September 1984 Report No. STAN-CS-84-1014

A P-complete Problem and Approximations to It

by

Richard Anderson and Ernst W. Mayr

Department of Computer Science

Stanford University
Stanford, CA 94305

Stanford University
Department of Computer Science

A P-complete Problem and Approximations to It

by

Richard Anderson* and Ernst W. Mayr

Abstract: The P-complete problem that we will consider is the High Degree Subgraph Problem. This problem is: given a graph $G = (V, E)$ and an integer k, find the maximum induced subgraph of G that has all nodes of degree at least k. After showing that this problem is P-complete, we will discuss two approaches to finding approximate solutions to it in $\mathcal{NC}$. We will give a variant of the problem that is also P-complete that can be approximated to within a factor of c in $\mathcal{NC}$, for any $c < \frac{1}{2}$, but cannot be approximated by a factor of better than $\frac{1}{2}$ unless $P = \mathcal{NC}$. We will also give an algorithm that finds a subgraph with moderately high minimum degree. This algorithm exhibits an interesting relationship between its performance and the time it takes.

This research was supported in part by an IBM Faculty Development Award and NSF grant DCR-8351757.
* Also supported in part by a National Science Foundation Graduate Fellowship.

1. Introduction

Problems which are P-complete under logspace reductions probably cannot be solved in a small amount of space. Because of the Parallel Computation Thesis [FW], an alternative way to view these problems is as problems that probably cannot be solved fast in parallel. Quite a few problems are known to be P-complete, such as the circuit value problem [L], linear programming [DLR] and unification [DKM]. Most of the combinatorial problems which are known to be P-complete rely on some mechanism which allows computations to be encoded in them. In the P-completeness proofs for network flow [GSS] and list scheduling [HM], large integer weights are used. The latter problem can be solved in $\mathcal{NC}$ if the weights are restricted to being small. Another mechanism that is used to force problems to being P-complete is to require that the solution be lexicographically minimum. Examples of P-complete problems of this type include finding the lexicographically minimum maximal independent set [C] or the lexicographically minimum maximal path [AM]. These results show that the natural greedy algorithms for the problems probably cannot be sped up using parallelism. This paper will discuss a P-complete problem which does not rely on any of these mechanisms. It can be considered a "purely combinatorial" problem. Since this problem does not rely on any special mechanism, it is an attractive problem for exploring schemes of approximate solution to P-complete problems. Two different approximation schemes will be discussed for the problem. One of them solves a variant of the problem to a factor arbitrarily close to half of the optimum in $\mathcal{NC}$. It is then shown that the problem cannot be approximated by a factor of better than one half unless $\mathcal{P} = \mathcal{NC}$. The second scheme finds a solution within a factor of the desired result. This algorithm has an interesting trade off between the time it takes and the degree of approximation achieved.

2. The High Degree Subgraph Problem

The High Degree Subgraph Problem is: given a graph $G = (V, E)$ and an integer k, find the maximum induced subgraph of the graph that has all nodes of degree at least k. There is a simple sequential algorithm for this problem which can be implemented to run in $O(|E|)$ time. The algorithm discards nodes of degree less than k until all nodes have degree at least k, or the graph is empty. The correctness of this algorithm follows from two easy lemmas. The first lemma establishes that there is a unique maximum induced subgraph of G with minimum degree at least k; it will be denoted by $HDS_k(G)$.

Lemma 1. *Let S and T be maximum induced subgraphs of G that have minimum degree at least k, then $S = T$.*

Proof: The induced subgraph on $S \cup T$ must have minimum degree at least k. Since S and T are maximum, $|S| = |T| = |S \cup T|$, so $S = T$. ∎

Lemma 2. *The sequential algorithm outlined above finds $HDS_k(G)$.*

Proof: Let S be the induced subgraph found by the algorithm. Since the nodes of S have degree at least k, $S \subseteq HDS_k(G)$. Suppose that $S \neq HDS_k(G)$. Let v be the first node of

$HDS_k(G)$ discarded by the algorithm and let T be the graph just before v is discarded. Since $d_T(v) < k$ and $HDS_k(G) \subseteq T$, v must have degree less than k in $HDS_k(G)$. Hence $S = HDS_k(G)$. ∎

It is possible that $HDS_k(G)$ is empty. The following lemma due to Erdős [E] establishes an importan t case when $HDS_k(G)$ is not empty.

Lemma 3. *If a graph has n vertices and m edges then it has an induced subgraph with minimum degree $\lceil \frac{m}{n} \rceil$.*

Proof: The result holds for graphs consisting of a single vertex. Suppose it holds for all graphs with less than n vertices. Suppose we have a graph with n vertices and m edges. If all vertices have degree at least $\lceil \frac{m}{n} \rceil$, then we are done. Otherwise the vertex with smallest degree can be deleted along with its incident edges, leaving a graph with $n-1$ vertices and $m - k > m - \lceil \frac{m}{n} \rceil$ edges. By the induction hypothesis it has an induced subgraph with minimum degree

$$\left\lceil \frac{m-k}{n-1} \right\rceil \geq \left\lceil \frac{m - \lfloor \frac{m}{n} \rfloor}{n-1} \right\rceil \geq \left\lceil \frac{m - \frac{m}{n}}{n-1} \right\rceil = \left\lceil \frac{m}{n} \right\rceil .$$

3. The High Degree Subgraph Problem is P-complete

The high degree subgraph problem can be reformulated as a decision problem HDS by asking if a specific node v is in $HDS_k(G)$. The proof that HDS is P-complete is a reduction from the Monotone Circuit Value Problem (MCVP). This problem is:

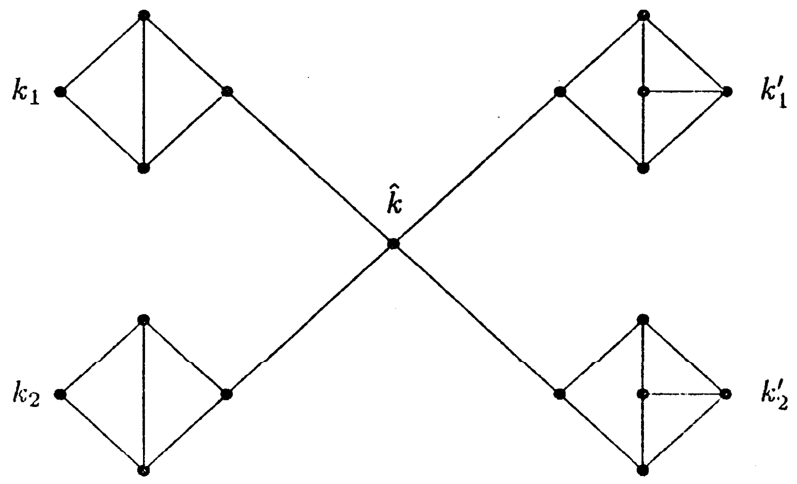
Given a circuit consisting of AND and OR gates and values for its inputs, compute the value of its output.

Here, a circuit β is a string $\beta_1, \ldots, \beta_n$ where $\beta_k \in \{0$ INPUT, 1-INPUT, AND(i,j), OR$(i,j)\}$. An AND(i,j) (resp., OR(i,j)) gate β_k forms the logical "and" (resp., "or") of the outputs of the gates β_i and β_j where $i < k$, and $j < k$. The gates β_i and β_j are called the inputs of gate β_k. We assume without loss of generality that AND and OR gates have out degree 1 or 2, and that 0-INPUTS and 1-INPUTS have out degree 1.

Theorem 1. *HDS is P-complete.*

Proof: To show that HDS is P-complete, a logspace reduction will be given which when given an instance of MCVP will construct a graph G with a distinguished node v, where v is in the maximum induced subgraph of degree k if and only if the output of the circuit is true. The proof will be for $k = 3$. A gate will be simulated by a collection of nodes. One of the nodes will give the value of the gate, it will be left in the maximum induced subgraph iff the gate's output is true. The false inputs are just an isolated node which will be removed. A true input consists of 4 nodes, connected to themselves. Since they therefore all have degree three at least three, they will never be removed. One of them is distinguished and will be connected to the gate that reads the input. The AND gates and OR gates have 4 distinguished nodes k_1, k_2, k'_1, k'_2. The nodes k_1, and k_2 are inputs to the gate and k'_1 and k'_2 are the outputs of the gates. If the output k'_x is connected to the input j_y of another gate, then there is an edge between k'_x and j_y. The gates are shown below. On both of the gates, k'_1 and k'_2 will only be removed if the node $\hat{k}$ is removed. In the OR gate, $\hat{k}$ will be removed if both k_1 and k_2 are removed, and in the AND gate, $\hat{k}$ is removed if either k_1 or k_2 is removed. The nodes k_1 and k_2 are removed if the connections into them are removed, in other words if they receive a false input. If a gate has a single output, the same construction is used and one of the outputs is left dangling. Note that the values will not propagate backwards through the circuit, so a node depends only on the nodes of the gates preceding it. Hence the graph simulates the circuit.

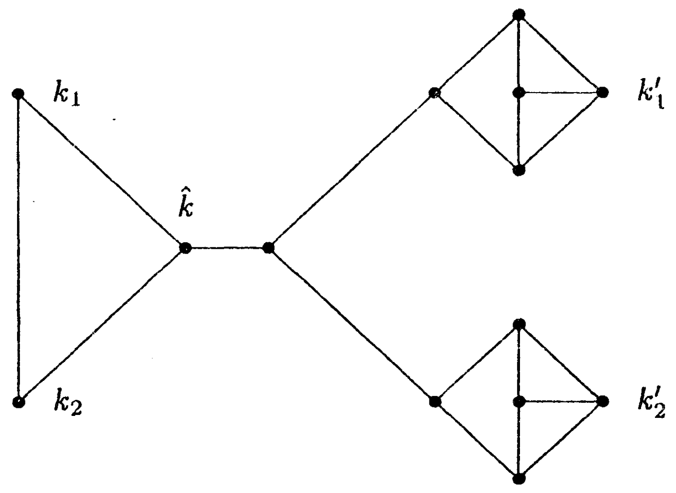
For $k > 3$, the same basic construction is used, the degrees of the nodes are just padded to bring them up to about degree k. First the graph G is constructed as outlined above. Then $k - 3$ nodes are added to the graph and these nodes are connected to all other nodes in the graph G'. It is clear that a node $v \in G$ will be in $HDS_k(G')$ iff it is in $HDS_3(G)$. ∎



OR gate

For the case $k = 2$, it is possible to find $HDS_k(G)$ in $\mathcal{NC}$. The algorithm for computing $HDS_2(G)$ has $\log n$ phases, where each phase removes all *chains*. A chain is a path of vertices that starts with a vertex of degree 1 and contains no vertex of degree greater than 2. The chains can easily be identified by path doubling techniques. When the chains are deleted, more nodes of degree 1 might be created, however each new node of degree 1 required the removal of at least two chains, so the number of chains removed decreases by at lease half at each phase. So we have the following theorem:

Theorem 2. *It is possible to compute $HDS_2(G)$ in $\mathcal{NC}$.* ∎

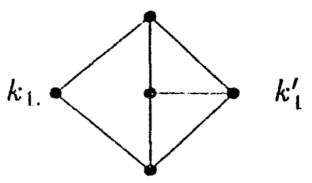


AND gate

The P-completeness result can be made slightly stronger. The stronger result is that it is P-complete to determine if $HDS_k(G)$ is nonempty. One implication of this is that it is probably not possible in $\mathcal{NC}$ to determine what is the degree of the highest degree induced subgraph that a graph has.

Theorem 3. *The problem of determining if $HDS_k(G)$ is nonempty is P-complete.*

Proof: The previous construction is modified so that all of the nodes are removed if the output of the final gate is removed. The case $k = 3$ will be described. The same method as was used in the first theorem may be used to extend this to other values of k. The 1-INPUTS are replaced by five vertices arranged:



The vertex k'_1 is connected to the gadget for the gate that it is connected to. The vertex k_1 is the leaf of a binary tree. This tree has the output of the final gate as its root. If the final output is removed, then all of the 1-INPUTS will be removed, and then everything will be removed. If the final output is not removed, then the nodes of the tree and the 1-INPUTS will not be removed, so the simulation proceeds as in the first construction. ∎

4. Approximations to P-complete Problems

If a problem is known to be P-complete, there is little hope of finding an $\mathcal{NC}$ solution for it. As is often done with NP-complete problems, we can lower our sights and attempt to find an approximate solution. The high degree subgraph problem is well suited for approximation. A variant of the problem which is an optimization problem is to ask what is the largest d such that $HDS_d(G)$ is nonempty. This value will be denoted by $\overline{HDS}(G)$. For this problem an approximation would be to find a d' where $\overline{HDS}(G) \geq d' \geq c\overline{HDS}(G)$ for some fixed $c < 1$. It will be shown that this problem can be approximated for $c < \frac{1}{2}$ in $\mathcal{NC}$, but cannot be approximated for $c > \frac{1}{2}$ unless $\mathcal{P} = \mathcal{NC}$.

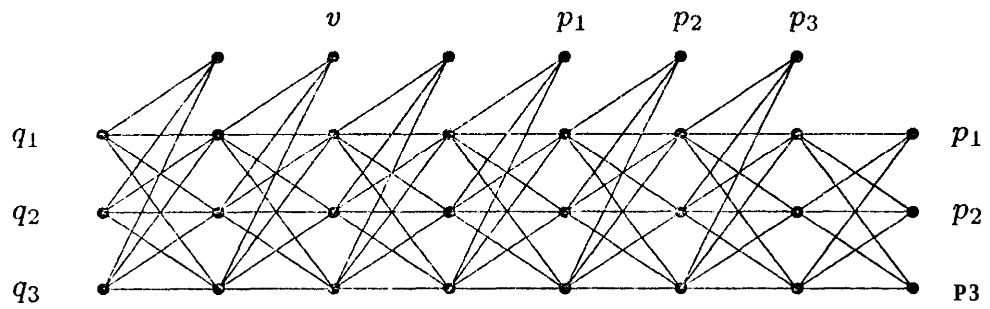
Theorem 4. *For any constant $c < \frac{1}{2}$, the optimization problem can be solved by an algorithm in $\mathcal{NC}$ to a factor of c.*

Proof: Let $\epsilon > 0$. The basic procedure for the algorithm is a routine $Test(k)$ which will return an answer which is either "the graph has no subgraph of degree at least k", or "the graph has a subgraph of degree at least $\frac{1-\epsilon}{2}k$". The routine $Test$ discards nodes from the graph until the graph is empty or less than $\epsilon n'$ of the nodes have degree less than k, where n' is the number of nodes currently in the graph. All nodes of degree less than k are discarded at each step. If the routine terminates with an empty set of vertices, the graph could not have had a subgraph with minimum degree k. If the algorithm terminates with n' vertices, then it must have at least $\frac{1-\epsilon}{2}kn'$ edges. By Lemma 3, it must have a subgraph with minimum degree at least $\frac{1-\epsilon}{2}k$. The procedure $Test$ is applied for each value between 0 and n. A value k will be found where the graph has a subgraph of degree at least $\frac{1-\epsilon}{2}k$ but no subgraph of degree $k+1$. ∎

The next theorem shows that the previous result is essentially the best possible assuming that $\mathcal{P} \neq \mathcal{NC}$. It will be shown that a circuit can be simulated by a graph which has $\overline{HDS}(G) = 2k$ if the output is 1 and $\overline{HDS}(G) = k+1$ if the output is 0. If the problem could be approximated by a factor of better than $\frac{1}{2}$ then the following construction could be used to solve the monotone circuit value problem.

Theorem 5. *If $\mathcal{P} \neq \mathcal{NC}$, then it is not possible to approximate $\overline{HDS}(G)$ by a factor greater than $\frac{1}{2}$ in $\mathcal{NC}$.*

Proof: This theorem will be proved by giving a logspace transformation of a monotone circuit to a graph which has $\overline{HDS}(G) = 2k$ if the output of the circuit is 1, and $\overline{HDS}(G) = k+1$ if the output is 0. This reduction is a variation of the reduction used in Theorem 1. The figures used will be for the case $k = 3$, the generalization to other values of k is straightforward. In the simulation, groups of k vertices will indicate the value of the wires. During the simulation, the vertices will all have degree $\geq 2k$ if the wire is true and degree at most $k+1$ if the wire is false. The circuit can be simulated by traversing the gates in order. The structure in the figure below is an expander, its purpose is to fanout values. It can be modified to fanout an arbitrary number of values by adding extra layers. The two vertices labelled p_1 are the same vertex, as are the vertices labelled p_2 and p_3. The vertices q_1, q_2, and q_3 are connected to the inputs to the expander. If we remove vertices of degree $2k$ or less then removing an output of the expander such as v will not cause any other vertex of the expander to be removed, so values will be propagated correctly.



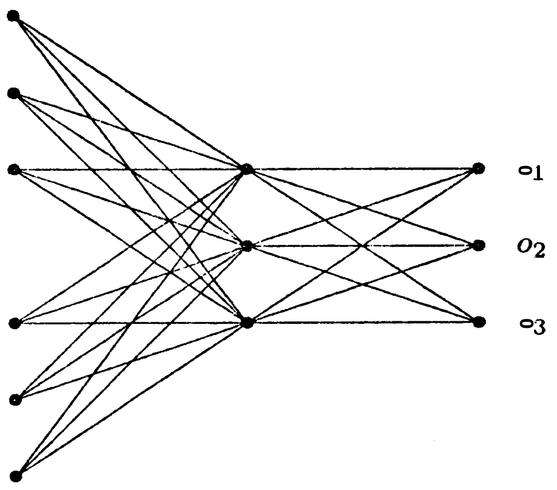
3-Expander

The OR and AND gates are illustrated in the figures below. The outputs o_1, o_2, and o_3 are connected to expanders, which are in turn connected to inputs of other gates. The expanders allow values to be fanned out and also insure that information is propagated correctly. In the AND gate, the long rectangle is a k^3-expander. The inputs of another expander are connected to the output of the circuit. The outputs of this expander are in turn connected to all of the inputs of the gates that are 1-INPUTS. 1-INPUTS will then have degree $2k$ and 0-INPUTS will be left with degree k. If the final output is false, the expander will be removed, and all of the 1-INPUTS will be removed.

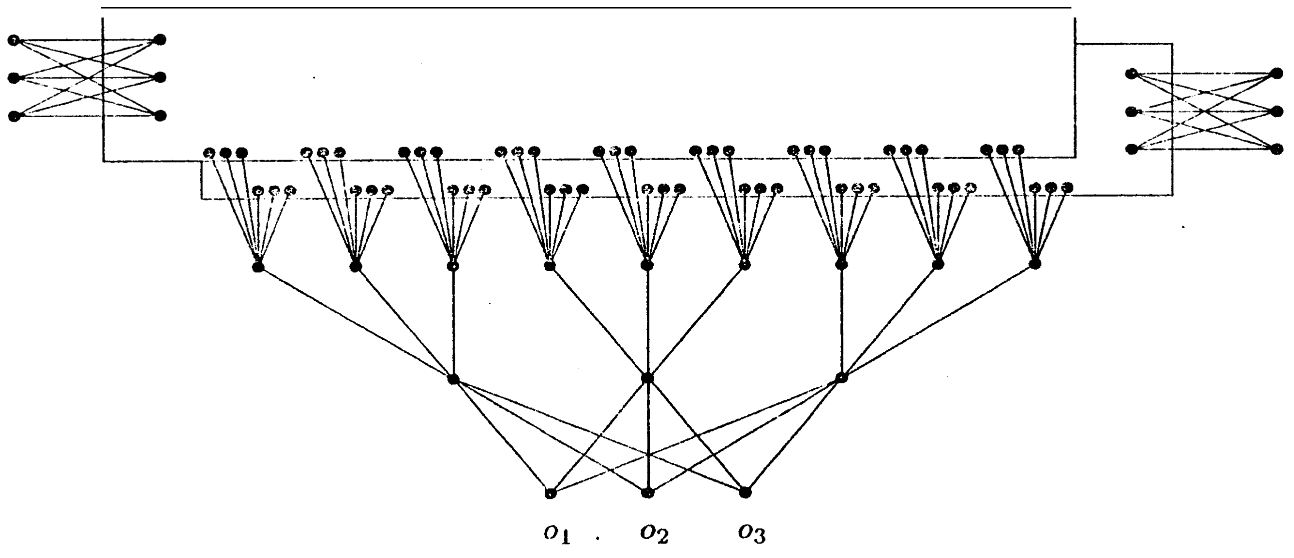
This graph simulates a circuit in a similar manner to the other simulations. Suppose the output of the circuit is true. If all nodes of degree less than $2k$ are removed, then the outputs of the false gates will be removed, and the outputs of the true gates will be left. Since the final output will not be removed, the 1-INPUTS will be left, this will leave a graph with minimum degree $2k$. If all nodes of degree $2k$ or less are removed, clearly everything is removed, so in this case $\overline{HDS}(G) = 2k$. Suppose that the final output of the circuit is false. If all nodes of degree at most $k+1$ are removed, the outputs of all the false gates will be removed. The removal of the output of the final gate will cause all of the 1-INPUT gates to be removed, which in turn will cause the rest of the nodes to be removed. Thus, if the output is false, $\overline{HDS}(G) \leq k+1$. Since the expander construct has a subgraph of minimum degree $k+1$, $\overline{HDS}(G) = k+1$. ∎

5. Finding a High Degree Subgraph

A second approach to approximating the High Degree Subgraph problem is to attempt to find a subgraph with high degree without insisting that it is the maximum subgraph with that degree. As long as we are looking for a subgraph with minimum degree d, where $d = \left\lceil \frac{|E|}{|V|} \right\rceil$, we know that such a subgraph exists. The difficulty that arises in this problem is that when nodes of degree d are discarded from the graph, the number of nodes of degree less than d still in the graph might actually increase. It appears to be difficult to identify which nodes will become nodes of degree less than d. A way that the number of nodes that



OR gate



AND gate using a k^3-expander

are to be discarded can be controlled is to throw out nodes of degree much less than d. If nodes of degree $\frac{d}{4}$ are removed, then the number of nodes that initially have degree at least d that get removed is bounded. This is formalized in the following lemma:

Lemma 4. *Let $G = (V, E)$ be an n node graph with k nodes of degree less than d, then G contains a subgraph with minimum degree $\frac{d}{4}$ that has at least $n - \frac{3}{2}k$ nodes.*

Proof: Suppose nodes of degree less than $\frac{d}{4}$ are removed until all nodes have degree at least $\frac{d}{4}$. Let S_d be the set of nodes that initially have degree at least d that are removed. Before a node of S_d is removed, it must have had $\frac{3}{4}d$ edges removed that go to it. Each node that is removed can have at most $\frac{d}{4}$ edges which go to members of S_d that are removed after it is. Putting these two facts together we have $\frac{3d}{4}|S_d| \leq \frac{d}{4}k + \frac{d}{4}|S_d|$, so $|S_d| \leq \frac{k}{2}$. Hence at most $\frac{3}{2}k$ nodes are removed. ∎

The lemma provides a way to find a subgraph with minimum degree $\frac{d}{4}$ in $O(n^{1/2}\log n)$ time. The algorithm is:

```
while at least $n^{1/2}$ nodes have degree less than $d$ do
    remove nodes of degree less than $d$;
while there is a node of degree less than $\frac{d}{4}$ do
    remove nodes of degree less than $\frac{d}{4}$;
```

Each iteration of a loop can be done in $O(\log n)$ parallel time. The first loop can not be executed more than $n^{1/2}$ times since it removes at least $n^{1/2}$ elements each iteration. The lemma insures that the second loop will not remove more than $\frac{3}{2}n^{1/2}$ elements, so it will also have $O(n^{1/2})$ iterations. This algorithm can be generalized to more than two phases. A phase is run as long as a certain number of elements can be discarded. When a new phase is run, the degree bound is reduced. The algorithm maintains two counts, *bound*, and *threshold*. If there are more than *threshold* vertices of degree less than *bound*, then they are discarded, otherwise the values of *bound* and *threshold* are altered. The function $f(n)$ determines the runtime and the performance of the algorithm. The previous algorithm corresponds to the second algorithm with $f(n) = n^{1/2}$. The second algorithm is:

```
begin
    $threshold := n/f(n)$;
    $bound := d$;
    while $threshold \geq 1$ and there is a node of degree less than $bound$ do
        $S := \{v | l(v) < bound\}$;
        if $|S| \geq threshold$ then
            $V := V - S$
        else
            $bound := bound/4$;
            $threshold := threshold/f(n)$
end.
```

Lemma 5. *The algorithm finds a subgraph with minimum degree at least*

$$\frac{d}{4^{\log_{f(n)} n - 1}}$$

in at most $\frac{3}{2}f(n)\log_{f(n)} n$ iterations.

Proof: A phase will be considered to be the group of iterations for which *bound* and *threshold* have fixed values. There will be $\log_{f(n)} n$ phases, since the value of *threshold* is reduced by a factor of $f(n)$ at the end of each phase and threshold is initially $\frac{n}{f(n)}$. When a phase ends, less than threshold vertices have degree less than ***bound***, so the minimum degree of the subgraph that is found will be the value of ***bound*** when ***threshold*** $= 1$ which is:

$$\frac{d}{4^{\log_{f(n)} n-1}}.$$

When a phase begins, there are less than $f(n)\cdot$*threshold* nodes of degree less than $4\cdot$*bound*. Lemma 4 says that by removing nodes of degree less than *bound*, at most $\frac{3}{2}f(n)$. ***threshold*** nodes will ***be*** removed. Since at least ***threshold*** nodes are removed at each iteration, there are at most $\frac{3}{2}f(n)$ iterations per phase. Hence there are at most $\frac{3}{2}f(n)\log_{f(n)}$ n iterations altogether. ∎

If different values are used for $f(n)$ an interesting time/performance trade off is exhibited. When f(n) = logn the algorithm is in $\mathcal{NC}$ and a subgraph with minimum degree $O(dn^{-\epsilon})$, for any $\epsilon > 0$ is found. Time and performance figures are given in the table below. The time is given as the total number of iterations, neglecting the factor of $\frac{3}{2}$. The performance is the ratio of $4d$ to the degree of the minimum degree subgraph that is found.

	Time	Performance
$f(n)$	$f(n)\ \log_{f(n)} n$	$4^{\log_{f(n)} n}$
$n^{1/k}$	$kn^{1/k}$	4”
4”	$\frac{4^k \log n}{2k}$	$n^{1/k}$
$\log^k$ n	$\frac{\log^{k+1} n}{k \log\log n}$	$n^{2/k \log\log n}$
$2^{\sqrt{\log n}}$	$2^{\sqrt{\log n}}\sqrt{\log n}$	$2^{2\sqrt{\log n}}$

Time/performance relationship

6. Conclusions

This paper has introduced a new P-complete problem, the High Degree Subgraph problem. This problem is interesting because natural approximations for its solution can be defined. In this paper, we explored two such approaches for approximate solutions. The first approach showed that it is possible to find, with an $\mathcal{NC}$ algorithm, the largest degree for which there exists a nonempty subgraph satisfying this degree bound, up to any factor smaller than $\frac{1}{2}$.Analogous to the situation for certain NP-complete problems, we also showed that the existence of any approximation algorithm in NC for this problem with a factor strictly bigger than $\frac{1}{2}$ would actually imply that $\mathcal{P} = \mathcal{NC}$.

The second approach for an approximation to the High Degree Subgraph Problem yields an approximation scheme trading time for the minimum degree of the subgraph found by the algorithm. For a specific setting of the parameters, it gives an $\mathcal{NC}$ algorithm which, given $d < n$, finds a subgraph that has minimum degree $O(dn^{-\epsilon})$. An open problem is whether this result can be improved to find a subgraph with minimum degree $O(d \log^{-k})$ or even O(d) in polylog parallel time.

An area of future research is to find other P-complete problems that can be efficiently approximated in some natural sense. Considering the Parallel Computation Thesis, a very closely related problem is to explore time/space trade-offs for approximation problems.

References

[AM] Anderson, R. , Mayr, E. "Parallelism and Greedy Algorithms", Technical Report No. STAN-CS-84-1003, Computer Science Department, Stanford University, April 1984.

[C] Cook, S.A., "The Classification of Problems which have Fast Parallel Algorithms", Technical Report No. 164/83, Department of Computer Science, University of Toronto 1983.

[DKM] Dwork, C., Kanellakis, P.C., Mitchell, J.C., "On the Sequential Nature of Unification", 1983.

[DLR] Dobkin, D., Lipton, R.J., Reiss, S., "Linear Programming is Log-Space Hard for P", *Information Processing Letters*, 8 (1979), pp 96-97.

[E] Erdős, P., "On the Structure of Linear Graphs", *Israel Journal of Mathematics*, 1 (1963), pp 156-60.

[FW] Fortune, S., Wyllie, J., "Parallelism in Random Access Machines", *Proc. 10th ACM STOC* (1978), pp 114-118.

[GSS] Goldschlager, L.M., Shaw, R.A., Staples, J., "The Maximum Flow Problem is Log Space Complete for P", *Theoretical Computer* Science, 21 (1982), pp 105-111.

[HM] Helmbold, D., Mayr, E., "Fast Scheduling Algorithms on Parallel Computers", 1984.

[L] Ladner, R.E., "The Circuit Value Problem is Log Space Complete for P", *SIGACT News* 7, 1 (1975), pp 583-590.