case we may say that the subgraph can best capture the relationship between the two persons.

One special case of the problem called 2-Keynodes Connection Subgraph Discovery Problem (CSDP) is first studied by C. Faliutsos et al. However as pointed out by them [1], there have been very few researches directly addressing the CSDP problem. C. Faliutsos proposed an efficient algorithm to tackle the 2-Keynodes CSDP based on the electricity analogues principal. They discovered an interesting result shown in Fig.1, which shows that there



Proceedings of the 18th IEEE International
Conference on Tools with Artificial Intelligence (ICTAI'06)
0-7695-2728-0/06 $20.00 © 2006 IEEE

Figure 1 Relationship between Alan Turing and Sharon Stone

are the three most important persons between Alan Turing and Sharon Stone: Harry Potter who is a science fiction character, Gillian Anderson, an actor in a movie about the engima cipher machine, and Kate Winslet, an actor of a popular science fiction television show.

However, if we would like to know the relationship among more people, for example, what is the relationship among Alan Turing, Sharon Stone and Mike Jordan, then how can we deal with the problem? In such case the proposed electricity analogues based approach does not work any more. We call this kind of more complicated problem an n-Connection Subgraph Discovery Problem, where n is greater than 2. The problem is formulated as follows:

N-Connection Subgraph Discovery Problem (*n*-CSDP)

Given: an edge-weighted undirected graph G, n interesting vertices $v_1, v_2, \ldots, v_n$ from G, and an integer budget β.

Find: a connection subgraph H containing $v_1, v_2, \ldots, v_n$ and at most other β vertices that well captures the relationship among $v_1, v_2, \ldots, v_n$.

To *n*-CSDP, the electricity analogues based method adopted in [1] does not work any more. The reason is that, electricity analogues based method is proposed on the basis of electrical currents in a network of resistors, thus lead their method only to be suitable for two nodes CSDP. So it is necessary for us to give a new approach to find the subgraph that captures well the relationships among *n* pre-specified nodes in a huge graph.

N-connection subgraph discovery may have wide applications. It can be used to find the several websites that most likely to leak some information (for example documents, mp3 or movies) on the Internet. Similarly it can be used to help us to control the infections. In other domains like semantic search and information retrieval, and in other networks like protein networks, language networks, chemical reaction networks, *n*-Connection Subgraph can also be useful for finding some special information. Thus it is more and more important for us to find effective approaches to discover special sub-structures from these networks.

In this paper, we present a novel method to solve *n*-CSDP. For this purpose, we first define our goodness function to measure how well a connection subgraph captures the relationship among the *n* pre-specified nodes. Meanwhile we will design a special UTM code to represent possible topology of a graph with fixed number of nodes. Based on the UTM code and our goodness function, we then design an evolutionary algorithm to evolve the topology of a graph by using specially designed genetic operators. Our experimental results show that the proposed algorithm can be used to discover the connection subgrapghs for different types of networks.

The rest of this paper is organized as follows. Section 2 gives a brief review of the related work. Section 3 presents our proposed algorithm. Section 4 shows our experimental results and brief analysis. The paper is concluded in Section 5.

2. Related Work

The first work directly addressing the connection subgraph mining problem is done by C. Faliutsos [1]. Then followed by that of S. Vast et al's [7]. The other indirect but related work includes community structures mining [3] and survivable networks [4], PageRank [5], RDF graph [12], graph clustering [10], graph partitioning [11], frequent subgraph mining [6] and other works on complex networks [9].

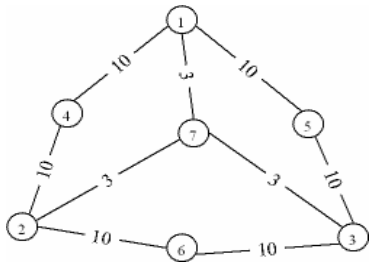


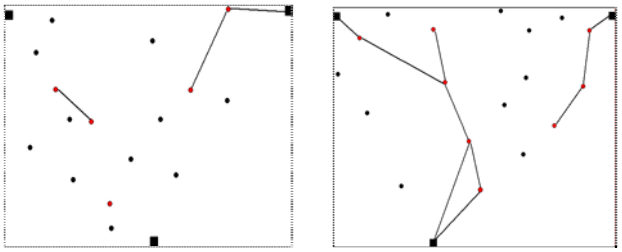
Figure 2: An example graph that C. Faliutsos's algorithm cannot find the connection subgraph among nodes 1, 2 and 3 as needed.

C. Faliutsos et al present an electricity analogues based method to find connection subgraph between two given nodes [1]. When the number of the given nodes is more than two, electricity analogues based method works any more. The reason is that electricity analogues based method is proposed on the basis of electrical currents in a network of resistors. Thus their proposed function of measuring the goodness of a connection subgraph is only suitable for a 2-Keynodes CSDP. To an *n*-CSDP where $n>2$, we may get some useless nodes, and meanwhile will miss some important nodes. For instance, suppose that node 1, 2 and 3 in Fig.2 are 3 Keynodes. If we still use their function designed for two Keynodes case, we may get a result subgraph excluding node 7. This is because that the degree of node 7 is 3 and the degree of any other node, i.e. node 4, 5, 6, is 2. So the weight of the edge connected to node 7 is small. But node 7 is an important node to describe the relationship among the three designated Keynodes and is actually the node that we hope to get.

S. Vast et al [7] present a method to extract a set of the nodes that best capture the relevant nodes among the *k* given nodes of interest. They project the nodes of the network which is viewed as an undirected graph into a

Proceedings of the 18th IEEE International
Conference on Tools with Artificial Intelligence (ICTAI'06)
0-7695-2728-0/06 $20.00 © 2006 IEEE

Euclidean space. Fig. 3 presents the results of the experiments of this method on the two real networks that we use in our experiments. It shows that both result subgrapghs are disconnected. Another problem of using S. Vast et al's algorithm to solve the n-CSDP is that the complexity of the algorithm is $O(n|G|^2)$. If the size of the given network is very large, the time cost of the algorithm is unacceptable.



(a) Subgraph obtained from scientific collaboration network

(b) Subgraph obtained from movie-actor network

Figure 3: Results of S. Vast et al's method on two real networks. The red nodes are the relevant nodes selected by the algorithm.

In [12], W.H. Milnor et al studied the problem of discovering informative subgraph in RDF Graphs with the electricity analogues approach. So in essence, their method is also only used in two nodes conditions just like [1], and is not suitable for n-CSDP (n>2). The work on the survivable networks [4] studies the ability of connection when some nodes of the network are missing. This work can also not be directly used to solve n-CSDP. For example, node 1 and node 2 in Fig.4 have the same survivability. When deleting any of them, some paths will be cut off. However, node 1 and node 2 have different importance to our problem since the degrees of these two nodes are quite different

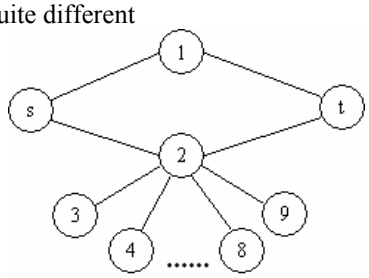


Figure 4: An example graph

3. Algorithms to find candidate subgraph and result subgraph

In the following we will first give some definitions to be used in the rest of the paper, then formulate the constraints to be considered in measuring the goodness of the result subgrapgh. To get the needed subgraph we will first extract a relatively bigger candidate subgraph compared with that of result subgraph through our neighbor-growth algorithm. Then we give a UTM code having good property to encode the topology of subgraph. On the basis of the UTM coding of individuals representing the possible result subgraph, the corresponding evolutionary operators are designed.

3.1. Preliminaries

3.1.1. Definitions

First, we give some definitions that we use in the rest of this paper.

Definition 1: **Keynodes**. N pre-specified nodes v_1, $v_2,...,v_n$ are called Keynodes if we need to find a connection subgraph containing them. The connection subgraph discovery problem having n Keynodes is called n-CSDP.

In social networks, as we have discussed in Section 2, famous persons are less important than those who are not famous. But how can we define whether a person is famous or not? In [1], the nodes that have small degrees are preferred. But actually whether a person is famous or not depends on whom he compare with. So we give a definition as follow:

Definition 2: **Starnodes.** The nodes having greater degrees than the greatest degree of all the Keynodes are called Starnodes.

3.1.2. How to judge the goodness of a subgraph

What we need to find is a connection subgraph that well captures the relationship among three given nodes. So an important problem is how to judge the goodness of a subgraph. Considering what we have discussed above, the result subgraph should satisfy the following constraints:

1) The subgraph must be connected well: It is obvious that the result subgraph must be connected. But what does a subgraph is connected well mean? Here it means that the result subgraph is not only a connection subgraph, but also is still connected by removing all the Keynodes. Under such condition, the topology of the subgraph may capture as well as possible the relationship among n given Keynodes. For example, if our result is like the graph shown in Fig.2 but excluding node 7. Indeed it is a connection subgraph, but node 4 is only connected to node 1 and 2, and is not connected to node 3 without node 1 and 2. The situation is the same for node 5 and node 6. Here we suppose that every Keynode is not connected directly to any other Keynodes.

2) The subgraph must have no more than a pre-specified number β of nodes. This is obvious because the size of our result subgraph is measured by the number of nodes.

Proceedings of the 18th IEEE International
Conference on Tools with Artificial Intelligence (ICTAI'06)
0-7695-2728-0/06 $20.00 © 2006 IEEE

3) The total weight of all the edges of the subgraph should be as heavy as it can be. How can a subgraph of a certain size capture the relationship among the Keynodes? The weight of an edge is heavier; the relationship between the corresponding two nodes is stronger. So the most natural way is to maximize the total weight of all the edges in the subgraph.

4) The number of the Starnodes appearing in the result subgraph must be constrained. As we have discussed in section 2, the Starnodes sometimes may not be important to represent the relationship among the Keynodes. This does not mean that the Starnodes are not needed in the result. The Starnodes may also help to represent the relationship among Keynodes. But too many Starnodes may not be well suited to capture the relationship among the Keynodes. So we hope to reflect this consideration by constraining the number of the appearances of Starnodes with some conditions.

The above four factors are the most important constraints that the result subgraph should satisfy. In our algorithm these factors will be considered in the judgement of the goodness of a subgraph.

Our algorithm can be divided into two steps: the first step is to generate the candidate subgraph to reduce the size of the computing space; the second step is to get the final result from the candidate subgraph with an evolutionary computational method.

3.2. Neighbor-growth based Algorithm to find candidate subgraph

Our goal is to find connection subgraph in a large graph, for example, the social networks which are complex graphs of very large size. Apparently searching directly in such large graph is a work of low efficiency. We also know that the mean distance between every two nodes in the complex network is small. For instance, in the condense matter physicist collaboration network the mean distance is 6.4 [8]. So we need to cut the nodes that are far away from the Keynodes, and extract a candidate subgraph.

Formally, this step takes vertexes $v_1, v_2,\ldots,v_n$ from the original graph G, and produces a much smaller graph G_{cand} by growing the neighborhoods around $v_1, v_2,\ldots,v_n$. But the neighborhoods should be grown carefully because we do not want to miss important nodes. We grow the subgraph according to the several principals we give out in section 3.1.2. In this step, the constraints of the candidate subgraph are as follows:

1) The candidate subgraph must be connected well.

2) The candidate subgraph must have more than β nodes, but not too big.

3) The total weight of the candidate subgraph's edges should be as large as it can be.

Here we can see that the second constraint is somewhat different from the constraint on the result subgraph. To make full use of the evolutionary computational algorithm in the second step, we consider it would be better that the size of the candidate subgraph is several times larger than the required size of result subgraph. We do not make any constraint on the number of the appearance of the Starnodes because the size of candidate subgraph is larger than that of the result subgraph and some Starnodes are useful.

In the following we give a function g to evaluate the connectivity importance of a node u:

$$g(u) = \sum_{i=1}^{n} \frac{weight(u, v_i)}{(length(u, v_i))^2}$$

where $length(u, v_i)$ is the length of the path with smallest number of the edges from node u to Keynode v_i, and $weight(u, v_i)$ is the sum of the weights of the edges on the path. To compute $g(u)$ for each u in the graph we traverse the graph n times, each time traversing from a different Keynode. It is obvious that if u is far from Keynodes, function $g(u)$ returns a low value. So if the function $g(u)$ returns a high value, it means that the node u is either near to a Keynode or not far from every Keynode.

Algorithm 1. Finding Candidate Subgraph containing most of the interesting connections among $v_1, v_2,\ldots,v_n$.

Input: a weighted and undirected graph G, n keynodes $v_1, v_2,\ldots,v_n$

Output: $G_{cond} \subset G$ which is much smaller than G but contains most of the interesting connections among $v_1, v_2,\ldots,v_n$.

```
Begin
  1) For each u in G and ∉ {v1, v2,…,vn} in G do
  2)     Compute g(u);
  3) maxu = the node u which has the biggest g(u);
  4) H = {maxu, v1, v2,…,vn };
  5) Gcond =H;
  6) For each vi in H do
  7)     k = length(u, vi);
  8)     Tk= {maxu};
  9)     Tk-1 = { }
  10)    While (k>1)
  11)        For each node t in Tk
  12)            Tk-1 = Tk-1 ∪ pickthemax (t, vi, c);
  13)            Gcond = Gcond ∪ Tk-1;
  14)        k--;
End
```

Proceedings of the 18th IEEE International
Conference on Tools with Artificial Intelligence (ICTAI'06)
0-7695-2728-0/06 $20.00 © 2006 IEEE

Figure 5: Algorithm of finding candidate subgraph

Our algorithm shown in Fig. 5 starts from the node *maxu* whose function value $g(maxu)$ is the greatest to grow the neighbors around *maxu*, each growing operation grows c neighbors for each node in T_k. In Algorithm 1 this growing operation is performed through the function $pickthemax(t, v_i, c)$ which returns c neighbors of node t. The weight of each neighbor of t is among the top c greatest ones and $length(t, v_i) = k\text{-}1$, where t is a node of set T_k. So we can control the size of candidate subgraph. For each Keynode v_i, growing operation can find the nodes that are on those paths from *maxu* to v_i whose total weights are among the biggest ones. This algorithm can find the most important nodes and keep the candidate subgraph small and well connected.

The computational complexity of finding *maxu* (step 1 and 2) is $O(nE)$ because each edge is needed to be traversed n times to compute the $g(u)$, where n is the number of the Keynodes and E is the number of the edges of the given graph. The computational complexity of finding neighbors (from step 6 to 14) is $O(n*2*E)$. Because each time of growing neighbors from *maxu* to v_i each edge will be visited twice in the worst case. So the computational complexity of our algorithm is $O(nE) + O(2nE) = O(nE)$.

3.3. Evolutionary Algorithm to find the result subgraph

In this step, the algorithm is given a candidate subgraph whose size is much smaller than the input graph and finds the final result subgraph which well captures the relationship among the given Keynodes. First every graph is uniquely coded, so that each different graph is differently coded. On the basis of this coding mode, an evolutionary computational method is presented to optimize the topology of the result subgraph.

3.3.1. Individual Representation with UTM Code

When using evolutionary computational method to get a good result subgraph we first need to give an individual coding method to represent an edge-unweighted undirected graph. The 0-1 adjacency matrix is usually used for its convenience. But it is obvious that if the size of a subgraph increases the scale of its adjacency matrix increases very fast. Given a graph that has n nodes, its 0-1 adjacency matrix consists of n^2 elements. For its symmetry only the upper triangular matrix is needed to be represented. So we give the UTM (Upper Triangular Matrix) code of a graph as follows:

Definition 4: UTM code. Given an edge-weighted undirected graph, its adjacency matrix $A_{n\times n}$, a 0-1 sequence (e_i) can be constructed based on the upper triangular part of the matrix, where $i = 1, 2, \ldots, l$, and l is the length of the sequence (e_i). In the sequence the k-th row is directly after $(k\text{-}1)$-th row, where $k = 2, \ldots, n$. The m-th subsequence $q_m = (e_j)$ consisting of s elements, where $j = m,\ldots,m+s\text{-}1$, i.e., $q_m = e_m,\ldots,e_{m+s-1}$, and $1 \le m \le \frac{n^2 - n}{2 \times s}$, can be represented as a certain decimal number u_m (if the length of the sequence is smaller than s, then add enough zeros to its end), where s is the number of the elements in q_m. The sequence consisting of $\lceil l/s \rceil$ decimal number u_i is called a **UTM code** U**,** i.e. $U = (u_1,\ldots,u_i,\ldots,u_{\lceil l/s \rceil})$, where u_i is called the i-th part of U, s denotes the **UTM code element length** and l denotes the **UTM code length**.

Given a graph and its adjacency matrix $A_{n\times n}$, the kth part of the UTM code u_k is defined as follows:

$$u_k = \sum_{p=1}^{s} a_{ij} \times 2^{s-p-1}$$

where a_{ij} is an element of $A_{n\times n}$, and

$$\left\lceil \frac{(k-1)s}{n} \right\rceil < i \le \left\lceil \frac{ks}{n} \right\rceil, 0 < j \le s,$$

$$k = \left\lceil \frac{(2n - i)(i-1)/2 + j - i}{s} \right\rceil,$$

$$p \equiv ((2n-i)(i-1)/2 + j - i) \bmod s, \ 0 \le p < s.$$

From the above we know that u_k has the following property:

$$\begin{cases} u_k \bmod 2^{s-p} \equiv 1 & if \ a_{ij} = 1 \\ u_k \bmod 2^{s-p} \equiv 0 & if \ a_{ij} = 0 \end{cases}$$

For example, Fig.6 shows a graph and its adjacency matrix. Suppose that UTM code element length s is set to be 8, and the sequence $(e_i) = (1010110001)$. Then the first eight zeros or ones can be represented by decimal number 172. The rest of the sequence is (01) and six zeros are added to its end to get (01000000) which is represented by a decimal number 64. So the UTM code of the graph is (172, 64), and its code length $l = \frac{n^2 - n}{2} = \frac{5 \times 5 - 5}{2} = 10$.

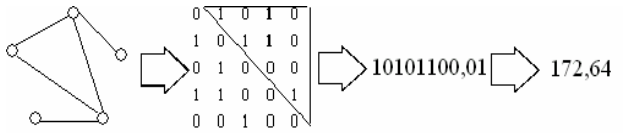


Figure 6: An example of the UTM code

The topology of a graph and its corresponding UTM code has the following relation:

Theorem 1: The mapping $f : GSet \rightarrow USet$ is a bijection, where dom $f = GSet$, ran $f = USet$ **,** ***GSet*** denotes the set of graphs each of which has n nodes, and ***USet*** denotes the set of UTM codes of graphs with n nodes.

Proof: Let's first prove that f is an injection.

Suppose that U_1 and U_2 are two different randomly selected elements from *USet*, i.e., $U_1 \neq U_2$, and their respectively corresponding UTM codes of length l are as follows:

$U_1 = (c_1, c_2, \ldots, c_p)$ and $U_2 = (d_1, d_2, \ldots, d_p)$, where $p = \lceil l / s \rceil$.

$\Rightarrow$ $\exists\, 1 \le i \le p$, such that $c_i \neq d_i$.

Meanwhile we suppose that U_1 and U_2 are decoded into the binary sequences (e_{1k}) and (e_{2k}) respectively, where $k = 1, 2, \ldots, l$.

$\Rightarrow$ $\exists\, 1 \le j \le l$, such that $e_{1j} \neq e_{2j}$

We also know that from sequence (e_{1k}) we can get its corresponding adjacency matrix denoted as $A_{n\times n}$, so we have $f(U_1) = A_{n\times n}$. To sequence (e_{2k}), we can also get its adjacency matrix $B_{n\times n}$ and have $f(U_2) = B_{n\times n}$

$\Rightarrow$ $f(U_1) = A_{n\times n} \neq B_{n\times n} = f(U_2)$

$\Rightarrow$ $f(U_1) \neq f(U_2)$

$\Rightarrow$ The mapping f is an injection.

Then we prove that the mapping f is a surjection.

For $\forall\, U \in USet$, suppose that $U = (c_1, c_2, \ldots, c_p)$. We can decode U and get its corresponding binary sequence (e_i).

From sequence (e_i) we can get an adjacency matrix $A_{n\times n}$ which corresponds to a graph $G \in GSet$.

$\Rightarrow$ For $\forall\, U \in USet$ we can find its preimage $G \in GSet$

$\Rightarrow$ The mapping f is a surjection.

Considering that we have proved that the mapping f is both an injection and a bijection, thus we can obtain the conclusion that Theorem 1 is true. ■

A certain topological graph has its corresponding adjacency matrix that is different to those of other topological graphs. So different graphs have different UTM codes. Theorem 1 can ensure that the using of operators to UTM codes will create new UTM codes which make sense. The UTM code can also be used to represent the directed graphs where elements 0, 1, 2 and 3 will be used to construct the adjacency matrix. We can represent any part of the matrix by a quaternary number in a similar way to what we have done to the undirected graphs.

The benefit of adopting our UTM code to encode the graph is that the space cost of representing a graph will largely be reduced, especially when the size of the graph is very large. For example, to represent a graph of 15000 nodes, we have to use 225 million numbers in the adjacency matrix, but the UTM code (s is set to be 16) only need 0.7 million numbers, about 3% of that of the corresponding adjacency matrix need.

Furthermore, we do not need to decode the UTM code when performing crossover and mutation operators. We can use these operators without knowing the topology of a graph. The reason for this will be explained later.

3.3.2. Topology Evolution

The n-CSDP can be viewed as a topology optimization problem. The final result is the best topology that can satisfy the constraints mentioned in Section 3.1.2. So we evolve the UTM code of the candidate subgraph, judge its goodness by the topology of the subgraph H' and finally get the result subgraph.

a) Fitness Function

The goodness of the topology of the given subgraph *H'* is judged with a fitness function by considering the four constraints mentioned in Section 3.1.2. Here we divide them into the following two categories:

- Constraints on the edge and node:
  1. The total weight of the subgraph's edges should be as big as it can be.
  2. The appearance of Starnodes must be constrained.
- Constraints on the whole subgraph:
  1. The subgraph must have no more than the pre-specified number of nodes.
  2. The subgraph must be connected well.

To each individual, we first calculate its total weight according to the constraints on the edge and node. Then those individuals that violate the constraints on the whole graph will be punished and finally we can get the fitness value for each individual.

Before calculating the total weights of all edges, the weight of each edge e is normalized as follows:

$$w'(e) = \frac{w(e)}{w_{max} \times \alpha}$$

where $w(e)$ is the given weight of edge e, w_{max} is the max of all the weights of edges of candidate subgraph H', and α is a punishment factor. To any edge e, α is a constant bigger than 1 when one node of e is a Starnode, and α is 1 otherwise. In this way we can constrain the appearance of Starnodes.

Then the total weights of all the edges in the candidate subgraph H' is calculated as follows:

$$E(H') = \sum_{e \in H'} w'(e)$$

$E(H')$ is the fitness value of a subgraph H' which satisfies the constraints on the whole graph.

There are three cases that the subgraph H' violates the constraints on the whole subgraph:

(1) At least one of the Keynodes is not connected.

(2) H' is not connected well. (Mentioned in 3.1.2)

(3) There are more than β nodes in H'.

Here the violation of the constraint that the subgraph H' must be connected well is divided into the two cases: 1) at least one of the Keynodes is not connected; 2) H' is not connected well. The reason for our division is that the individual belongs to case 1 is useless for evolution. For any subgraph H', the function P(H') for punishing its violation of the constraints is as follows:

P(H')=1

if at least one of the Keynodes is not connected

P(H')=0.00001; //case 1: severely punished

else if H' is not connected well

$P(H') = P(H) \times 0.5$; //case 2

else if there are more than β nodes in H'

$$P(H') = \frac{P(H')}{\varepsilon \times (sizeof(H') - \beta)}$$; //case 3

// ε is a constant greater than 1.

The final fitness function for subgraph H' is defined as follows:

$$fitness(H') = E(H') \times P(H')$$

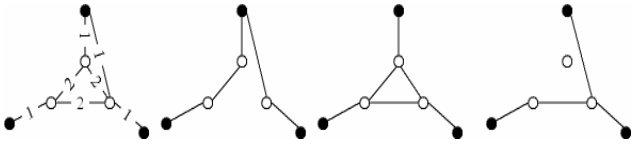
With this function each individual violating the constraints on the whole subgraph will be punished by reducing its fitness value. Fig.7 gives a graph H with 3 Keynodes shown as darkened nodes, the weights for each edge, and its three subgraphs. Suppose that β=5, l =2, α =1.5, we can get the fitness value of H_1, H_2 and H_3 according to the fitness functionas follows:

$fitness(H_1) = E(H_1)P(H_1)=(1+2/\alpha+1/\alpha+1/\alpha) \times 0.5=1.83$

$fitness(H_2) = E(H_2)P(H_2)=(1+2/\alpha+2/\alpha+1/\alpha+2/\alpha+1/\alpha) \times \frac{1}{2\times(6-5)}=1.83$

$fitness(H_3) = E(H_3)P(H_3)=(1+2/\alpha+1/\alpha+1/\alpha) \times 1=3.67$

From the above we can see that subgraph H_3 has the greatest fitness value and obviously H_3 is the subgraph that best captures the relationship among the 3 Keynodes according to our constraints.



(a) H (b) H_1 (c) H_2 (d) H_3

(a) is original graph H; (b), (c) and (d) are different topological subgraphs of H

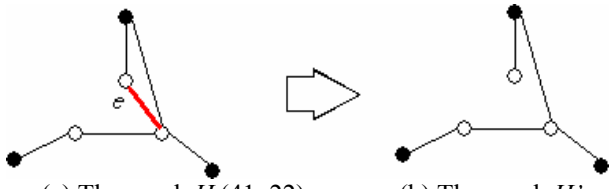
Figure 7: An example of a graph and its different subgraphs

b) UTM Mutation Operator

Given a UTM code $U = (u_1, u_2, ..., u_x, ..., u_n)$, $(0 \le u_i < 2^{s+1})$, the mutation operator is as follow:

$$u_x' = \begin{cases} u_x + 2^{s-k+1} & \textbf{if } u_x \bmod 2^{s-k+2} < 2^{s-k+1} \\ u_x - 2^{s-k+1} & \textbf{if } u_x \bmod 2^{s-k+2} \ge 2^{s-k+1} \end{cases}$$

This mutation operator only changes the connection condition of one certain edge e represented by the kth part of the subsequence corresponding to u_x in U and get new u_x', where k =1, 2, …s. If edge e is not connected in the topology of this individual, edge e is added in the topology; otherwise edge e is removed from the topology.



(a) The graph H (41, 22) before mutation (b) The graph H' (41, 6) before mutation

Figure 8: An example of mutation operation

For instance, the UTM code of graph H shown in Fig. 8(a) is (41, 22). The mutation operation on the individual will remove one edge. Suppose that the edge to be removed is red edge of graph H. Then after mutation the second part of its UTM code will change from 22 to (22 - 2^4) = 6. So the UTM code of graph H' in Fig. 8(b) is (41, 6).

To avoid that the mutation operator to add an edge which does not exist in the original graph to produce the nonsense subgraph, we should restrict this operator only to perform mutation on the edges really existing in the

candidate subgraph. So the algorithm can constringe much faster than without this restriction.

Even if we do not know the topology of a graph, we can also perform this mutation operator to the graph. We need not decode the UTM code to the corresponding adjacency matrix. Thus it saves a lot of time spent on the mutation operation for every generation.

c) UTM Crossover operator

The UTM crossover operator in our algorithm is similar to normal one-point crossover operator. Its operation is: select a crossover point in UTM code randomly, exchange the parts before the point of two individuals to get two new individuals.

(a) two individuals *A* and B before crossover (b) two new individuals *A*' and *B*' after crossover

Figure 9: An example of crossover operation

For instance, as shown in Fig.9, given two individuals *A* and *B*, *s* = 8, the crossover operation will be performed directly on the UTM code of *A* and *B*.

For the special property of UTM code, the crossover operator can exchange the topological subparts around one node of two individuals and produce two new individuals. Similar to the mutation operation, we also need not know the topologies of the two individuals. This will also be helpful to save a lot of time spent on the crossover operation in every generation. If needed, multi-point crossover operator or other type crossover operators can also be easily designed.

4. Experiment and Analysis

Two practical graphs are used in our experiments. They are two social networks, a kind of complex networks, i.e., scientific collaboration network and movie-actor network. Each node in the social networks represents a person and each edge between any two nodes represents the corresponding two persons exist a kind of relationship.

The scientific collaboration network and movie-actor network are small-world networks. The structures of the two networks are different [13]. The scientific collaboration network is a scale-free network that is characterized by a vertex connectivity distribution that decays as a power law. The movie-actor network is a broad-scale network which is characterized by a connectivity distribution that has a power law regime followed by a sharp cutoff. The experiments performed on these two different kinds of networks show that our algorithm has the ability to solve *n*-CSP on the networks with different topological characteristics.

All experiments were performed on a PC with Pentium IV 2.0GHz CPU and 512M main memory running Microsoft Windows 2000 Professional Edition. Our algorithms are implemented with Microsoft Visual C++ 6.0.

4.1. Scientific Collaboration Network

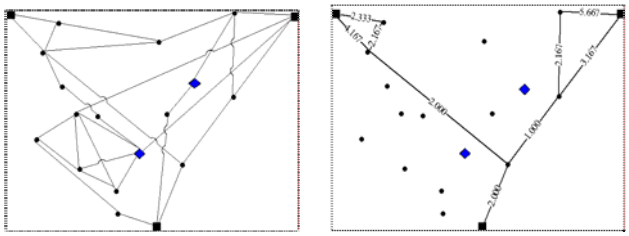
The Scientific Collaboration Network provided by M. Newman [8] has 16726 nodes and 95188 edges. It is a social network that represents the coauthor relationships among physicists. Each node in this network corresponds to a physicist. Each edge between two physicists has a weight, and the bigger the weight is, the stronger the coauthor relationship between them is. The weight w_{ij} of an edge between two physicists i and j is defined as:

$$w_{ij} = \sum_k \frac{\delta_i^k \delta_j^k}{n_k - 1}$$

If physicist *i* is one of the coauthors of the paper *k* then $\delta_i^k = 1$ else $\delta_i^k = 0$. The number of coauthors of paper *k* is denoted as n_k.

$$\sum_{j(\neq i)} w_{ij} = \sum_k \sum_{j(\neq i)} \frac{\delta_i^k \delta_j^k}{n_k - 1} = \sum_k \delta_i^k$$

The total weight of all edges for every physicist is equal to the sum of his coauthor papers. So the weight w_{ij} can represent the strength of the coauthor relationship between physicists *i* and *j*.



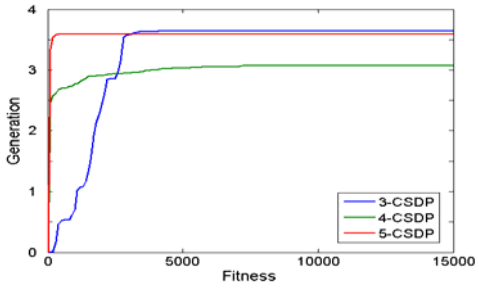
(a) Candidate subgraph (b) Result subgraph

Figure 10: The candidate subgraph and the result subgraph. The result subgraph shows the relationship among the three nodes (black square).

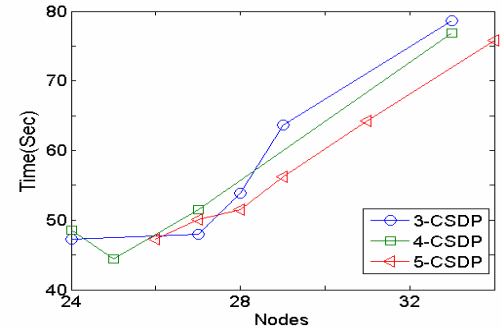
In our evolutionary algorithm, the size of population is set to be 30, UTM code length d is 8, and the termination generation is 15000. We choose three nodes representing three physicists Casati. G, Stern. A and Kim. D, in the network as the Keynodes.

Fig.10 (a) is the candidate subgraph that we get from the scientist collaboration network. It consists of 19 nodes

and 35 edges. Fig.10 (b) is the result subgraph. The square shape nodes are the Keynodes and the diamond shape nodes are the Starnodes. There are two Starnodes in the candidate subgraph. But we can see that the Starnodes are not included in the result subgraph. The result subgraph consists of 8 nodes and 9 edges, and it satisfies the constraints given before. This result subgraph can capture the relationship among the three Keynodes very well.



(a) Average Fitness Value



(b) Average Running Time

Figure 11: Performance on the scientist collaboration networks

In Fig.11 (a), β is set to be 15. The x-axis corresponds to the running generations of our evolutionary algorithm, and the y-axis corresponds to the average fitness value of all the individuals of a generation. The evolutionary algorithm converges in no more than 100 generations when solving 5-CSDP and converges after 5000 generations when solving 3-CSDP. This is because to a given graph when β is fixed, the bigger n is, the smaller the searching space is.

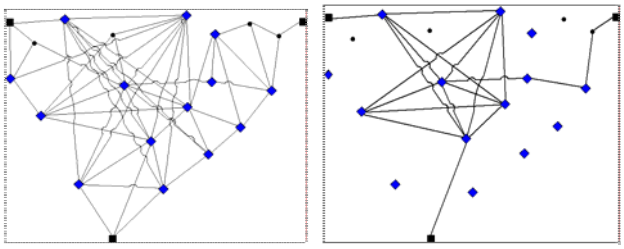
Fig.11 (b) shows the time spent when the running generation of our evolutionary algorithm is set to be 15000 for 3-CSDP, 4-CSDP and 5-CSDP respectively. The x-axis corresponds to the number of nodes that the candidate subgraph has, and the y-axis corresponds to the ten times' average running time of the algorithm. We can find that the n-CSDP that has bigger n cost a little less time than the smaller ones.

4.2. Movie-Actor Network

The movie-actor network constructed on basis of the IMDB by R. Albert and A. Barabasi has 392304 nodes and 15012983 edges [2]. Each node of the movie-actor network represents an actor, every edge means that the corresponding two actors have acted in one movie and the weight of the edge denotes the number of the movies that the two actors have acted together.

We performed the experiments on movie-actor network just like what we have done on the scientist collaboration network. The size of population is set to be 30, UTM code element length s is 8, and the termination generation is 15000.

Fig. 12 (a) shows the candidate subgraph, and Fig. 12 (b) shows the result subgraph. The candidate subgraph has 21 nodes and 53 edges. The movie-actor network has more connection and its structure is more complex than that of the scientist collaboration network. We choose three nodes that are less famous actors than those of many other nodes in the network. We find in the candidate subgraph that only 4 nodes are not Starnodes and in the result subgraph only one node is not Starnode. But the degrees of the Starnodes (Diamond shape node) in result subgraph are close to those of the Keynodes (Sqaure shape node). This means that the Starnodes correspond to the actors not very famous. So it shows that the obtained result subgraph can capture well the relationship among the three Keynodes.

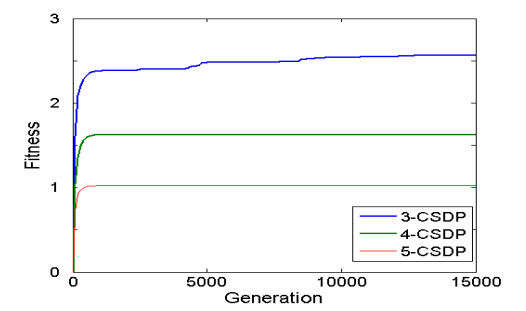


(a) Candidate subgraph (b) Result subgraph

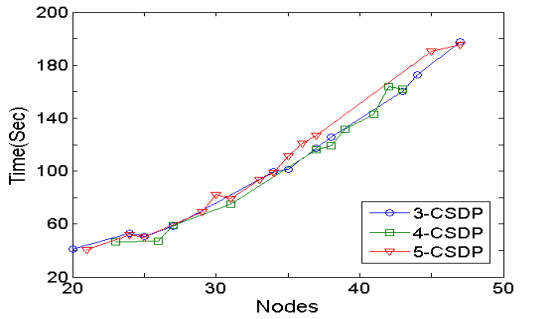
Figure 12: The candidate subgraph and the result subgraph. The result subgraph shows the relationship among the three black nodes.

In Fig.13 (a), β is also set to be 15, the x-axis corresponds to the running generations of our evolutionary algorithm, and the y-axis corresponds to the average fitness value of all the individuals of a generation. The evolutionary algorithm converges in no more than 100 generations when solving both 5-CSDP and 4-CSDP, and it usually converges after 5000 generations when solving 3-CSDP. This shows again that to a given graph when β is fixed, the bigger n is, the smaller the searching space is. Fig.13 (b) shows the ten times' average running time spent when the running generation of our evolutionary algorithm is set to be 15000 for 3-CSDP, 4-CSDP and 5-CSDP respectively. It also shows again that

the *n*-CSDP that has bigger *n* cost a little less time than the smaller ones.



(a) Average Fitness Value



(b) Average Running Time

Figure 13: Performance on the movie-actor networks

The experiments on the two different social networks show that our proposed approach solves *n*-CSDP efficiently and has good scalability. To a given graph when β is fixed, the bigger the *n* is, the smaller the searching space is. Thus the algorithm converges faster when *n* is set to be larger.

5. Conclusion

This paper proposes a method of solving the *n*-connection subgraph discovery problem. With our algorithm we can get a small subgraph that captures well the information about the relationships among more than 2 pre-specified nodes. The main contributions of the paper are as follows: First we propose a candidate subgraph generation algorithm which can get the most useful parts from the originally given huge graph. Then we define the goodness function to measure how well a subgraph can capture the relationship among the designated nodes. Third a UTM code to represent every topology of a certain size graph is designed. Our UTM code can guarantee that for each different the graph topology has its unique code representation. Last we provide an evolution algorithm to evolve the graph topology based on the UTM code and the special operators we designed.

The experiments on the real dataset demonstrate that our algorithm performs well on the complex networks having different topology characteristics, and also has good scalability. Furthermore the experimental results show that as the number of pre-specified number of Keynodes, i.e. *n* increases, the algorithm converges faster. In the future we will extend our work by considering the case that graphs may contain more than one type of entities, e.g. people, company, products.

6. References

[1] C. Faloutsos, K. Mccurley, A. Tomkins, Fast Discovery of Connection Subgraphs. KDD 2004: 118-127.

[2] R. Albert and A. Barabasi. Statistical mechanics of complex networks. Review of Modern Physics, 2002.

[3] L. Danon, A. Diaz-Guilera, J. Duch and A. Arenas, Comparing community structure identification, J. Stat. Mech. (2005) P09008.

[4] F. Ablayev, Lower bounds for one-way probabilistic communication complexity and their application to space complexity. Theoretical Comp. Sc., 157 (1996), 139--159.

[5] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[6] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraph in the presence of isomorphism. In Proc. of the 3rd IEEE International Conference on Data Mining (ICDM), 2003.

[7] S. Vast, P. Dupont and Y. Deville, Automatic extraction of relevant nodes in biochemical networks, Atelier Apprentissage et Bioinformatique, CAp 2005, Conférence d'Apprentissage, Nice, pp. 21-31, 2005.

[8] M. E. J. Newman. Structure of Scientific Collaboration Networks. Proc. Natl. Acad. Sci. USA 98, 404-409 (2001).

[9] Albert-László Barabási, Réka Albert. Emergence of scaling in random networks Science 286, 509 (1999).

[10] U. Brandes, M. Gaertler, and D. Wagner. Experiments on graph clustering algorithms. In Proc. 11th European. Symposium of Algorithms (ESA '03), LNCS 2832, pages 568–579. Springer-Verlag, 2003.

[11] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. The Bell System Technical Journal, 49(2):291–307, 1970.

[12] W.H. Milnor, C. Ramakrishnan, M. Perry, A.P. Sheth, J.A. Miller and K.J. Kochut, Discovering Informative Subgraph in RDF Graphs. 4th International Semantic Web Conference (ISWC 2005)

[13] L. A. N. Amaral, A. Scala, M. Barthélémy, and H. E. Stanley, Classes of small-world networks, Proc. Natl. Acad. Sci. U.S.A. 97(2000), 11149-11152.