A New Efficient Approximation Algorithm for Chromatic Number

Guillermo De Ita[1], Meliza Contreras[2] Erica Vera[1]

[1] Universidad Autónoma de Puebla, deita@cs.buap.mx, ee_vera@hotmail.com
[2] Universidad Angelópolis, mel_22281@hotmail.mx

Abstract. We design a new approximation polynomial-time algorithm for the graph coloring problem. Our proposal is based on selecting, in iterative manner, a critical vertex v of the graph. The criterion to select is based on choose the node with maximum degree and with maximum degree of its neighborhood into the set of vertices composing odd cycles. The algorithm consists of two embedded loops. While in the internal loop a critical node is selected to be colored and it is deleted as well as its upon edges from the current graph. The external loop controls when there is not possible to select more vertices and, while remains odd cycles in the current graph, new colors are used. The stop criterion to finish the two loops is when the current subgraph is bipartite.
Our algorithm establishes an average number of $(\sqrt{4\delta+1}+2\delta-1)/2$ colors for approximate the chromatic number of any graph G, δ being the initial average degree of the input graph G.

1 Introduction

The problem to determine the minimum value of colors needed for coloring a graph is a NP-complete problem, even for graphs G with degree (maximum degree) $\Delta(G) \geq 3$. A consequence of this is that there is not a complete theoretical characterization of colorability [8].

The graph coloring problem is an abstraction of certain types of scheduling problems. In the graph k-coloring problem we wish to assign each vertex one of k colors such that every pair of vertices connected with an edge are assigned different colors. The chromatic number of a graph G denoted by $\chi(G)$ is the minimum value k such that G has a k-coloring. This problem arises in a host of applications, and was one of the 22 NP-complete problems on Karp's list. Subsequently, much effort was spent on trying to design efficient approximation algorithms, namely, given a k-colorable graph to try to color it with as few colors as possible [1].

One of the first bound to color a 3-colorable graph was established by Wigderson [13], he showed how to color 3-colorable graphs with at most $3 \cdot \lceil \sqrt{n} \rceil$ colors, where n is the number of nodes on the graph. Blum and Karger [3] applied semidefinite programming (SDP) to improve this bound to $O(n^{3/14})$, where the notation O is used to suppress polylogarithmic factors. Recently, Arora, et. al. [1] using stronger SDPs have improved the bound to $O(n^{0.2111})$.

© S. Torres, I. López, H. Calvo. (Eds.)
Advances in Computer Science and Engineering
Research in Computing Science 27, 2007, pp. 113-122

Received 15/02/07
Accepted 08/04/07
Final version 22/04/07

On the other hand, no polynomial-time algorithm is known for coloring any graph where the ratio of the number of colors used to the optimal number is bounded by a constant. In fact, guaranteeing a small constant bound on the ratio is NP-Hard [2]. For example, is known that if there were an approximation polynomial-time algorithm that uses fewer than $(4/3)\chi(G)$ colors, then the 3-colorability problem could be solved in polynomial time and then, $NP = P$.

For unrestricted number of colors we know that a graph coloring can not be approximated with ratio n^{ϵ} for some ϵ; the current value for the exponent ϵ is $1/10$ [4]. However, the upper bound that is achieved by the known approximation algorithm, i.e. the proposal in [1], is still far to the theoretical bound.

We present in this article, a polynomial-time algorithm which uses an average number of $(\sqrt{4\delta+1}+2\delta-1)/2$ colors for approximate $\chi(G)$, δ being the initial average degree of the input graph G.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with vertex set (or nodes set) V and set of edges E. Two vertices v and w are called *adjacent* if there is an edge $\{v, w\} \in E$, joining them. Sometimes, we denote with $E(G)$ and $V(G)$ rather than E and V to emphasize that these are the edges and vertex sets of a particular graph G. The *Neighborhood* of $x \in V$ is $N(x) = \{y \in V : \{x, y\} \in E\}$ and its *closed neighborhood* is $N(x) \cup \{x\}$ which is denoted by $N[x]$. Note that v is not in $N(v)$.

We denote the cardinality of a set A, by $|A|$. Given a graph $G = (V, E)$, the degree of a vertex $x \in V$, denoted by $\delta(x)$, is $|N(x)|$. The size of the neighborhood of x, $\delta(N(x))$, is $\delta(N(x)) = \sum_{y \in N(x)} \delta(y)$. The maximum degree of G or just the degree of G is $\Delta(G) = max\{\delta(x) : x \in V\}$, while we denote with $\delta_{min}(G) = min\{\delta(x) : x \in V\}$ and with $\delta(G) = (2 \cdot |E|)/|V|$ the average degree of the graph.

Given a subset of vertices $S \subseteq V$ the subgraph of G denoted by $G|S$ has vertex set S and set of edges $E(G|S) = \{\{u, v\} \in E : u, v \in S\}$. To $G|S$ is called the *subgraph of G induced by S*. We write $G - S$ to denote the graph $G|(V - S)$. The subgraph induced by $N(v)$ is denoted as $H(v) = G|N(v)$ which has to $N(v)$ as the set of nodes and all edges upon them.

A path from a vertex v to a vertex w in a graph is a sequence of edges: $v_0v_1, v_1v_2, \ldots, v_{n-1}v_n$ such that $v = v_0$ and $v_n = w$ and v_k is adjacent to v_{k+1} for $0 \leq k < n$ and, the length of the path is n. A simple path is a path such that $v_0, v_1, \ldots, v_{n-1}, v_n$ are all distinct. A cycle is just a nonempty path such that the first and last vertices are identical, and a simple cycle is a cycle in which no vertex is repeated, except that the first and last vertices are identical.

A k-cycle is a cycle of length k, that is, a k-cycle has k edges. A cycle of odd length is called an odd cycle, while a cycle of even length is called an even cycle. A graph G is acyclic if it has not cycles.

A *complete graph* of n nodes has $n(n+1)/2$ distinct edges, we denote K_n the complete graph of n nodes. A graph G is a *regular graph* if all vertices have the same degree, G is $k-$*regular* if it is regular, of degree k.

A connected component of G is a maximal induced subgraph of G, that is, a connected subgraph which is not a proper subgraph of any other connected subgraph of G. Note that, in a connected component, for every pair of its vertices x, y, there is a path from x to y. If an acyclic graph is also connected, then it is called a free tree.

A coloring of a graph $G = (V, E)$ is an assignment of colors to its vertices. A coloring is *proper* if adjacent vertices always have different colors. An k-coloring of G is a mapping from V into the set $\{1, 2, \ldots, m\}$ of k "colors". The chromatic number of G denoted by $\chi(G)$ is the minimum value k such that G has a proper k-coloring. If $\chi(G) = k$, G is then said to be k-chromatic. The problem to determine the value $\chi(G)$ is polynomial computable when $\chi(G) \leq 2$, but when $\chi(G) \geq 3$, the problem becomes NP-complete, even for graphs G with degree $\Delta(G) \geq 3$.

Given a graph $G = (V, E)$, $S \subseteq V$ is an independent set in G if for whatever two vertices v_1, v_2 in S, $\{v_1, v_2\} \notin E$. Let $I(G)$ be the set of all independent sets of G. An independent set $S \in I(G)$ is *maximal* if it is not a subset of any larger independent set and, it is *maximum* if it has the largest size among all independent sets in $I(G)$.

Let $G = (V, E)$ be a graph, G is a *bipartite graph* if V can be partitioned into two subsets U_1 and U_2, called *partite sets*, such that every edge of G joins a vertex of U_1 and a vertex of U_2.

If $G = (V, E)$ is a k-chromatic graph, then it is possible to partition V into k independent sets $V_1, V_2, ..., V_k$, called *color classes*, but it is not possible to partition V into $k - 1$ independent sets.

3 Polynomial Coloring Procedures

The main class of graphs which is known to be colored in polynomial time is the class of bipartite graphs.

Lemma 1 *A graph G has chromatic number 2 if and only if G is bipartite.*

Since the partite set U_1 can be colored with the first color while the other partite set U_2 is coloring with the second color. Furthermore, as the bipartite property can be recognized in polynomial time, then a 2-colorable graph can be recognized in polynomial time, based on the following property.

Lemma 2 *A graph G is bipartite if and only if G contains no odd cycles.*

Given an input connected graph $G = (V, E)$, if we apply the depth-first search over G starting the search, for example with the node $v \in V$ of minimum degree, we obtain a depth-first graph T_G, which we will denote as $T_G = dfs(G)$. If G is an acyclic graph, we call to T_G the spanning tree of G, and in this case, we can color T_G with only two colors. We color the vertices in T_G by levels, in alternating way between the two colors in accordance with the change of level, that is, all vertices in same level have same color.

The depth-first search also allow us to detect the odd cycles of a graph. We present, in a schematic way, a recursive procedure for the depth-first search. We show in the following procedure, the different status of a node during the search. At the beginning, every node is not discovered and, when a node u as well as all its adjacent nodes have been visited then u is marked as finished.

Procedure dfs(v)

1. Mark v as discovered
2. For each node $w \in N(v)$
   (a) If w is not discovered then $dfs(w)$
   (b) else mark the edge $\{w, v\}$ as a *back edge*
3. Mark v as finished
4. Returns

The procedure dfs runs in time $O(m+n)$ where n and m are the number of nodes and the number of edges of the input graph G, respectively. Thus, dfs is a linear-time procedure over the length of G which can be used for coloring a graph.

Each *back edge* that we find during the depth-first search marks the beginning and the end of a base cycle (or fundamental cycle). Let $C = \{C_1, C_2, ..., C_k\}$ be the set of fundamental cycles found during the depth-first search. Each back-edge $c_i \in T_G$ determines a base cycle $C_i \in C$, $i = 1, \ldots, k$.

If T_G is acyclic or contains only even fundamental cycles, then T_G is bipartite and it is colorable with just two colors. Like the case when T_G is a tree, we could advance visiting nodes in T_G by levels assigning to each node of the same level the same color. The two colors are applied in an alternating way in accordance with the change of level.

Given two distinct base cycles C_i and C_j from C, if C_i and C_j share common edges we say that both cycles are *intersected*, that is, $C_i \oplus C_j$ conforms a new cycle, where $\oplus$ denotes the operation or-exclusive between the set of edges of the cycles. If the cycles C_i and C_j have no common nodes nor edges then we say that both cycles are *independent*, that is, $C_i \oplus C_j = C_i \cup C_j$. While if the cycles have not common edges but maybe they have a common node, we say that both cycles are *non-intersected*. Note that a pair of independent cycles are non-intersected.

Also is known, that any simple odd cycle request of three colors to be colored. Then, we can recognize in polynomial time a set of graphs which are 3-colorable. We present in the following section the prerequisites so that a graph is 3-colorable.

3.1 A 3-Coloring Algorithm

Theorem 1 *If T_G, the resulting depth-first graph of an input graph G contains only non-intersected base cycles, then $\chi(G) \leq 3$ and a 3-coloring is done in linear time.*

Proof: we present as proof the following linear time algorithm.

In order to recognize if a graph G is 3-colorable, we apply $T_G = dfs(G)$ and test if T_G has one of the following base cases:

1. If T_G has not odd fundamental cycles then G is 2-colorable. This option considers the case where T_G is a bipartite graph.
2. If any base cycle in T_G includes odd cycles, but such odd cycles are non-intersected with any other cycle of T_G then T_G is 3-colorable.
   We can color T_G by levels, but using the third color for coloring each end-node of every odd cycle. The end-node of a cycle is the node where the back edge was found during the depth-first search.
3. We color embedded cycles from the most intern to extern cycle. When we start to color a new cycle, we use a different color to their neighboring nodes (at most two neighboring nodes), since we consider that only the nodes in the internal cycles have already been colored. In this case, we can consider to T_G like an instance of a *Series-Parallel graph*, which is know that is colorable in polynomial time [9].

Then, if the topological structure of the input graph G is whatever of the latter basic cases, then G is 3-colorable and it is coloring in polynomial time. Thus, the class of graphs that they have the previous basic topologies, conforms us a new polynomial class of graphs for the 3-colorable problem.

We present in the following section a polynomial time algorithm based on the selection of the critical nodes to approximate the chromatic number.

4 A Polynomial Approximation Algorithm for the Chromatic Number of a Graph

First, we review some results that we will use to design our proposal, as well as the knowing upper bounds for the chromatic number of a graph.

Lemma 3 *If G is a connected non-regular graph, then $\chi(G) \leq \Delta(G)$.*
If G is regular, then $\chi(G) \leq \Delta(G) + 1$.

Proof. First, assume that G is non-regular, choose a vertex v_1 of minimum degree in V as the root of a spanning tree. We traverse the resulting graph in preorder (first the child nodes and after the parent node). When each vertex $v_i \neq v_1$ comes to be colored, the parent of v_i has not already been colored. Therefore at most $\delta(v_i) - 1$ adjacent vertices have already been colored. Hence $\chi(v_i) \leq \Delta(G)$. When v_1 comes to be colored, all adjacent vertices have already been colored. Since $\delta(v_1) < \Delta(G)$, we conclude that $\chi(v_1) \leq \Delta(G)$. Hence $\chi(G) \leq \Delta(G)$.

If G is a regular graph, then the proof proceeds as above, except that $\chi(v_1) \leq \Delta(G)+1$. The conclusion is followed. Notice that if G is regular, only one vertex needs to use color $\Delta(G) + 1$.

If a depth-first search is applied for ordering the nodes of a complete graph K_n, and after we color the nodes of the spanning tree in preorder, then it could be shown that it is needed to use exactly n colors.

Given a graph $G = (V, E)$ and a node $v \in V$, the subgraph induced by $N(V)$ is denoted as $H(v)$; recalls that it consists of $N(V)$, the neighborhood of v, and all edges in E between vertices of $N(v)$.

Lemma 4 *If G is k-colorable, then for any $v \in V$, $H(v)$ is $(k-1)$-colorable.*

Since 2-colorable graphs can be identified and colored (with only two colors) in polynomial time, the neighborhood of any vertex in a 3-colorable graph can be colored with two colors in polynomial time.

Let $G = (V, E)$ be a graph with $\chi(G) = m$. If we remove an edge $\{u, v\}$ from G, there are two possibilities, either $\chi(G-\{u,v\}) = m$ or $\chi(G-\{u,v\}) = m-1$. In the latter case, we say that the edge $\{u, v\}$ is critical. If a node u has a critical edge upon it, then we extend the definition and the node u is critical too.

A graph G is critical if $\chi(G - \{u, v\}) = \chi(G) - 1$ for all edges $\{u, v\} \in E$. If $\chi(G) = m$, we say that G is m-critical.

It is easy to see that every graph G contains a critical subgraph. If $\chi(G - \{u, v\}) = \chi(G)$ for some edge $\{u, v\} \in E$, we can remove such edge. Continuing deleting edges like this until every edge is critical. The result is a critical subgraph.

According to those latter results, we design the following algorithm.

Let $G = (V, E)$ be a graph with $|V| = n$, $|E| = m$, and let T_G be the graph generated for the depth-first search over G.

The general strategy of our proposal for coloring G consist on:
First: To recognize the general topology of G. This is done for applying a depth-first search on the graph.
Second: Test if G can be colored with two colors, a linear procedure is executed for detecting this condition
Third: If G is not 2-colorable, we detect the node v which is part of an odd cycle and with maximum conflicting for coloring v and the odd cycles in G. That is done by executed a polynomial time procedure.
Fourth: We color v with the active color, v and its edges upon it are deleted from the current graph. And the control is returned to the first step.

This procedure is executed in iterative way while the current graph can not be recognized as a bipartite graph. We show the pseudo-code of this proposal.

Procedure Select_Candidate_Node(T_G,NV)
Input: T_G is a subgraph, NV is the neighborhood over the vertices that can not be colorable with color k

Output: $v \in V$ **a vertex to be coloreable**
Procedure

1. $Vertices = minus(T_G, NV)$; /* Computes $Vertices = V(T_G) - NV$ */
2. choose $v \in Vertices$ such that /* v is a critical node */
3. degree(v) and degree($NV(v)$) are maximum over the set of odd cycles nodes
4. Otherwise /* If every odd cycle was covered by NV */
5. choose $v \in Vertices$ such that /* v has maximum degree in T_G */
6. If (maximumOver_$TG(degree(v), T_G)$ == true) then
7. Return v

Algorithm Seek_Chromatic_Number(G)
Input: G a non directed graph
Output: An approximate value for $\chi(G)$
Procedure

$k = 3$; /* Starting with the class color $k = 3$ */
$T_G = dfs(G)$; /* The nodes of the graph are ordered */
1) while(is_bipartite(T_G)==false) /* While there is an odd cycle in G */
{$NV = \emptyset$; /* NV is the neighborhood for the class color k */
2) while (is_subset(NV, $V(T_G)$)==true)
a) { $v = Select\_Candidate\_Node(T_G, NV)$;
b) $Color(v) = k$; $delete(T_G, v)$;
$add(NV, N(v))$; /* $NV = NV \cup N(v)$
c) H =Max_Component(T_G); /*If T_G is disconnected then consider the component with maximum value for $\chi(T_G)$*/
d) $T_G = dfs(H)$; /* Maintain ordered the remaining nodes */
} k++;
}
3) Call 2-Coloring(T_G); /*At the end, the remaining graph is 2-colorable*/
Return

The procedure *Seek_Chromatic_Number* consist of two embedded loops. In each iteration of the external loop, an independent set of class color k is built. This class color is formed by the critical nodes of the current graph and compose an independent set I_k of the graph. Each node in a independent set I_j is colored with the color $j + 2$, like it is showed in first and second graph in Figure 2.

The internal loop is applied to find each critical node v of the current graph and for building the neighborhood $NV(v)$ for the independent set I_j where v is, like it is showed in second graph in Figure 1. When a critical node is detected, it is colored and deleted from the graph as well as incident edges upon it; as T_G is changing in each iteration of the internal loop, it might even be disconnected, then it is necessary to order the nodes using the depth-first search again.

The external loop finishes when the remaining subgraph T_G is bipartite, and then T_G is 2-colorable, like it is showed in third graph in Figure 2.

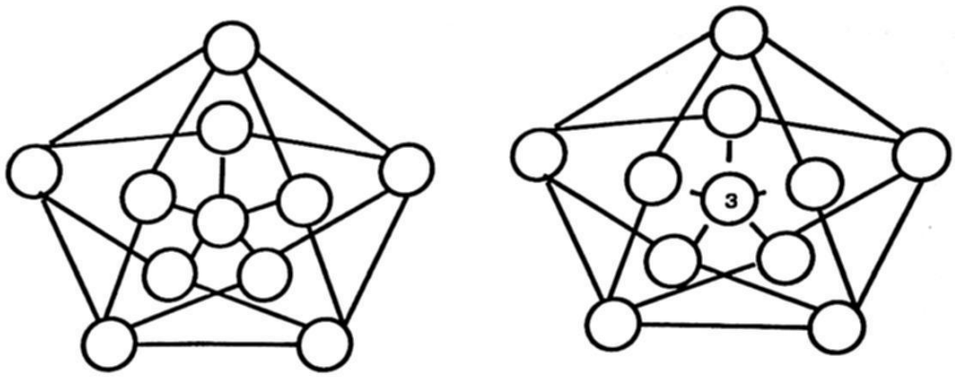

Fig. 1. Executing the algorithm over the Grötzsch graph

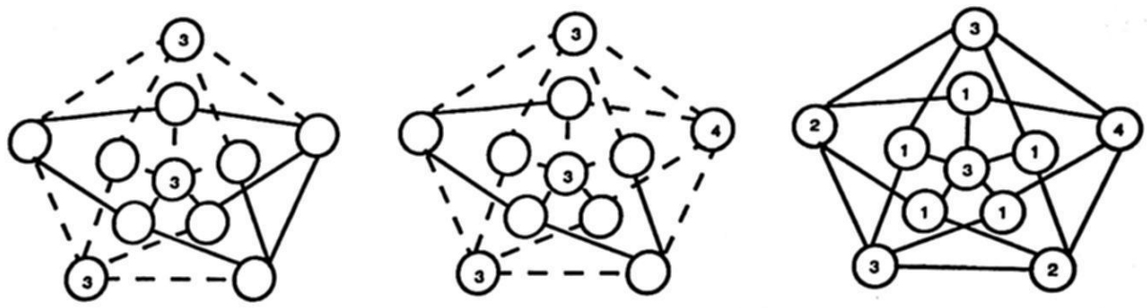

Fig. 2. Three iterations of the main loop

5 Complexity Analysis

The most expensive operation into the internal loop (step 2) is to determine the critical node v to be colored, line (a). This step is performed in time $O(m * n)$, and as that loop iterates at most n times, then the complexity time of the internal loop is $O(m * n^2)$. While the external loop (step 1) is executed $(m - n)$ times in the worst case (when all cycle in G is odd), and the most expensive instruction is the internal loop, so the total complexity time of the procedure is polynomial and it is $O((m - n) * m * n^2)$.

We focus ourselves on determining the average number of colors used by our algorithm, rather than the worst-case behavior. Let $rel = m - n$ be the variable used to denote the relation between the number of edges and nodes in whatever subgraph. The variable rel denotes the number of cycles minus 1 in a connected graph. We know that for any graph H, if $m \leq n$ then H is bipartite or H is a simple odd cycle and then H is 3-colorable. We analyze the value that rel_i takes in each subgraph $T_{G_i} \subset G$ generated after of the iteration i by the main loop (external loop), and let K_i be the independent set built in such iteration.

Given an initial graph $G = (V, E)$ with $|V| = n_0$ and $|E| = m_0$ in each iteration of the main loop the number of nodes and edges are updated as: $n_{i+1} = n_i - |K_i|$ and $m_{i+1} = m_i - |E(K_i)|$, since in each iteration the nodes in K_i are deleted to the current graph as well as its incident edges: $E(K_i)$.

Let $G_{i+1} = G_i - K_i$ be the remaining subgraph generated from G_i after to finish the iteration i into the main loop. As each K_i is an independent set of G_i, there are not edges of G_i connecting any two nodes of K_i. Furthermore, the edges in $E(K_i)$ cover every node of G_i, that is, $E(K_i)$ is an edge cover of G_i, then we have that $|E(K_i)| \geq n_i$ and the number of remaining edges in G_{i+1} is $m_{i+1} = m_i - |E(K_i)| \geq m_i - n_i$, so $m_i - n_i$ is an upper bound for m_{i+1}.

The behavior between the number of edges and number of nodes for each subgraph G_i, hands:

$rel_0 = m_0 - n_0$.
$rel_1 = m_1 - n_1 \leq (m_0 - n_0) - n_1 = (m_0 - n_0) - (n_0 - |K_0|) = m_0 - 2n_0 + |K_0|$.
$rel_2 = m_2 - n_2 \leq (m_1 - n_1) - (n_1 - |K_1|) = m_0 - 2n_0 + |K_0| - (n_0 - |K_0| - |K_1|) = m_0 - 3n_0 + 2|K_0| + |K_1|$.
$rel_3 = m_3 - n_3 \leq (m_2 - n_2) - (n_2 - |K_2|) = m_0 - 3n_0 + 2|K_0| + |K_1| - (n_0 - |K_0| - |K_1| - |K_2|) = m_0 - 4n_0 + 3|K_0| + 2|K_1| + |K_2|$.
...
$rel_k \leq m_0 - (k+1)n_0 + k|K_0| + (k-1)|K_1| + \ldots + |K_{k-1}|$.

An average value for $|K_i|, i = 0, \ldots, k-1$ is computed for considering that $\sum_{v \in K_i} \delta(v) \approx \sum_{j=1}^{|K_i|} \delta(G_i) \geq n_i$, where $\delta(G_i)$ is the average degree of the subgraph G_i, and then $|K_i| \cdot \delta(G_i) \geq n_i$ so $|K_i| \geq n_i/\delta(G_i) \approx n/\delta$, being δ the average degree of the initial graph G. Thus, we can approximate the value $|K_i|$ by n/δ.

Then, $rel_k \leq m_0 - (k+1) \cdot n_0 + \sum_{i=0}^{k-1}(k-i) \cdot (n/\delta) = m_0 - (k+1)n + (n/\delta) \cdot ((k)(k+1)/2)$. And we want to know the average number of iterations for the external loop until arrive that $rel_k \leq 0$. So, we want to determine the average value for k where $rel_k \leq 0$, that is, $m_0 + (k(k+1)/2) \cdot (n/\delta) \leq (k+1)n_0$.

Let $m = m_0$ and $n = n_0$. As $m = (\delta \cdot n)/2$ then

$[(\delta \cdot n)/2 + (n/\delta) \cdot (k(k+1))/2]/n \leq k+1$, so $\delta + (k(k+1))/\delta \leq 2 \cdot (k+1)$. Thus,

$$\delta/(k+1) + k/\delta \leq 2 \tag{1}$$

In order to solve (1) we express the equation in terms to the variable k, obtaining: $0 \leq -(1/\delta)k^2 + (2 - (1/\delta))k + (2 - \delta)$ and factoring the polynomial in k, we have: $0 \leq -(1/\delta)(k + (\sqrt{4\delta+1} + 1 - 2\delta)/2)(k - (\sqrt{4\delta+1} + 2\delta - 1)/2)$, and the interval where the value for k hands the equation (1), is: $-(\sqrt{4\delta+1} + 1 - 2\delta)/2 < k < (\sqrt{4\delta+1} - 1 + 2\delta)/2)$. Thus, the maximum value for k handing (1) and which represents the average number of colors used for our algorithm, is: $\chi(G) \approx (\sqrt{4\delta+1} + 2\delta - 1)/2$.

Hence our procedure uses $\tilde{O}(\delta)$ colors (where the notation $\tilde{O}$ is used to suppress polylogarithmic factors) for coloring the input graph G, being δ the initial average degree of the graph G.

6 Conclusions

First, we show a new polynomial class of graphs for the 3-coloring problem which includes to the *Series-Parallel graphs* like instances of this class. After, we show a new approximation polynomial-time algorithm for determining the chromatic number of a graph G. Our proposal is based on selecting, in iterative manner, a critical vertex v of the graph. The criterion to select is based on choose the node with maximum degree and with maximum degree of its neighborhood into the set of vertices composing odd cycles.

Our algorithm establishes an average number of $(\sqrt{4\delta+1}+2\delta-1)/2$ colors for approximate the chromatic number $\chi(G)$ for any input graph G, being δ the average degree of the graph G. Hence our procedure uses $\widetilde{O}(\delta)$ colors for coloring an input graph G.

References

1. S. Arora, E. Chlamtac, M. Charikar, New Approximation Guarantee for Chromatic Number, *Proceedings STOC 2006*, May 2006.
2. Baase S., Gelder A. V., *Computer algorithms: Introduction to Design & Analysis*, Addison Wesley, 2000.
3. Blum A., Karger D., An $O(n^{3/14})$-coloring algorithm for 3-colorable graphs, *Information Processing Letters*, 61(1):49-53, 1997
4. Bellare M., Sudan M., Improved Non-approximability results, Draft, 1993.
5. Dyer M., Greenhill C., *Some #P-completeness Proofs for Coulorings and Independent Sets*, Research Report Series, University of Leeds, 1997.
6. Dyer M., Greenhill C., Corrigendum: The complexity of counting graph homomorphism, *RSA: Random Structures and Algortihms*, 25:346-352,2004.
7. Greenhill Catherine , The complexity of counting colourings and independent sets in sparse graphs and hypergraphs, *Computational Complexity*, 1999.
8. Kocay W., Kreher D., *Graphs, Algorithms, and Optmization*, Chapman & Hall/CRC Press, 2005.
9. Johnson D., The NP-Completeness Column: An Ongoing Guide, *Jour. of Algorithms* 6,434-451, 1985.
10. Roth D., On the hardness of approximate reasoning, *Artificial Intelligence 82*, (1996), 273-302.
11. Russ B., *Randomized Algorithms: Approximation, Generation, and Counting*, Distingished dissertations Springer, 2001.
12. Vadhan Salil P., The complexity of Counting in Sparse, Regular, and Planar Graphs, *SIAM Journal on Computing*, Vol. 31, No.2, (2001), 398-427.
13. Wigderson A. Improving the performance guarantee of approximate graph coloring, *Jour. of the ACM*,30 (4):729-735, 1983.