On Finding Densest Subgraphs and their Applications

Allison Hoch*

December 4, 2009

*Research completed under the advisement of Dr. Samir Khuller and supported NSF REU. Department of Computer Science, University of Maryland, College Park, MD 20742. E-mail : allie@umd.edu

1 Introduction

There is an increasing amount of biological data being represented by graphs, eg. protein interactions, metabolic pathways, gene regulation, gene annotation, etc. Identifying highly connected or dense regions in graphs has both theoretical and practical significance. One way of finding these types of subgraphs is to identify dense subgraphs. **Density** is the sum of the weights of all edges in a subgraph divided by the number of vertices in the subgraph. The problem of finding a graph's densest subgraph can be solved in polynomial time despite the fact that a graph contains an exponential number of subgraphs [1, 2, 3].

In this paper, we apply the densest subgraph problem to *gene annotation graphs* in order to identify patterns and relationships in the graphs. Gene annotation graphs are bipartite graphs that describe the relationships between various gene and plant characteristics. By simply finding the densest subgraph in the gene annotation graph we have little control over the results and are given only one pattern. We propose various modifications to the densest subgraph problem to allow more control and provide more patterns. Since the goal in this setting is to identify patterns, it is feasible that the densest subgraph would not be the only reasonable solution. All subgraphs whose density is relatively dense would potentially represent patterns. In section 2, we look at ways of identifying all of the subgraphs with the greatest possible density, as well as subgraphs whose density is close to the maximum density in order to identify many patterns in each graph.

Another modification to the densest subgraph problem that we consider is finding the densest subgraph after being given a set of vertices required to be contained in the subgraph. This would allow a biologist studying particular gene or plant characteristics to force different combinations to be part of the dense subgraph, thus ensuring patterns that are identified are more meaningful in the context of what they are studying. This method is described in section 3.

The gene annotation graph does not describe the relationship between different gene characteristics or between different plant characteristics. As such, we add a restriction that can use an additional graph, or set of graphs, which describe the relationship between the vertices on different sides of the bipartite gene annotation graph. The distances between all pairs of vertices in these additional graphs can be computed to define a distance threshold on the vertices in the dense subgraph. For gene annotation graphs, there are two additional graphs, one describing the relationship between the gene characteristics and one describing the relationship between the plant characteristics. The distance threshold can ensure that the vertices in the dense subgraph are both highly connected in the original graph, but also closely related in the additional graphs. Described in section 4, we call this the *distance restricted dense subgraph problem* and it allows us to identify more meaningful dense subgraphs and patterns.

These methods and their applications to gene annotation graphs will be published in RECOMB 2010 (Conference on Research in Computational Molecular Biology) [4]. Sections 5 and 6 will describe this application and experiments on this type of data as presented in [4].

2 All Densest Subgraphs and Almost Dense Subgraphs

In this section, I describe an algorithm for computing all densest subgraphs as well as "almost" dense subgraphs, details of which can be found in [4]. Algorithms from Goldberg and Lawler [2, 3] find the densest subgraph in polynomial time using a series of s-t min cuts problems. The **min cut** is defined by two sets of disjoint vertices (A and B), one containing the source (s) and one containing the sink (t) such that the sum of the edges from A to B is minimized. The in-degree of the **source** must be zero, and the out-degree of the **sink** must also be zero. The following describes Goldberg's algorithm [2] to calculate the densest subgraph in a graph and Picard's algorithm [3] to calculate all min cuts, both of which are used to calculate all densest and "almost" dense subgraphs.

Goldberg's algorithm sets up a network flow graph by creating a source and a sink to add to an existing graph (G). Every undirected edge with weight w in the original graph is replaced by two directed edges with weight w. An edge from the source to every vertex in G is added with weight $m' = \Sigma_{e \in E(G)} w(e)$, where E(G) is the set of all edges in G and w(e) is the weight of edge e. An edge from every vertex in G to the sink is added with weight $m' + 2g - d_i$, where g is a guess of the density of the densest subgraph and d_i is the sum of all weights of edges in G adjacent to vertex i. The algorithm maintains an upper bound (u) and lower bound (l), initially set to zero and m' respectively. It performs a binary search while $l\text{-}u \geq 1/(n*n\text{-}1)$, where n is the number of vertices in the graph. In each iteration, g is set to (u+l)/2 and the network flow graph is reconstructed based on the current g. Then, the min cut is calculated. If the source is the only vertex on the source side of the cut (A), then u = g, otherwise l = g. When the algorithm terminates, the densest subgraph is equivalent to the vertices on the source side of the cut minus the source, with density g if G is unweighted. If G is weighted then g is not actually a guess of the density of G but merely a variable. The density must be calculated when the algorithm terminates and finds the set of vertices in the densest subgraph.

Picard's algorithm [5] can be used to calculate all min cuts in a graph. The first step is to calculate the max flow on a flow network graph (G). **Max flow** defines "flow" on each edge of the graph to maximize the flow leaving from the source and thus entering the sink. This flow must satisfy the following criterion: for every vertex, except the source and sink, the sum of all flow entering a vertex must equal the sum of the flow leaving the vertex. The flow on an edge may not exceed its weight (or capacity). The flow entering a vertex (v) is equal to the sum of the flow on all the edges directed at v, and the flow leaving v is equal to the sum of flow on all the edges directed from v. From max flow, a residual graph can be computed. In the **residual graph**, each directed edge from G is potentially replaced by two directed edges. Consider an edge from u to v with flow (f) and capacity c. In the residual graph there will be an edge from u to v with weight c –f and an edge from v to u with weight f. Picard's algorithm next computes the strongly connected components of residual graph. **Strongly connected components** are disjoint sets of vertices of a graph such that every vertex in a set can reach every other vertex in the set. The component containing the sink, called T, and all its predecessors can be removed because these components will never be on the source side of the cut. Next, the component containing the source, called S, and all its successors are removed since these components will always be on the source side of the cut. Finally, the closure is computed on the remaining graph. Every closure combined with S and its successors represents a set of vertices on the source side of the cut. The

closure is all subsets of vertices such that for every vertex in a subset, all its predecessors are also in the subset. The closure can be computed efficiently using [6].

2.1 All Densest Subgraphs

To find all densest subgraphs the last step in Goldberg's algorithm needs to find all min cuts not just one min cut. This can be done using Picard's algorithm [5]. After termination of Goldberg's algorithm to find the densest subgraph, I run Picard's algorithm to find all min cuts. Then each set of vertices on the source side of the cut (minus the source) represents a densest subgraph. An outline of this algorithm is in section 2.2.

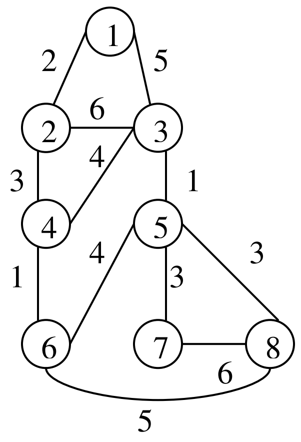
2.2 Almost Dense Subgraphs

To compute "almost" densest subgraphs we remove vertices that were almost not included in the strongly connected components or the closure on the strongly connected components because these vertices do not increase the density significantly. To achieve this goal, "small" weighted edges in the residual graph, R, must be removed. In the case of an unweighted graph with integer weights, removing edges in the residual graph will not produce interesting results because all possible edges have a weight of zero or one. But in the weighted case, removing low weighted edges in the residual graph and then proceeding to calculate the strongly connected components will produce not only all densest subgraphs but also subgraphs whose density is close to the density of the densest subgraph. If the density of the densest subgraph is D and all edges in the residual graph with weight less that α are set to zero, then all subgraphs with density $D + \alpha$ will be computed. The following outlines the algorithm to find all densest subgraphs and "almost" dense subgraphs:

```
$l \leftarrow 0$; $u \leftarrow m'$;
while $u - l \geq 1/(n*(n-1))$:
    $g \leftarrow (u + l) / 2$;
    Construct the flow network N as described above;
    Find min-cut $\{A, B\}$;
    If $A == \{s\}$ then $u \leftarrow g$;
    Else $l \leftarrow g$;

Construct the residual graph R of N where edges $< \alpha$ are set to zero
Compute a graph of the strongly connected components of R, called SCC
Remove T and its predecessors from SCC
Remove S and its successors from SCC, called them SS
All densest subgraphs $\leftarrow$ closure on remaining SCC combined with $(SS – \{s\})$
```

The following is an example of the results of the above description. Consider the following graph:



If α is set to 1, the following subgraphs are computed:

{1, 2, 3, 4}	density = 5.0
{5, 6, 7, 8}	density = 5.25
{2, 3, 4, 5, 6, 7, 8}	density = 5.142857
{1, 2, 3, 4, 5, 6, 7, 8}	density = 5.375

One subgraph, {4, 2, 3} with density 4.333 is computed but discarded because its density is less than D – α = 5.375 – 1 = 4.375.

2.3 An approximation to the densest subgraph

An approximation method described by Charikar in [1] gives a two approximation of the densest subgraph, which we found to be incredibly accurate for the real world data used in sections 5 and 6. The following describes the algorithm:

```
max ← density of graph G
maxGraph ← G
while G is not empty
    remove vertex with min degree from G
    if density of G > max
        max ← density of G
        maxGraph ← G
output maxGraph as the densest subgraph.
```

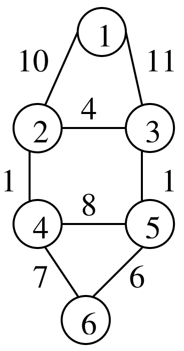
3 Dense Subgraphs with a Specified Set

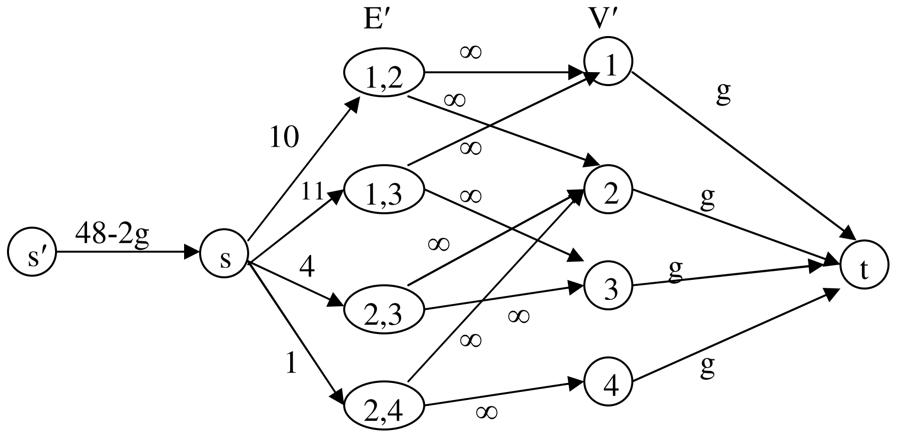
This section will present two methods we develop in [4] for finding the densest subgraph with a specified set, one using flow based methods and the other using linear programming. Each method augments current methods to finding dense subgraphs to allow this added ability. For the flow based method, Lawler's algorithm is modified [3], and for the linear programming method work by Charikar [1] is modified. One advantage of the flow based method is that it can be combined with the methods from the previous section to find almost dense subgraphs with a specified set.

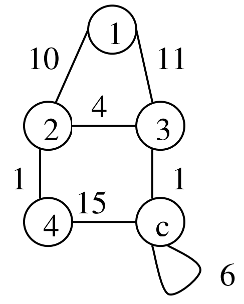
3.1 Finding subgraph using s-t min cuts

Lawler's method of finding dense subgraphs is similar to Goldberg's algorithm [2] previously described, but it constructs the flow network differently. The following is a description of the method for finding the standard densest subgraph. There is a vertex in the flow network for each vertex in the original undirected graph (G), call this set V′, and a vertex for each edge in G, call this set E′. There is an edge from the source (s) to each vertex e in E′ with weight equal to the weight of e in G. There is an edge from each vertex v in V′ to the sink (t) with weight equal to g*w(v), where g is a guess of the density of the densest subgraph and w(v) is the weight of the vertex v. Finally there are two edges added from each vertex e in E' (corresponding to edge from x to y in G) to x and y in V′ with weight ∞. To find the optimal subgraph the min cut is computed a number of times using a binary search to identify the optimal g. To initialize, the lower bound is set to zero and the upper bound is set to the sum of the weights of all edges in G (or w′(E)). At each iteration if the capacity of the min cut is less than w′(E) then the lower bound is set to g. Otherwise, if only s is in the source side of the cut (V_1) then the upper bound is set to g. Finally, if V_1 contained more than the source the optimal solution has been found. The dense subgraph corresponds to $V_1 \cap V'$.

To modify this method to find the densest subgraph for a specified set C the first step is to contract all the vertices in C into one vertex c. All edges between vertices in C are now self loop edges (or one self loop with weight equal to the sum of all edges in C) and w(c) = the sum of the weights of vertices in C. Next, C is removed. The flow network is set up similarly, except for the vertices in C being contracted and removed. Also a new source (s′) is added with an edge from s′ to s with weight w′(E) – g*w(c). The initialization of the lower bound and upper bound can be set more accurately than before so that the lower bound is initialized to the weight of all edges in C divided by w(c) and the upper bound is initialized to w′(E)/w(c). At each iteration the testing criterion for setting the lower bound and upper bound are similar except that if the capacity of the min cut is less than w′(E) – g*w(c) the lower bound is set to g. For the final solution of the densest subgraph, C is added to the set of vertices in the densest subgraph specified by the flow network. The following is an example of this method, the images on the left is the original graph G and G with C contracted, and the image on the right is the flow network. The set of vertices {5,6} are being forced into the optimal subset. The vertices in E' are named for the vertices they connect in G.







The resulting g is equal to 8 and when g is set to 8 there are more than one possible min cut (as expected). In one possibility $V_1 = \{s', s\}$ and in another possibility V_1 equals everything except t, and in both cases the capacity of the cut is 48. Therefore the most dense subgraph containing the subset {5,6} is the vertices {1,2,3,4,5,6} which has density 8. (If there was no forced subset, the densest subgraph would be {1,2,3} with density 8.333.

3.2 Finding subgraph using linear programming

The second method for finding dense subgraphs with a specified set was developed by modifying Charikar's linear programming method for finding dense subgraphs [1]. The basic method (without a specified subset) is to solve the following linear program, where E is the set of edges in some graph G:

$$\begin{gathered}\max \textstyle\sum_{i,j}\ x_{i,j}\\ \text{all } i,j \in E\ x_{i,j} \le y_i\\ \text{all } i,j \in E\ x_{i,j} \le y_j\\ \textstyle\sum_i\ y_{i,} \le 1\\ x_{i,j},\ y_j\ \ge 0\end{gathered}$$

Let $S(r) = \{i: y_i \ge r\}$. For each possible r (or each unique value of y_i) check the density of S(r) computing S(r) with the greatest density, which corresponds to the densest subgraph. When solving this problem without a specified subset, for all i,j ϵ S(r), $y_i = x_{i,j} = 1/|\text{maximum } S(r)|$. This fact leads to the approach taken to force a subset C to be part of the densest subgraph.

To modify this problem to work with specified subsets, for all i ϵ C $y_i = 1/p$. p is the size of the densest subgraph, and since this is not known to begin with, the linear program must be solved for all possible p. The new algorithm is:

```
For p = |C| to n
    Solve the above linear program with the added restriction that for all i ϵ C $y_i = 1/p$
    Solve for the densest subgraph by checking each value of r.
```

The densest subgraph computed over all iterations of the loop is the densest subgraph containing C. In this case there are no conclusions about the final values of y_i and $x_{i,j}$.

4 Distance Restricted Densest Subgraph Problem

The problem of finding distance restricted dense subgraphs is motivated by the following situation. Finding the dense subgraph is equivalent to finding a highly connected or related subgraph. Assuming the graph being searched describes some sort of relationship between the vertices in the graph, what if there is another measure of the vertices relationship that must also be taken into account? With the distance restricted densest subgraph problem additional graph(s) can describe other relationship(s) (a small distance is equivalent to a close relationship and a large distance is equivalent to a distant relationship). Then, a threshold can be set so that only subgraphs where every pair of vertices with a distance in the additional graphs of less than the threshold are allowed.

Unfortunately, this problem is NP hard for general graphs. I implemented two methods for approximating a solution. The first step is to calculate distances between all pairs in the additional graphs. Then, the first method iterates through every vertex (v) in the original graph (G) and picks every vertex within threshold/2 from v. For each of these sets of vertices, calculate the densest subgraph to find the overall densest subgraph. The problem with this method is that the subgraph that is identified may not be as dense as the optimal solution but it is guaranteed that threshold will be obeyed (see [4]).

The second method is similar, but instead picks every vertex within the threshold from v. With this method there is no guarantee that that the distance threshold is met but the subgraph identified is at least as dense as the optimal solution. (see [4]).

5 Applications

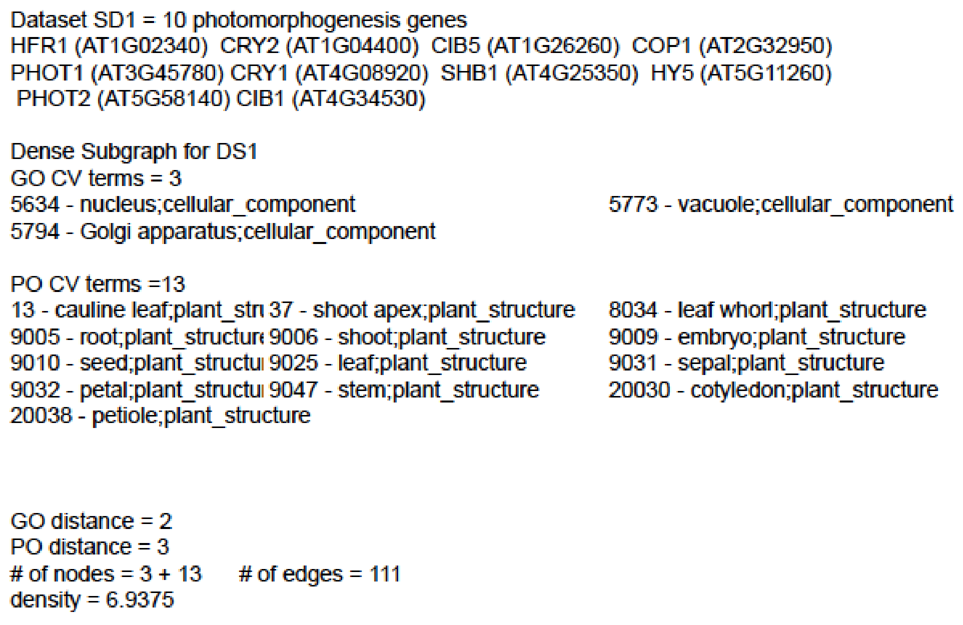
One application of using dense subgraphs to find patterns is gene annotation graphs. These graphs are bipartite graphs, which describe the relationship between gene and plant ontologies (or characteristics). Every gene and plant ontology (GO and PO) are nodes in the graph and are associated with a gene. The edges connecting the GO an PO terms represent some sort of relationship that has been noticed in the lab. Biologist interested in a certain set of genes would like to look at the GO and PO terms associated with these genes to identify patterns and discover new relationships. The methods discussed above can be helpful in identifying such patterns.

These graphs are a good application of almost dense subgraphs (section 2) because biologists are simply looking for patterns, and it is likely that they would not be solely interested in the densest subgraph. Finding dense subgraphs with a specified subset (section 3) would be helpful to biologist because they could specify GO or PO terms in which they are particularly interested. Finally, the distance restricted subgraph problem (section 4) is applicable because in addition to the bipartite graph (G1), there are also graphs describing relationships between the GO and PO terms. There are two graphs, one with only GO terms (G2_go) and one with only PO terms (G2_po). Relationships between vertices in the G1 graph, which are far apart in either G2 graphs are not very significant. More meaningful patterns will be found by computing the distances between all vertices in the G2 graphs and limiting the allowed distance when computing dense subgraphs on G1.

We tested these methods on the gene annotation graphs and the findings were examined by a biologist. She found the patterns we identified interesting validating because while some were patterns she did not expect to see, many were patterns that could be confirmed by current literature. One advantage of our method is that future patterns could be identified much more quickly and could then be verified in the lab without as much trial and error. Some of the patterns we identified that are verified by literature took months or years to identify.

6 Experiments

The following are graphs and charts prepared for [4] using the gene ontology graph. The following charts describe the densest subgraphs identified for a set of 10 genes and 20 genes (section 2). When processing the gene ontology graph only a portion of the graph was examined at once since it is such a large graph. This is why the charts are associated with genes because in each chart, only the GO and PO terms connected to a set of genes were considered.



Dataset SD1 = 10 photomorphogenesis genes
HFR1 (AT1G02340) CRY2 (AT1G04400) CIB5 (AT1G26260) COP1 (AT2G32950)
PHOT1 (AT3G45780) CRY1 (AT4G08920) SHB1 (AT4G25350) HY5 (AT5G11260)
PHOT2 (AT5G58140) CIB1 (AT4G34530)

Dense Subgraph for DS1
GO CV terms = 3

5634 - nucleus;cellular_component	5773 - vacuole;cellular_component
5794 - Golgi apparatus;cellular_component	

PO CV terms =13

13 - cauline leaf;plant_str	37 - shoot apex;plant_structure	8034 - leaf whorl;plant_structure
9005 - root;plant_structur	9006 - shoot;plant_structure	9009 - embryo;plant_structure
9010 - seed;plant_structu	9025 - leaf;plant_structure	9031 - sepal;plant_structure
9032 - petal;plant_structu	9047 - stem;plant_structure	20030 - cotyledon;plant_structure
20038 - petiole;plant_structure		

GO distance = 2
PO distance = 3
# of nodes = 3 + 13 # of edges = 111
density = 6.9375

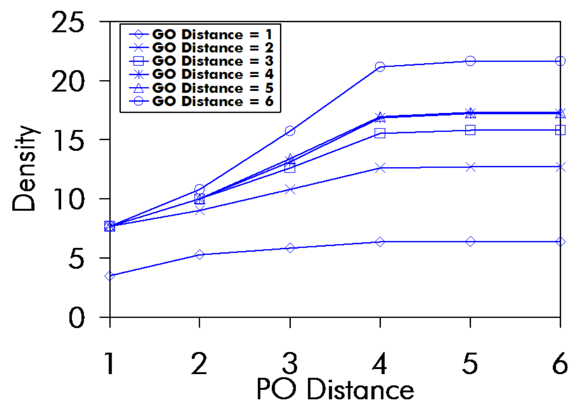
Dataset SD3
20 genes involved in different biological pathways.
FT, SOC1, FLC are involved in regulating flowering time.
AG, WUS, SEU, REV and BPE are involved in flower development.
ARF3 and ARF4 regulate auxin level, RDR6 and HEN4 are related to RNA regulation, etc.
These pathways may not be interconnected.
At4g16280, At2g33860, At5g60450, At1g65620, At2g27250, At5g10140,
At1g65480, At1g13440, At4g20910, At5g64390, At3g49500, At2g27100,
At2g45660, At3g48750, At5g60910, At1g43850, At1g59640, At2g01500,
At4g18960, At5g60690.

GO Nodes:
3676 - nucleic acid binding;molecular_function
3677 - DNA binding;molecular_function
3700 - transcription factor activity;molecular_function

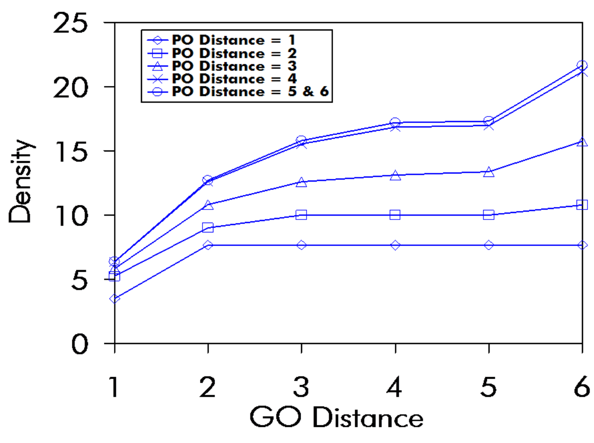
PO Nodes:
37 - shoot apex;plant_structure
230 -inflorescence meristem;plant_structure
9029 - stamen;plant_structure
9030 - carpel;plant_structure
9031 - sepal;plant_structure
9032 - petal;plant_structure
9046 - flower;plant_structure
9047 - stem;plant_structure
9052 - pedicel;plant_structure

GO Distance = 2
PO Distance = 3
# of Genes = 13 Number of GO=3 Number of PO=9 Number of edges =
Density = 11.667

Graphs (a) and (b) describe the density as the threshold allowed distance for the G2_go and G2_po graphs are varied (section 4).



(a) Impact of Varying the PO distance on Density



(b) Impact of Varying the GO Distance on Density

7 Conclusion

The methods presented in this paper could be helpful in identifying patterns in a number of situations, especially real world situations in which the most extreme results are not always the only results of interest. Finding almost dense subgraphs, dense subgraphs including a specified set and the distance restricted subgraph problem all represents ways of finding more meaningful patterns in real world data. For example, further biological applications include protein-protein interaction graphs in which there is a lot of recent research on how to identify patterns. Further experiments will done with our methods on the gene annotation graphs to determine how helpful the patterns we find are.

References

[1] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In APPROX, pages 84-95, 2000.
[2] A. V. Goldberg. Finding a maximum density subgraph. Technical report, 1984.
[3] E. Lawler. Combinatorial optimization - networks and matroids. Holt, Rinehart and Winston, New York, 1976.
[4] Saha, B., Hoch, A., Khuller, S., Raschid, L., Zhang, X. Dense Subgraphs with Restrictions and Applications to Gene Annotation Graphs. (submitted to RECOMB 2010).
[5] J.-C. Picard and M. Queyranne. On the structure of all minimum cuts in a network and applications. In Mathematical Programming Study13, pages 8-16, 1980.
[6] Scharage, L. and Baker, K.R. Dynamic Programming Solution of Sequencing Problems with Precedence Constraints. Operations Research Vol. 26 No. 3 1978.