**Solving Connected Dominating Set Faster than 2^n**

**1. ABSTRACT**

* In the n-node undirected graph we find minimum cardinality such that each node not in S is adjacent to some node in S. Where S is minimum cardinality connected.
* This is equivalent to finding the maximum number of leaves in a spanning tree
* The best known exact time algorithm for such kind of problems is Ω(2^n) which gives all the subset of the nodes
* I are trying to reduce this 2^n barrier to O(1.9407^n) for connected dominating set problem.
* I use all the new domination rules and its analysis based on measure and conquer technique

**2.Key words**

* Exponential-time exact algorithm, NP-hard problem, connected dominating set, maximum leaf spanning tree.

**3.Introduction**

* The recent interest in exact exponential algorithms has several motivations. Indeed, there are applications that require exact solutions of NP-hard problems, although this might only be possible for moderate input sizes. Decreasing the exponential running time, say, from O(2n) to O(20.9n), increases the size of the instances solvable within a given amount of time by a constant multiplicative factor. This kind of improvement can be crucial in several applications. On the other hand, the study of exact algorithms leads to a better understanding of NP-hard problems, and initiates new combinatorial and algorithmic challenges.
* In this paper I consider the classical NP hard problem, the CONNECTED DOMINATING SET problem (CDS).
* A connected dominating set of a graph G =(V,E) is a subset of nodes S ⊆ V such that S is a dominating set of G and the sub graph of G induced by S is connected.
* The Connected Dominating Set problem asks to ﬁnd a connected dominating set of smallest possible cardinality.
* This is equivalent to finding a spanning tree with maximums number of leaves.
* Another recent application of this problem is in wireless ad-hoc networks: a small connected dominating set is often a convenient backbone to route the ﬂow throughout the network.(see .e.g. [3])
* The problem is NP-hard [1] and there is a (lnΔ + O(1))-approximation algorithm, where Δ is the maximum degree [2]. Such an approximation guarantee cannot be improved unless NP ⊆ DTIME (n^O(loglogn)) [3].
* The current best exact algorithm for Connected Dominating Set is the trivial Ω(2n) enumerative algorithm, that tries all possible subsets of nodes. Better results are known for the general (unconnected) version of the problem [4, 5, 6, 7]: the current best algorithm for Dominating Set has running time O(1.5137n) [8].
* Indeed, Connected Dominating Set belongs to a family of non-local problems that turns out to be particularly hard to solve exactly. Probably the best known example of this kind of problem is the Travelling Salesman Problem.
* ﬁnd a minimum cost tour that visits all the nodes of a weighted graph. The fastest known algorithm for this problem, which dates back to the sixties [9], is based on dynamic programming and has running time Ω(2n).
* A relevant example is Steiner Tree: ﬁnd a minimum size subtree of a given graph spanning a given subset of k nodes. For this problem an O(1.4143n) time algorithm can be obtained by combining the O((2 + €)^k n^O(1)) dynamic-programming (exponential space) algorithm in [10] (for small k), with trivial O(2n−knO(1)) enumeration of Steiner nodes (for large k). Finding a polynomial space algorithm faster than 2n is still open.
* Another very recent example is a O(1.7548n) algorithm for Feedback Vertex Set: ﬁnd a minimum cardinality subset of nodes of a graph whose removal makes the graph acyclic [11, 12].

**3.My results**

* By presenting the ﬁrst algorithm for Connected Dominating Set that breaks the 2n barrier: our recursive algorithm takes polynomial space and runs in time O(1.9407n). The algorithm is based on the simple strategy “stay connected”, which means that all partial solutions generated recursively must be connected.
* As a second contribution of this paper, we establish a lower bound of Ω(4n/5) for the worst case running time of our algorithm.

**4. The Algorithm**

* Let G =(V,E) be an-node undirected and simple graph. The open neighbourhood of a node v is denoted by N(v)={u ∈ V : uv ∈ E}, and the closed neighbourhood of v is denoted by N[v]=N(v)∪{ v}. The sub graph of G induced by a set S ⊆ V is denoted by G[S]. A set S ⊆ V of nodes of G is connected, if G[S] is connected.
* ASSUMPTIONS

1. The graph is connected (otherwise there is no solution)

2. The minimum connected dominating set has cardinality at least two (otherwise the problem is trivially solvable in polynomial time).

**EXPLAINING ALGORITHM**

The reduction rules are:

(a) If there is a candidate v which is a promise, select it (add it to S);

(b) If there are two candidates v and w (which by(a) are not promises)such that N(v)∩F ⊆ N(w)∩F, discard (add it to D);

(c) If there is an available node v which does not dominate any free node, discard v.

The algorithm branches according to the following rules:

(A) If there is a candidate v which dominates at least three free nodes w1,w2, andw3, or which dominates an available node w such that, after selecting v, w does not dominate any free node, branch on the two sub problems

• (S1,D1)=(S ∪{v},D);

•(S2,D2)=(S,D∪{v}).

(B) If there is a candidate v which dominates a unique free node w, let U ={u1,u2,...,uk}= N(w)∩A\N[v]be the set of the available neighbours of w which are not in the closed neighbourhood of v. Branch on the three sub problems:

• (S1,D1)=(S,D∪{v});

•(S2,D2)=(S ∪{v,w},D);

• (S3,D3)=(S ∪{v},D∪{w}∪U).

Observe that we might be discarded or a promise. Moreover one of the ui’s could be a promise. In those cases one or more sub problems are infeasible, and the algorithm simply halts on such infeasible sub problems. The same kind of situation may happen also in the following cases.

(C) If there is a candidate v which dominates two free nodes w1 and w2, namew1 and w2 such that if w2 is available (a promise), so is w1. Let Ui={ui,1,ui,2,...,u i,k i}= N(w i)∩A\N[v] be the available neighbours of w i which are not in the closed neighbourhood of v.

There are three diﬀerent subcases: (C.1) If w1 and w2 are adjacent, w1 is available and w2 is discarded, branch on the three sub problems:

• (S1,D1)=(S,D∪{v});

•(S2,D2)=(S ∪{v,w1},D);

• (S3,D3)=(S ∪{v},D∪{w1}∪U1).

(C.2) If w1 and w2 are adjacent and both available, branch on the four sub problems:

• (S1,D1)=(S,D∪{v});

•(S2,D2)=(S ∪{v,w1},D);

• (S3,D3)=(S ∪{v,w2},D∪{w1});

•(S4,D4)=(S ∪{v},D∪{w1,w2}∪U1 ∪U2).

(C.3)Otherwise(either w1 and w2 are not adjacent, or they are adjacent and both discarded), branch on the ﬁve sub problems

• (S1,D1)=(S,D∪{v});

•(S2,D2)=(S ∪{v,w1},D);

• (S3,D3)=(S ∪{v,w2},D∪{w1});

•(S4,D4)=(S ∪{v},D∪{w1,w2}∪U1);

• (S5,D5)=(S ∪{v},D∪{w1,w2}∪U2).

**Theorem 1** (correctness) The algorithm above computes a minimum cardinality connected dominating set.

**Analysis**

Altogether the weight of the problem is

k = k(G,S,D)=k(S,D)=∑ v∈F α(v)+ ∑v∈A β(v). (1)

In order to simplify the analysis, we make the following assumptions: • for a given available node v, β(v)=β ∈(0,1] if v is a promise and β(v) = 1 otherwise.

• α(v)=α|v| is a non-decreasing function of the frequency of v, denoted by |v|, that is the number of available nodes which dominate v (recall that v does not dominate itself). More precisely we assume

0=α0 = α1 <α 2 <α 3 = αi, ∀i≥4.

The reasons for the simplifying assumptions above will be clearer from the analysis. Note that the size of the original problem is upper bounded by (1+α3)n. For notational convenience, we deﬁne

Δαi = αi −αi−1,i ≥1.

Intuitively, Δαi is the reduction of the size of the problem due to the reduction from i to i−1 of the frequency of a free node

**Theorem 2** (running time)

The running time of the CDS algorithm of Section 2 is O(1.9407n).

**An Exponential Lower Bound**

**Theorem 3**

(lower bound) The worst case running time of the CDS algorithm of Section 2 is Ω(4n/5) = Ω(1 .3195n).

***REFERENCES:-***

[1] M. R. Garey and D. S. Johnson. Computers and Intractability. A Guide to the Theory of NPCompleteness. Freemann, 1979.

[2] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. Algorithmica, 20(4):374–387, 1998

[3] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. Algorithmica, 20(4):374–387, 1998

[4] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and Conquer: Domination - A Case Study, Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005), Springer LNCS vol. 3580, 2005, pp. 191–203.

[5] F. V. Fomin, D. Kratsch, andG. J. Woeginger. Exact(exponential) algorithms for the dominating set problem. Proceedings of the 30th Workshop on Graph Theoretic Concepts in Computer Science (WG 2004), Springer LNCS vol. 3353, 2004, pp. 245-256.

[6] F. Grandoni. A note on the complexity of minimum dominating set. Journal of Discrete Algorithms, 4(2):209–214, 2006.

[7] B. Randerath and I. Schiermeyer. Exact algorithms for MINIMUM DOMINATING SET. Technical Report, zaik-469, Zentrum f¨ur Angewandte Informatik K¨oln, April 2004.

[8] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and Conquer: Domination - A Case Study, Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005), Springer LNCS vol. 3580, 2005, pp. 191–203.

[9] M. Held and R.M. Karp. A dynamic programming approach to sequencing problems. Journal of SIAM, pages 196–210, 1962.

[10] D. M¨olle, S. Richter, and P. Rossmanith. A faster algorithm for the Steiner tree problem. Proceedings of the 23d Symposium on Theoretical Aspects of Computer Science (STACS 2006), Springer LNCS vol. 3884, 2006, pp. 561-570.

[11] F. V. Fomin, S. Gaspers, and A. V. Pyatkin. Finding a minimum feedback vertex set in time O(1.7548n), in Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC 2006), Springer LNCS vol. 4169, pp. 184–191.

[12] I. Razgon. Exact computation of maximum induced forest. Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006). Springer LNCS vol. 4059, p. 160-171.

**Sharp Separation and Applications to Exact**

**and Parameterized Algorithms**

***Abstract***

This research paper is basically for drawback of many divide-and-conquer algorithms that employ the fact that the vertex set of a graph of bounded treewidth can be separated in two roughly balanced subsets by separator. In this they are proving that trade off between size of separator and sharpness to fix the size of the two sides of the partition. This result is a powerful tool to design exact and parameterized algorithms for NP-hard problems. There are two applications, first is a O(2n+o(n))-time algorithm for the DEGREE CONSTRAINED SPANNING TREE problem and second is with running time O(16k+o(k) +nO(1)) for the k-INTERNAL OUT-BRANCHING problem.

***Introduction***

The point of parameterized and correct calculations is to take care of NP-difficult issues precisely, with the smallest possible worst-case running time. Parameterized algorithms seek to perform better when the instance considered has more structure than a general instance to the problem. Exact and parameterized algorithms have relation. Latest techniques to design and analyze exact algorithms are Inclusion-Exclusion, Subset Convolution, Measure & Conquer, and Iterative Compression. Best-studied problems, are Independent Set, (Connected) Dominating Set, Steiner Tree, Feedback Vertex Set, Coloring, Satisfiability, Traveling Salesman and Hamiltonian Path, and many others.

***Result***

This paper contains prove of a trade-off between the size of the separator S and the sharpness with which we can fix the size of VL and VR in the partition, for graphs of treewidth t.

**Lemma 1 (Sharp Separation)** Let G = (V ,E) be a graph of treewidth t and w : V → {0, 1}. Then for any integer p ≥ 0 and 0 ≤ x ≤ w(V ), there is a partition (VL,S,VR) of V such that |S| ≤ t · p, w(VL) ≤ x + w(V ) 2p , w(VR) ≤ w(V ) − x +w(V ) 2p , and there is no edge in G with one endpoint in VL and the other endpoint in VR, that is, S separates VL from VR. Given a tree-decomposition of G of width t, Here the weight function w is used to model a subset W ⊆ V of vertices that we wish to separate.

**DEGREE CONSTRAINED SPANNING TREE (DCST**). Given a graph G = (V ,E) and a function D : V →2{1,...,n}. Find a spanning tree T of G maximizing |{v ∈ V : dT (v) ∈ D(v)}|. S can be computed in polynomial time.

DCST normally sums up numerous NP-hard crossing tree and way issues contemplated in the writing. For example, we can code the acclaimed HAMILTONIAN PATH problem, discover a spreading over way of a given chart, by letting D(v) = {1, 2} for all vertices; A crossing tree with n hits is a Hamiltonian way. Another illustration is the FULL DEGREE SPANNING TREE issue, in which scanning for a traversing tree which amplifies the quantity of vertices having an indistinguishable degree in the chart from in the tree.

Second application of Sharp separation is **k-INTERNAL OUT-BRANCHING**: Given a digraph G = (V ,E) and a positive integer k, check whether there exists an out-branching with at least k internal vertices.

k-INTERNAL SPANNING TREE, was first studied by Prieto and Sloper , who gaving algorithm with running time 24k log knO(1) and a kernel of size O(k2) for the problem. Recently, Fomin et al. gave an improved version of it with running time 8knO(1) and a kernel with at most 3k vertices.

Definition An (n, t)-universal set F is a set of functions from {1, . . . , n} to {0, 1}, such that for every subset S ⊆ {1, . . . , n}, |S| = t, the set F|S = {f |S | f ∈ F} is equal to the set 2S of all the functions from S to {0, 1}.

Theorem 3 There is a deterministic algorithm with running time O(2t tO(log t)n log n) that constructs an (n, t)-universal set F such that |F| = 2t tO(log t) log n.

**Degree Constrained Spanning Tree**

It is O(2n+o(n))-time algorithm for the DEGREE CONSTRAINED SPANNING TREE problem (DCST). We rather consider a weighted generalization of the problem. Suppose we have undirected graph G = (V ,E), with node weights w : V →R≥0, and a list of desirable degrees D(v) for each vertex v. The hits hit(G) of a subgraph G is the set of nodes v such that dG(v) ∈ D(v). Our goal is to find a spanning tree T of maximum weight w(hit(T )) :=v∈hit(T ) w(v).

**k-Internal Out-Branching**

We are stating parameterized algorithm with running time O(16k+o(k) + nO(1)) for the k-INTERNAL OUT-BRANCHING problem. When we combines the Sharp Separation Lemma with the divide-and-color paradigm in, and a polynomial-size kernel for the problem. The biggest difference wrt the algorithm in previous section is that the Sharp Separation Lemma is used to divide the problem into balanced (rather than very unbalanced) subproblems.

***Conclusions***

This paper consists of simple separation theorem for graphs of bounded treewidth, which turns out to be a useful tool in the design of divide-and-conquer algorithms, both exact (exponential) and parameterized as stated in abstract and our aim. We demonstrated the applicability of our theorem by giving an algorithm for k-INTERNAL OUT-BRANCHING which is algorithm of runtime of O(16k+o(k) + nO(1)) time and an algorithm for DEGREE CONSTRAINED SPANNING TREE which is algorithm of runtime of O(2n+o(n)). It will be we efficient if we are able to find further applications of our separation result in the fields of exact and parameterized algorithms.

**Dynamic Programming for Minimum Steiner Trees**

***Abstract***

We are here presenting a dynamic programming algorithm which solves the Minimum Steiner Tree Problems with k terminals with O\*(ck) time complexity for all c > 2. This improves the running time of the earlier fastest exponential time algorithms, which was Dreyfus-Wagner algorithm of order O\*(3k) and the so-called as “full set dynamic programming” algorithm, solving rectilinear instances in time O\*(2.38k).

***Introduction***

The Steiner tree problem is one of the most popular NP-hard problems. Given a graph G = (V, E) of order n = |V|, edge costs c: E → R+ and a set Y ⊆ V of k = |Y| terminals, we are to find a minimum cost tree T ⊆ E connecting all terminals. Note that, we identify a subtree of the underlying graph with its edge set T ⊆ E. The node set of the tree is denoted by V (T). So an Optimal Steiner Tree for Y is a tree T = T(Y) that minimizes the value of c (T) subject to Y ⊆ V (T).

***Algorithm ASC (“Attach Small Components”)***

1. For each Y˜, Y ⊆ Y˜ ⊆ V, |Y˜| = k + [1/e] do:
2. Compute T(X) for all X ⊆ Y ˜, |X| ≤ €k + 1.

For all X ⊆ Y˜, |X| > €k + 1, compute T(X) recursively, according to

T(X) = min {T(X1) ∪ T(X2)|X = X1 ✶ X2, |X2| ≤ €k + 1}.

There are O (n1/e) choices for Y˜ of size k˜ = k + [1/e]. The time needed for 1) (using Dreyfus-Wagner) is negligible for reasonably small € > 0. So the total running time is bounded by

n1/e Ei (k˜/i!) (i/€k + 1) ≤ n1/e k˜ 2k˜ ((k˜/2)/ €k˜). This yields our main result.

**Faster Steiner Tree Computation in Polynomial-Space**

By:- Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch

* Finding the best approximation algorithm for the Steiner tree problem has been a challenge for many mathematicians as there are very less number of research papers written on this topic.
* All the best known approximation algorithms for this problem are based on Dynamic Programming approach and require exponential time.
* Assuming the practical limitations of exponential-space algorithms, the authors of this paper decided to address designing of faster polynomial space algorithms.
* Their first contribution was a simple polynomial space O((27/4)^k.n^O(log k)) time algorithm which was top down recursive implementation of Dreyfus Wagner Theorem which is termed as Classical Algorithm for Steiner tree Problem.
* This approach leads to a very high running time because the main reason is that, by applying recurrence D&W, one generates some subproblems with almost the same number of terminals as in the original problem.[1]
* Later,this algorithm was refined to O((5.96)^k.n^O(log k)) in polynomial space based on a variant of classical tree separator problem for roughly k<=n/4.
* Finally,when the above algorithm(for small k) is combined with an improved branching strategy(for large k),an improved O(1.60^n) time polynomial space algorithm is obtained.
* This refined branching strategy is based on a charging mechanism which shows that, for large values of k, configurations of terminals and non-terminals must exist.
* As a corollary, exponential-space time complexity is improved from O(1.42^n) to O(1.36^n).
* At the ending of the paper, analysis of the proposed algorithms is done using Measure and Conquer approach which is based on quasiconvex analysis of multivariate recurrences.

References:-

[1] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch, Faster Steiner Tree Computation in Polynomial-Space,September’ 2008.

Fast polynomial-space algorithms using Mobius inversion: Improving on Steiner Tree and related problems

**Abstract** **:**

For a graph with n vertices and K terminals and bounded integer weights on the edges, we compute the minimum Steiner Tree in O(2k) time and polynomial space, and for NP-complete Spanning Tree and in some Steiner Tree cases we get the result in O(2n).

Previously we used to use Dynamic programming among subsets which was the fastest known algorithm for these problems, and require exponential space, but in this report, we have used Mobius inversion for recurrences, and Inclusion-Exclusion algorithm of Karp for counting Hamilton paths

**Introduction:**

We have been using dynamic programming for solving NP-hard problems, but unfortunately an exponential storage requirement seems to be inherent to this technique and is useless in practice. Therefore, we need more exact polynomial-space algorithms and in this report, we have improved some algorithms in such a way that they maintain the best known upper bound on the running time. Here we have introduced Inclusion-exclusion algorithm and Mobius inversion algorithm.

We show that some dynamic programming algorithms can be improved to obtain polynomial-space algorithms with the same worst-case running time.

Steiner Tree is one of the most well-studied NP-complete problems. The Dreyfus-Wagner dynamic programming algorithm has been the fastest exact algorithm for over 30 years. However, recently Bj¨orklund et al gave an O(2K)-time algorithm for the variant with bounded integer weights. Whereas Fuchs et al gave an O(CK)-time algorithm for the general case, for any c > 2.

**Inclusion – Exclusion formulations:**

Principle of Inclusion – exclusion: Let U be a set and A1, A2,,…An⊆ U. With the convention ∩i ∈∅ Ai = U, the following holds:

|∩ Ai| i∈ {1,2,3…n} = |∩ Ai | i∈X Eq1

U 🡪 Universe

A1, A2,,…An 🡪 requirements

We will call the task of computing |∩ Ai | i∈X  fro an arbitrary } the simplified problem. Note that if the simpliﬁed problem can be computed in polynomial time, there exists an O(2n)-time polynomial-space algorithm that evaluates Equation 1.

**Steiner Tree:**

A Steiner Tree is a tree with a given set of terminals K V(Vertices) and an integer c. In Steiner Tree we have to find the shortest path that covers all the given terminals.

Here is a table given below with different algorithms to solve Steiner Tree and their time complexity:

|  |  |
| --- | --- |
| Algorithm | Time Complexity |
| Dynamic Programming | O(2k) |
| Inclusion-Exclusion | O(2n) |
| Mobius Inversion | O(2n) |

We use Dreyfus-Wagner for dynamic programming, Inclusion-Exclusion for Hamilton paths, and Mobius Inversion for recurrences.

**Mobius Inversion:**

Mobius inversion states that if f and g are two arithmetic functions satisfying

g(n) = for every integer n>=1

then

f(n) = = for ever integer n>=1

where is the Mobius function

Here we will be introducing the Mobius inversion for the Steiner Tree. Basically it consist of the following of the two transforms:

1)Given a function f :2V → Z+ andY ⊆ V, the zeta transform ζf(Y ) and the Mobius transform µf(Y), are deﬁned as:

ζf(Y ) =

µf(Y) =

2)Folklore: The Mobius transform is the inverse of the Zeta transform; that is , for every Y ⊆ V , f(Y ) = µζf(Y ).

**Conclusion:**

We studied applications where the zeta transform is computable in polynomial time. As mentioned in the introduction, our algorithms considerably improve on dynamic programming in practice: in addition to improving the space requirement, our algorithms can potentially be made faster in practice when combined with other techniques. We want to mention that applying Mobius inversion to a problem is not straightforward: ﬁrst one has to come up with a function with the wanted properties, in order to successfully apply Mobius inversion.

**Approximating the Minimum Degree Spanning Tree to within One from the Optimal Degree**

**Martin Furer\* Balaji Raghavachari**

***Abstract***

The research paper describes an iterative polynomial time approximation algorithm for constructing a spanning tree which has the least maximal vertex degree among all the possible spanning trees for a graph (which is the problem of Minimum Degree Spanning Tree). The suggested algorithm produces a spanning tree whose maximal degree is at most O(Δ\* + log n). Here Δ\* is the maximal degree of a vertex in the optimal spanning tree. The result can be generalized to two cases- 1.) Steiner tree and 2.) directed graph. With further improvements the algorithm can construct a spanning tree of at most Δ\* + 1, which is the closest bound possible in polynomial time (by January 1992).

***Introduction***

There are various categories of spanning trees based on the complexity of the problem. The problem of computing a minimum spanning tree based on the edge weights can be solved in polynomial time. However, with even little constraint the problem can turn out to be NP-hard. The problem of Minimum Degree Spanning Tree (MDST) is a NP-hard problem.

MDST problem is the special case of finding a minimum Steiner Tree where all the vertices in the graph are terminal vertices and thus minimum degree spanning tree is a more general problem.

MDST is applied in mail and news distribution by sites on the Internet where distribution should not be done on a priority basis to reduce the work done by their site and cater large hits on their sites in a faster manner. It is also used to design efficient power grids, where the design should consist of least maximum degree because the cost of splitting the output of a station may grow largely with the degree of the split.

This paper shows that there is a polynomial time approximation algorithm for the minimum degree spanning tree problem (both directed and undirected), minimum degree Steiner tree problem which produces a Steiner tree of degree O(Δ\* + log n) and that there exist a polynomial time approximation algorithm for the minimum degree spanning tree of degree at most Δ\* + 1.

***A Simple Approximation Algorithm***

The following algorithm solves the problem in polynomial time.

**Steps:**

1. Start with an arbitrary spanning tree T of G. Let k be the maximal degree of T.
2. Try to reduce the degree of some vertex whose degree is between k and

k - ⌈ log n ⌉, using local “improvement” steps which can be implemented using polynomial time standard techniques for searching graphs.

**Improvement:** Consider an edge (u, v) of G which is not in T such that when added to T, it produces a cycle C. Suppose there is a vertex ‘w’ in C with the property that p(w) ≥ max{p(u), p(v)} + 2. The maximum of {p(u), p(v), p(w)} can be decreased by at least one adding the edge (u, v) and deleting one of the edges in C incident to w. This is called improvement.

1. Construct Si which contains all those vertices of T which have degree at least i.
2. The algorithm stops when no vertex in Si has a local improvement.

***The Steiner Problem***

The Minimum Degree Steiner Tree problem: Given a graph G(V, E) where V is the set of vertices and E is the set of edges and a set of vertices (called terminal vertices), D such that D ⊆ V. The goal is to find a tree which spans all the vertices in the set D and has a minimum degree.

**Steps:**

1. Start with an arbitrary tree T which spans D and retain only those edges which separate the set D in T.
2. Let W be the set of vertices spanned by T.
3. Generate Locally Optimal Steiner Tree (LOST) using the improvement mentioned below.

**Improvement for this problem:** A path between any two vertices in W which goes entirely through vertices of V - W except for the end points is called a non-tree path. Adding any non-tree path to T introduces a unique cycle. To again make it into a tree, we remove one edge of T from this cycle which is incident on a vertex of high degree, this decreases the degree of that vertex by one.

1. Repeat till none of the non-tree paths produce any improvement for any vertex in Si for i = k - ⌈ log n ⌉, where k is the maximal degree of the tree. If this property is true for all i, then the tree is a locally optimal Steiner tree.
2. The algorithm stops when no improvement can be done on the tree and all the vertices in D have been spanned.

***Directed Spanning Trees***

**Input:** Directed Graph G(V, E) and a special vertex ‘r’ which should be made the root of the tree in such a way that the root of the tree is accessible from all the vertices of the graph.

**Properties of a rooted spanning tree:**

1. The tree should not contain any cycle.
2. Every vertex has an out degree of exactly one except ‘r’.
3. All the vertices in the tree are connect to ‘r’ by a path.

The maximum indegree the vertices in the tree can be the degree of the root ‘r’.

**Steps:**

1. Start with an arbitrary rooted spanning tree T.
2. Consider a vertex v of indegree i.
3. The goal is to reduce the indegree of any vertex by attaching one of the ‘i’ subtrees of ‘v’ to another vertex of smaller degree.

**Improvement step:** It consists of two parts. We first move the root of the subtree T’ that is being removed from ‘v’ to a “convenient” vertex in that subtree. T’ is then attached to another vertex outside the tree to which the new root has a connection.

1. Here the set of convenient vertices are those in the strongly connected part of the root of T’ in the graph induced by the vertices of T’ with all non-tree edges removed from vertices of degree i - 1 or greater.
2. We define locally and pseudo optimal directed spanning trees as before.
3. The algorithm tries to decrease the degrees of vertices in Si for

i = k - ⌈ log n ⌉ using the improvement step above.

***Conclusion***

Th paper demonstrates iterative algorithms for approximating the minimum degree spanning tree and related problems. However, the exact complexities of some of the suggested algorithms are not clear.

***References***

1. Martin Fürer, Balaji Raghavachari: Approximating the Minimum Degree Spanning Tree to Within One from the Optimal Degree. SODA 1992: 317-324
2. Fast polynomial-space algorithms using Mobius inversion: Improving on Steiner Tree and related problems ? Jesper Nederlof
3. Sharp Separation and Applications to Exact and Parameterized Algorithms Fedor V. Fomin · Fabrizio Grandoni · Daniel Lokshtanov · Saket Saurabh
4. Faster Steiner Tree Computation in Polynomial-Space Fedor V. Fomin1?, Fabrizio Grandoni2??, and Dieter Kratsch
5. Solving Connected Dominating Set Faster than 2n ∗ Fedor V. Fomin†. ‡ Fabrizio Grandoni§ Dieter Kratsch
6. Dynamic Programming for Minimum Steiner Trees B. Fuchs a W. Kern b D. M ̈olle c S. Richter c P. Rossmanith c X. Wang b,1